Adrian Rezus

Meyhorst 91-01
6537 KJ Nijmegen
The Netherlands.

C L A S S I C A L P R O O F S

(Lambda Calculus Methods in Elementary Proof Theory)

Nijmegen, August 5, 1990

1. Introduction. During the early thirties, Haskell B. Curry has noticed a certain *correspondence* subsisting between the behavior of *stratified systems of combinators* and *axiomatic presentations* of the concept of *formal deduction* for the *positive implication* of Hilbert.

The latter is, in fact, a tiny fragment of classical propositional logic and coincides with the pure implicational part of Heyting's [1930] first-order logic **HQ**.

Curry's observation is the core of what has become known since as the *propositions-as-types isomorphism*.

> The principle behind the isomorphism has been re-discovered by nearly everybody else concerned with typed λ-calculi, systems of typed/stratified combinators and/or the proof-theory of Heyting's logic. A few significant names, in historical order, are: F. B. Fitch (1934), C. A. Meredith (± 1951), H. Läuchli (1956), W. A. Howard (1968), N. G. de Bruijn (1969-70), D. S. Scott (1969-70), P. Martin-Löf (1970-71), J.-Y. Girard (1970-71), etc.

Roughly speaking, the isomorphism associates, in a specific *formal setting* and for a preferred *semantics* of the logical operations:

- propositions/formulas ↔ types/type-expressions,
- proofs/proof-terms ↔ constructions/λ-terms (combinators)

and can be also understood as a way of characterizing *proofs* rigorously, from an *operational* viewpoint.

More recently, such a characterization has become a basic tool in work on computer implementations of proof-systems.

This technique has been known to work for (and has been applied to) so-called "constructive" cases, exclusively, as, e. g.,

- the first-order Heyting logic **HQ** (H. Läuchli, W. A. Howard),
- the Heyting propositional logic with *propositional quantifiers* (J.-Y. Girard) and
- a formalized variant of Bishop's "constructive mathematics", better known as Constructive Type Theory (P. Martin-Löf).

As a matter of fact, the *propositions-as-types isomorphism* has been, so far, just the name for a *formal variant* of the so-called **BHK**-*interpretation* (Brouwer-Heyting-Kolmogorov) of the Heyting proof-operations (the proof-operations involved in **HQ**, say; cf. e. g., [Troelstra & van Dalen 1988]).

Less known is a slightly more general point of view, advocated by Carew A. Meredith (a former student of Jan Łukasiewicz in Dublin; see [Meredith 1977]), around 1951-56. Unfortunately, it has never been formulated explicitly in print, except for a few practical hints concerning a *general combinatory proof-notation* for *purely implicational systems of logic* (see, e. g., [Meredith & Prior 1963], [Rezus 1982], [Kalman 1983], [Meredith & Hindley 1990] or [Bunder & Meyer 199], for details).

Meredith's standpoint allows to accommodate, in a stratified/typed combinatory setting, *arbitrary* concepts of a formal proof (i. e., not necessarily "intuitionistic"). Also, for well-behaved cases, the combinatory presentations admit of an alternative, equivalent typed λ-calculus variant.

In this book, we extend the technique based on Meredith's ideas to the case of *first-order classical logic* **CQ**.

> A number of extensions to logics containing **CQ** (as, e. g., most Lewis-style *modal* logics, as well as other first-order "intensional" logics with genuinely "classical" features) are also possible and will be discussed elsewhere.

Technically, the *elementary proof theory of first-order (classical) logic* is considered here as a *typed λ-calculus* or, equivalently, as a *stratified system of "proof-combinators"* ["combinatory logic"].

The resulting formalism appears to be a proper (in fact, conservative) extension of the ordinary typed λ-calculus **λ**. The latter one is just a formalized/type-disambiguated variant of Curry's *basic theory of functionality* ([Curry *et al.* 1958,1972], [Hindley & Seldin 1986]).

Moreover, the extension is shown to preserve the desirable properties of a typed λ-system, viz.

- *consistency* (here: *non-triviality of proof-equality in first-order classical logic*),
- *strong-normalization* (here: *termination for the processses of proof-détour elimination* or *proof-reduction*) and
- *confluence of proof-reduction in the "core" proof-theory of classical logic* (here: *definiteness of processes of proof-détour eliminations in a "type-normal" fragment of* **CQ**).

From this we have at least two (related) groups of consequences.

Theoretically, we display *rigorous versions* of both a "*natural deduction*" and an "*axiomatic*" point of view on *proving in first-order classical logic*, matching resp. the typed λ-calculus and the stratified "combinatory logic" presentations. If compared with this approach, the traditional (Gentzen/Hilbert-style) ways of coping with the same problems appear as *loose ways of speaking*, both *redundant* (i. e., introducing irrelevant data), on the one hand, and *elliptic* (i. e., operationally under-specified), on the other.

The purely *operational side* of the approach can be best summarized by saying that attempted, essentially, is an answer to the question **what is a classical proof?** as if posed by a *programmer*. It should become quickly obvious that, e. g., an *interpreter* for the proof-languages introduced in this book presupposes essentially the same implementation philosophy as an interpreter/compiler for the so-called *functional programming languages*.

> In this respect, there is here much ressemblance with some work on the **Automath** family of language, also *operationally oriented* ([de Bruijn 1980], [Rezus 1983], [Barendregt & Rezus 1983]). As distinctive features, one could note that **[1°] Automath** is a *more comprehensive* enterprise, concerned with the formalization and the automatic proof-checking of the *actual mathematics texts* in terms of computer languages. The *logic concern* is subordinated to this *basic pragmatic attitude*. Also **[2°] Automath** is bound to adopt a *relativistic point of view on proving*: we can, in fact, formalize a text (an actual piece of mathematics) in **Automath** starting *at any one of its points* and organize the amount of logic actually needed in the formal reconstruction of the text as a separate, *ad hoc* module (an **Automath**-"book"). So **[3°]** there is no need to be concerned explicitly with theoretical matters, as, e. g., characterizing the *completeness* of a particular choice of proof-primitives: what matters in **Automath** is the formal correctness of the result (the last line of a "book"). On the technical side, the *conceptual background* required here is virtually the same as that presupposed by *Abstract* **Automath** [Rezus 1983] and the Constructive Type Theory of Martin-Löf (as presented in, e. g., [Rezus 1986]).

Abstract and technical summary. The elementary proof-theory of first-order classical logic ([Gentzen 1935], [Prawitz 1965,1971]) is formalized here as a *typed* λ-*calculus* $\lambda\mathfrak{J}!$ (sections **3**, **5**). The calculus is equationally equivalent to a *combinatory theory* **C[CQ]** (sections **2**, **8**). Both $\lambda\mathfrak{J}!$ and **C[CQ]** are *extensional* (in the usual λ-calculus sense).

Formally, $\lambda\mathfrak{J}!$ contains the $[\supset,\forall]$-fragment of the Heyting proof-calculus, here: $\lambda!$. This is just a formalized/type-disambiguated variant of the Curry-Seldin *theory of generalized functionality* (cf., e. g., [Hindley & Seldin 1986]), with the usual (β-η-type) reduction/conversion rules (the $\forall$-proof-syntax duplicates, in a sense, the "$\mathfrak{J}$-free" $\supset$-proof-syntax at a first-order level).

The essential ingredient (over the $\lambda!$-syntax), used in order to accommodate genuinely classical principles of reasoning is a *new abstraction operator* $\mathfrak{J}$. This is intended to formalize *reductio ad absurdum* or the *principle of indirect reasoning*.

With $\bot$ (*falsum*) in the primitive provability/type-syntax, classical negation is introduced "inferentially", by $A^- := A \supset \bot$.

Then the stratification induced by *reductio ad absurdum* can be expressed by a typing rule:

$(\supset i\not\delta)$: $\quad \Phi[\ x : A^- \ ] \vdash a[\![x]\!] : \bot \Longrightarrow \Phi \vdash \not\delta x{:}A^-.a[\![x]\!] : A,$

where Φ is any "proof-context" [= assumption-set].

From an intuitive point of view, the (β-type) *reduction rules* for $\not\delta$ (the "$\beta\not\delta$-rules" of the calculus) are intended to measure the complexity of a given application of *reductio ad absurdum*. The associated "$\not\delta$-extensionality" assumption (meant, as usually in type-theories, to insure the *unicity of the syntactic operations associated to a type-constructor*) can be viewed as a *reversal* of the usual $(\eta\supset)$-rule.

> Using a $(\beta\supset)$-*reversal* rule in this context leads, however, to an inconsistent calculus/theory, viz., to a formal variant of the **Automath**-principle "classical proofs are irrelevant".

The reduction rules of $\lambda\not\delta!$ generate an appropriate notion of *proof-equality* for **CQ**. We show **Cons**[$\lambda\not\delta!$], i. e., Post-consistency for the equality of $\lambda\not\delta!$, *by a pure type-free* λ*-calculus method* (section **6**). In particular, it should be clear that the proof of **Cons**[$\lambda\not\delta!$] does not depend on considerations about proof-reduction.

The book is organized in *three main parts* and a *coda*. (**a**) The sections **2**–**4** are concerned with *stratification matters*, (**b**) the sections **5**–**8** are devoted to a characterization of the *equational proof-behaviors*, whereas (**c**) sections **9**–**12** characterize the notion of *proof-reduction* [= *proof-détour elimination*] *in first-order classical logic*. (**d**) The final section, **13**, shows that a natural (extensional) variant of Heyting's proof-calculus for the first-order logic **HQ** can be embedded into $\lambda\not\delta!$.

(**a**) The main result on *stratification* is as intended/expected: the underlying type-structures π[**CQ**] (of $\lambda\not\delta!$) and π_c[**CQ**] (of **C**[**CQ**]) yield *equivalent stratification criteria* and both are shown to coincide, in extent, with the concept of *provability in first-order classical logic* (section **4**).

(**b**) The *equivalence* is preserved at *equational* level: indeed, the λ-variant $\lambda\not\delta!$ and the combinatory theory **C**[**CQ**] can be embedded reciprocally, into each other; otherwise, the proof of this result matches a pattern well-known from "type-free" calculi (section **8**). The hard part of the work behind this kind of the equivalence (viz., the embedding of $\lambda\not\delta!$ into the combinatory theory **C**[**CQ**]) is done by appropriate abstraction "algorithms": these are "typed" extensions of the familiar ("extensional") abstraction algorithms of H. B. Curry ([Curry *et al.* 1972], [Hindley & Seldin 1986]).

As expected, the "typed" $\not\delta$-abstractions can be eliminated in favor of usual typed λ-abstractions and *duplex negatio* proof-combinators $\Delta[\![A]\!] : A^{--} \supset A$, by $\not\delta x{:}A^-.a[\![x]\!] := \Delta[\![A]\!](\lambda x{:}A^-.a[\![x]\!])$.

The "oracle functionals" $\Delta[\![A]\!]$ *extend* the notion of *construction* or *effective procedure*, as available in Brouwer's Intuitionism and Bishop's Constructive Mathematics [Troelstra & van Dalen 1988].

Somewhat unexpected is only the fact that the "complex" $\Delta[\![A]\!]$'s (where $A \equiv [B \supset C]$ or $\forall u.B$, say) can be eliminated recursively, *modulo* proof-equality, in favor of "atomic" $\Delta[\![A]\!]$'s (where $A \equiv \top$ or $\bot$ or a "prime" type/proposition $P[\![u_1,\ldots,u_n]\!]$) and a few "strictly linear" proof-functionals (section 7).

> These are *the* **next** *proof-combinators*, stratified variants of the "type-free **next** combinators" used essentially in the proof of **Cons**[$\lambda\eth!$] in section 6; "strictly linear" is taken here in the sense of C. A. Meredith and means "definable in terms of **BCI**-combinators". (As conjectured by C. A. Meredith in the fifties and as shown by J. Roger Hindley [1989], *the* **BCI**-*definable combinators are stratifiable*.)

(c) The latter result can be also reformulated properly in terms of *proof-reduction*. Specifically, we show (section 10) that, in $\lambda\eth!$, the notion of reduction generated by an "extensional lifting" of the "complex" $\beta\eth$-rules is *strongly normalizable* and *confluent*. From an intuitive point of view, this means that we can safely restrict our considerations to a "type-normal" fragment $^{TN}\lambda\eth!$ of $\lambda\eth!$, i. e., to the fragment where *reductio ad absurdum* is applied only to atomic types/propositions ($\top$, $\bot$ or "primes"). Technically, the full $\lambda\eth!$-calculus can be viewed as the result of a *systematic abbreviation procedure* applied to $^{TN}\lambda\eth!$.

As expected, one has first **SN**[$\lambda\eth!$], *strong normalization* for the *standard* notion of proof-reduction of (the full) $\lambda\eth!$-calculus (section 11). In other words, the proof-détour eliminations in $\lambda\eth!$ are characterized by *terminating processes*, no matter which is the reduction-strategy actually employed: *normal forms* [*of proofs*] *do always exist in* $\lambda\eth!$. We show, however, something more, viz. the fact that **SN**[$\lambda\eth!$] <=> **SN**[$\lambda!$] <=> **SN**[λ], where $\lambda!$ is as above and λ is the ordinary typed λ-calculus (with types from a language based on $\top$, $\bot$ and $\supset$). The first half of this (i. e., **SN**[$\lambda\eth!$] <=> **SN**[$\lambda!$]) is obtained by extending a familiar "negative translation" argument (implicit in the $\neg\neg$-interpretations of **CQ** into a proper fragment of Heyting's first-order logic **HQ**) to the *proof languages* under focus. The second half (**SN**[$\lambda!$] <=> **SN**[λ]) follows by a translation-argument, collapsing the $\lambda!$-syntax onto the ordinary λ-syntax.

Moreover, we obtain also **CR**[$^{TN}\lambda\eth!$], i. e., *confluence* for $^{TN}\lambda\eth!$, a Church/Rosser-type theorem (section 12), by extending a standard technique (due to W. W. Tait and P. Martin-Löf, cf. [Rezus 1981], [Barendregt 1984], [Takahashi 1989]) to the case in point. This insures **UN**[$\lambda\eth!$], *unicity of normal forms in* the full calculus $\lambda\eth!$, i. e., the fact that the *terminating proof-reduction processes* referred to above are *definite*, as regards the end-results.

(d) As a by-product, we can show (section 13) that a natural typed λ-variant of the Heyting proof-calculus for **HQ** (without *ad hoc* "permutative" and "⊥-rules"), **λH!** say, can be interpreted directly into **λℐ!**. This intepretation preserves equality and reductions resp., whence we have also **Cons**[**λH!**] and **SN**[**λH!**], trivially. Without the *ad hoc* rules mentioned above, one has, however, ¬**CR**[**λH!**].

Related work. Apparently, the papers of Glen Helman [1983,1987] are the only references available in print, so far, concerned with the explanation of *classical proofs* in a context close to the point of view advocated here. Helman has, however, no proper proof-syntax and intends to apply category-theoretic methods to the study of classical proof-equality. If considered in the prolongation of the familiar **CCC**/topos-theoretic treatement of Heyting's logic, this kind of approach should fail, in view of a well-known fact (A. Joyal) on the unicity of arrows **A** → **0** (up to equivalence) in bi-cartesian closed categories.

In extent, this book covers most of the subject matter of Prawitz' [1965] standard - although technically defective - presentation of **Natural Deduction** from a *rigorous* point of view (typed λ-calculus), advocating programmatically the *priority* of the *classical logic concept of a proof* over the Heyting ("intuitionistic") analogue. The latter is accounted for as a *special* case.

> Further details, related to work of [Prawitz 1965,1971] (e. g., on "sequent" systems, Beth *tableau*-search, λ-model construction, etc.) and the analogous type-theoretic approach to first-order modal logics are discussed elsewhere.

Our exposition is *complementary* to material appearing in Girard's Paris 1986-87 course [Girard *et al.* 1989]; the latter is concerned exclusively with Heyting proof-calculi.

Acknowledgments. The bulk of the work reported in this book has been presented in lectures and conferences. In detail:

- the presentation of classical proof-calculi relies on talks given during 1988-89 on various occasions and in several places (as, e. g., [Rezus 1988,1989,1989a]);
- the work related to Heyting's logic and Martin-Löf's type theory (**CST**) originates in lectures given at the University of Nijmegen (Department of Computer Science) during 1983-1987 and has been mainly supported by the Dutch Scientific Research Foundation [**NWO**] ([Rezus 1983a,1986,1986a,1987,1987a,b,c]);
- the general type-theoretic design is inspired by early work of the author (and other's) on the family of **Automath** languages and **CST** (essentially, [Rezus 1983,1986b], [Barendregt & Rezus 1983] and [Rezus 1987a,b]).

Finally, this book can be also considered as a development of ideas due, essentially, to Carew A. Meredith and David Meredith.

2. First-order classical logic: combinatory proof-syntax. The *proof-syntax* of the first-order classical logic is considered here as an extension of the usual *provability syntax*, the syntax of formulas/propositions, with a new syntactic category: the *proof-terms*.

Formally, the proof-theory of first-order classical logic appears as a *type-theory*.

Type syntax. We identify, *via* the *propositions-as-types isomorphism*, types/type-expressions and propositions/formulas.

2.1 Definition (*First-order type syntax*).

(1) *The universe of discourse* $\mathcal{U}$ is supposed to contain *referential points*, denoted by $\mathcal{U}$-*terms*. The latter are ranged over by s, t, possibly decorated. For *pure logical theories*, the $\mathcal{U}$-terms are:

- $\mathcal{U}$-*variables* (ranging over $\mathcal{U}$): u, v, w, ... and, possibly,
- $\mathcal{U}$-*constants* (denoting distinguished points of $\mathcal{U}$).

(2) *The types/propositions* are ranged over by A, B, C, D, E, ... and make up an inductive class generated from

- *proper atoms*: $P[\![u_1, \ldots, u_n]\!]$ and
- *propositional constants*: $\top$ (*verum*), $\bot$ (*falsum*),

by closing under the primitive type-forming operators; here, minimally, $\supset$ (*material implication*) and a first-order universal quantifier $\forall_{\mathcal{U}}$ ($\mathcal{U}$-*generalization*):

- $(A \supset B)$, $(\forall u.A[\![u]\!])$.

2.2 Notation.

(1) In what follows, $L[\supset,\forall]$ is the provability language of first-order classical *logic*. $L[\supset]$ is the corresponding propositional language. (The propositional constants $\top$, $\bot$ are not displayed, since they are assumed to be *always present*). A provability language for the first-order Heyting logic **HQ** requires also an *external conjunction* (denoted here by $\wedge$ or by &) as well as $\vee$ (*disjunction*) and an existential first-order quantifier $\exists_{\mathcal{U}}$ ($\mathcal{U}$-*specialization*); notation: $L[\supset,\wedge,\vee,\forall,\exists]$ (assuming the primitive atoms $\top$ and $\bot$, too).

(2) *The type/proposition spelling.* We spare on parentheses, by assuming associativity to the left and by applying the usual conventions about separating dots [Church 1956]. So, e. g., $A \supset B \supset C \supset .C \supset A \supset (D \supset A)$ is the same thing as $((A \supset B) \supset C) \supset ((C \supset A) \supset (D \supset A))$.

(3) In a classical provability context, the remaining connectives and the existential quantifier are introduced as abbreviations:

- (A^-) $:= (A \supset \bot)$ [*inferential negation*],
- (A^{--}) $:= (A^-)^-$ [*double negation*],

- t $:= \bot^-$ [*internal verum*],
- f $:= \top^-$ [*internal falsum*),

- $(A \wedge B)$ $:= (A \supset B^-)^-$ [*internal conjunction*],
- $(A \vee B)$ $:= (A^- \wedge B^-)^-$ [*internal disjunction*],
- $(A \supset\subset B)$ $:= (A \supset B) \wedge (B \supset A)$ [*internal equivalence*],
- $(\exists u.A[\![u]\!])$ $:= (\forall u.(A[\![u]\!])^-)^-$ [*existence*].

Note that the "window" brackets $[\![...]\!]$ do *not* belong to the (object) syntax.

2.3 Remark.

(1) The *universe* U and the *U-variables* make up a typical first-order ingredient. We let Var_U be the set of *U-variables*. For any first-order proposition/formula A, $FV_U(A)$ is the set of *free* **U**-*variables* of A, defined in the expected way. In *first-order theories*, one has also specific ("non-logical") means to construct *U-terms*, as, e. g., function variables and/or function constants. The *U-terms* are left unspecified. In what follows, "$\Vdash t :: U$" is shorthand for "t is a *U*-term".

(2) In a $\forall$-free (purely "propositional") provability language, the *proper propositional atoms* $P_i[\![u_1, \ldots, u_n]\!]$ become bare *propositional letters/variables* p_i, $(i \in \mathbb{N})$. In such cases, we use p, q, r, ..., possibly decorated, as syntactic variables ranging over propositions.

Combinatory proof-syntax. It is well-known that the concept of *classical provability* can be characterized recursively by an *axiomatic presentation*.

We introduce such axiomatics in a slightly more rigorous way than in a usual logic text-book: indeed, in the spirit of the Curry-Meredith-Howard *proposition-as-types isomorphism*, an axiomatic presentation corresponds to a *stratified combinatory system*. We show that this is also the case for the (first-order) classical logic **CQ**. Next, the following distinctions are in force.

- *The concept of a classical proof* [*proof in* **CQ**] admits of a *combinatory presentation* π_c[**CQ**] in a *stratified* ["typed"] *combinatory proof-language*. It will be clear that π_c[**CQ**] is a proper [in fact, conservative] extension of the ordinary typed combinatory logic [a formalized/type-disambiguated version of Curry's "basic functionality theory"]. Actually, π_c[**CQ**] yields only a *stratification criterion* for a set of basic objects, the *ground proof-combinators*. Under the intended [*propositions-as-types*] interpretation, a ground proof-combinator is just "the primitive proof" of an axiom in a specific axiomatic presentation of **CQ**.

- Moreover, the *concept of a* **CQ**-*proof*, as isolated in π_c[**CQ**], can be shown to admit of an *equational presentation* **C**[**CQ**]. **C**[**CQ**] formalizes the *proof-equality of* **CQ**. The outcome is a *combinatory logic* [a "combinatory proof-theory"] *for* **CQ**.

The stratification theory π_c[**CQ**]. We describe first the *theory* of the first-order classical proof-combinators, called *Boolean proof-combinators*. A *Boolean proof-combinator* is a *stratified object*. The stratifications are mentioned by *explicit* [*type-*] *parametrizations*. The set of axiom-schemes/rules displayed below yields a combinatory formulation of **CQ**. Technically, we define π_c[**CQ**]. (The associated equational theory is discussed later.)

2.4 Definition (*First-order combinatory proof-terms*).

(1) The *ground Boolean combinators* are obtained from

- $\underline{\Omega}$ [a "constant proof"],
- *primitive Boolean combinators*:
  - *basic Boolean combinators*: $\underline{I}[\![A]\!]$, $\underline{K}[\![A,B]\!]$, $\underline{S}[\![A,B,C]\!]$, $\underline{\Delta}[\![A]\!]$,
  - *Boolean* $\mathcal{U}$*-combinators*: $\underline{K}_{\mathcal{U}}[\![A]\!]$, $\underline{S}_{\mathcal{U}}[\![A,B]\!]$, $\underline{@}_{\mathcal{U}}[\![A,B]\!]$, $\underline{\Phi}_{\mathcal{U}}[\![A]\!]$,

by closing under *ground* $\mathcal{U}$*-generalization* $!_u(\ldots)$, $u \in Var_{\mathcal{U}}$.

(2) The *combinatory proof-terms* are obtained from *ground Boolean combinators* and *proof-variables* (*p-variables*) *for atomic proofs*: x, y, z, ... by closing under two *proof-operators*:

- *message-passing* ("*Meredith detachment*"): @(...,...) and
- $\mathcal{U}$*-parameter-passing* ("*instantiation*"): @(...,...).

(3) *The combinatory proof-terms*, ranged over by X, Y, Z, ..., F, G, ... are defined inductively by:

- The atomic proof-variables and the ground combinators are combinatory proof-terms.
- If X, Y are combinatory proof-terms and **t** is a $\mathcal{U}$-term, then @(X,Y) and @(X,**t**) are combinatory proof-terms.

(4) A *Boolean proof-combinator* is a combinatory proof-term that does not contain p-variables.

The proof-variables make up a set Var_c. Ω is the schematic "proof" of an arbitrary non-logical "axiom" ($\top$).

2.5 Notation.

(1) Throughout in what follows we use the more familiar shorthand:

- $(XY) := @(X,Y)$, $(X[\mathbf{t}]) := @(X,\mathbf{t})$, $(!_u.X) := !_u(X)$.

(2) We make economy on parentheses by association to the left and by omitting oft the outermost pair.

Let $\mathbf{u} \equiv [u_1 \ldots u_n]$ be a sequence of pairwise distinct U-variables, $n \geq 0$ (alternative notation: $\mathbf{u} \equiv [u_1 :: U] \ldots [u_n :: U]$). We write next $\mathbf{u}[v]$ for $[u_1 \ldots u_n, v]$, and assume that v is distinct from the elements of $\mathbf{u}$. For convenience, we write "$v \in \mathbf{u}$" for the fact that that v is among the elements of $\mathbf{u}$. Also, for any type/proposition A, "$FV_U(A) \subseteq \mathbf{u}$" means that the U-parameters of A are in $\mathbf{u}$.

The *provability predicate* $\vdash_U(\ldots)$ of **CQ** is introduced in terms of proof-combinators.

2.6 Definition (π_c[CQ]-stratification).

(1) *Classical axiom(-scheme)s*:

$*(\underline{\Omega})$: $\vdash_U \underline{\Omega} : \mathsf{T}$,
$*(\underline{I})$: $\vdash_U \underline{I}[\![A]\!] : A \supset A$,
$(\underline{K})$: $\vdash_U \underline{K}[\![A,B]\!] : A \supset .B \supset A$,
$(\underline{S})$: $\vdash_U \underline{S}[\![A,B,C]\!] : A \supset (B \supset C) \supset .A \supset B \supset (A \supset C)$,
$(\underline{\Delta})$: $\vdash_U \underline{\Delta}[\![A]\!] : A = \supset A\ [\ \equiv A \supset \bot \supset \bot \supset A\ ]$,
$(\underline{K}_U)$: $\vdash_U \underline{K}_U[\![A]\!] : A \supset \forall u.A$ [u not in $\mathbf{u}, FV_U(A)$],
$(\underline{S}_U)$: $\vdash_U \underline{S}_U[\![A,B]\!] : \forall u.(A \supset B) \supset .\forall u.A \supset \forall u.B$ [u not in $\mathbf{u}$],
$*(\underline{\mathbf{B}}_U)$: $\vdash_U \underline{\mathbf{B}}_U[\![A.B]\!] : A \supset \forall u.B \supset .\forall u.(A \supset B)$ [u not in $\mathbf{u}, FV_U(A)$],
$*(\underline{\Phi}_U)$: $\vdash_U \underline{\Phi}_U[\![A]\!] : \forall u.\forall v.A \supset \forall v.\forall u.A$ [u, v not in $\mathbf{u}$].

(2) *Basic rules* [v not in $\mathbf{u}$]:

$[\supset e]_o$: $\vdash_U F : A \supset B, \vdash_U X : A \Longrightarrow \vdash_U FX : B$,
$[\forall e]_o$: $\vdash_U F : \forall v.\ A[\![v]\!], \Vdash_U t :: U \Longrightarrow \vdash_U F[t] : A[\![v:=t]\!]$,
$[\forall i]_{oo}$: $\vdash_{U[v]} X[\![v]\!] : A[\![v]\!] \Longrightarrow \vdash_U !_v.X[\![v]\!] : \forall v.A[\![v]\!]$,
where $X[\![v]\!]$ is a ground Boolean combinator.

Where A is a type/proposition, X is a combinatory proof-term and $\mathbf{u} \equiv [u_1 \ldots u_n]$, $n \geq 0$, the explicit U-parametrization of the axiom-schemes "$\vdash_U X : A$" above (read: "X proves A") is supposed to be such that $FV_U(A) \subseteq \mathbf{u}$. U-variable clashes are avoided as usually. The starred items in the above turn out to be redundant in π_c[CQ]. The *basic stratification rules* $[\supset e]_o$, $[\forall e]_o$, $[\forall i]_{oo}$ are known as *modus ponens*, *instantiation*, and *ground U-generalization*, resp.

We can now define the concept of **CQ**-*provability "under assumptions"*.

Proof-contexts: stratification relative to a formal context (basic type-assignment). From an intuitive point of view, a *hypothesis* or an *assumption* in a proof is a *virtual proof*. Formally, the *virtual proof-space* is a a class of *finite sets* of *assumptions*. The "points" of this "space" are the so-called "proof-contexts". This terminology comes from **Automath** (cf. [van Daalen 1980], [Rezus 1983]).

A (*proof-*) *context* (an *atomic type-assignment*) can be viewed as a partial map Φ from finite sets of p-variables to sets of types/propositions. This is much similar to a Curry *basis* [Curry *et al.* 1958,1972]. Note, however, that a context, in the present sense, is *U-parametric*.

The combinatory concept of a proof defined by π_C[**CQ**] is extended next to proof-contexts. Technically, this corresponds to the concept of a *formal deduction under hypotheses* [*in* **CQ**] and characterizes syntactically *the classical consequence relation* (cf. for instance, [Montague & Henkin 1956]).

2.7 Remark.

(1) Contexts are displayed in full, if necessary and Φ, possibly decorated, range over contexts. So, a context Φ is denoted by showing its "graph" (as a map). We write, for instance, $\Phi := [\ x_1 : A_1\ ] \ldots [\ x_n : A_n\ ]$. Every element $[\ x_i : A_i\ ]$ of the context Φ is said to be a *basic cell* of the context.

(2) In meta-notation, $\mathcal{U}$-parameters can be shown explicitly, either as in $\Phi[\![u]\!]$ or by making them explicit as *$\mathcal{U}$-cells*, of the form $[u_i :: \mathcal{U}]$. In the latter case, we can display the *$\mathcal{U}$-cells*, as well, intersticing them among the basic cells. [Note that $\mathcal{U}$ is *not* a type/proposition.] So, the *generic notation for $\mathcal{U}$-parametric contexts* could be:

$$[x_i : A_i]_{i<n} := [x_1 : A_1] \ldots [u_j :: \mathcal{U}] \ldots [x_n : A_n].$$

(3) A p-variable x (a $\mathcal{U}$-variable u) is said to be *fresh for* a context Φ if x (resp. u) is not in the domain of Φ.

(4) The notation "$\Phi \Vdash t :: \mathcal{U}$" is shorthand for "**t** is an $\mathcal{U}$-term relative to Φ". (The "relative"-proviso is not important if the $\mathcal{U}$-term structure is not specified. However, in the general case, such a proviso is meant to insure the fact that Φ is of the form $\Phi[\![u]\!]$, if **t** contains a $\mathcal{U}$-variable u.)

2.8 Notation.

- The *empty context* ($\emptyset$) is unique and is shown by [ ].
- The *union* $\Phi_1 \cup \Phi_2$ of two contexts Φ_1, Φ_2 is denoted by Φ_1,Φ_2.
- *The extension of* a context Φ *with a basic cell* [x : A] is a map $\Phi' = [\ x_1 : A_1\ ] \ldots [\ x_n : A_n\ ][\ x : A\ ]$, such that Φ' is also a context.

2.9 Conventions.

(1) [**Convention Φ**]. We use the sequential notation $\Phi[\ x : A\ ]$ for context(-extension)s, assuming, unless otherwise specified explicitly, that [x : A] is *not* an element of Φ.

(2) [**Convention $\Phi_{\mathcal{U}}$**]. If $\Phi \equiv [\ x_1 : A_1\ ] \ldots [\ x_n : A_n\ ]$ and the $\mathcal{U}$-variable u is fresh for Φ [i. e., if u is not free in the A_i's] then it is convenient to indicate this by writing $\Phi[\ u :: \mathcal{U}\ ]$ and to consider that $\Phi[\ u :: \mathcal{U}\ ]$ stands actually for a context $\Phi'[\![u]\!]$.

"*Formal deduction under hypotheses*". In what follows, the predicate $\vdash_{\mathcal{U}}$ is extended up to a relation $\vdash_c$ holding between contexts Φ and "typings" $\varphi \equiv [X : A]$, where X is a combinatory proof-term and A is a type/proposition in $L[\supset, \forall]$. The statements of the form $\Phi \vdash_c \varphi$, with φ as above, are said to be *t-statements* ["typing" statements]. From an intuitive point of view, thereby defined is a concept $\pi_c[\mathbf{CQ}]$ of *formal deduction under hypotheses in* **CQ**.

2.10 Definition (*The combinatory proof-system* $\pi_c[\mathbf{CQ}]$).

(1) ***Context rules***. Let "φ" be "X : C". Then

$\langle\ \rangle$: $\vdash_{\mathcal{U}} \varphi \Longrightarrow [\ ] \vdash_c \varphi$, ($\mathbf{u} \equiv [u_1 :: \mathcal{U} \ldots [u_n :: \mathcal{U}$, $n \geq 0$),
$\langle \mathbf{I} \rangle$: $[x : A] \vdash_c x : A$,
$\langle \mathbf{K} \rangle$: $\Phi \vdash_c \varphi \Longrightarrow \Phi[\ x : A\ ] \vdash_c \varphi$,
$\langle \mathbf{K}\forall \rangle$: $\Phi \vdash_c \varphi \Longrightarrow \Phi[\ u :: \mathcal{U}\ ] \vdash_c \varphi$.

(2) ***Type-derivation***.

$[\supset e]$: $\Phi \vdash_c F : A \supset B$, $\Phi \vdash_c X : A \Longrightarrow \Phi \vdash_c FX : B$,
$[\forall e]$: $\Phi \vdash_c F : \forall u.A[\![u]\!] \Longrightarrow \Phi \vdash_c F[t] : A[\![u:=t]\!]$, if $\Phi \Vdash t :: \mathcal{U}$.

So $\pi_c[\mathbf{CQ}]$ defines a set of *t-statements* $\Phi \vdash_c X : A$, *its admissible "typings"*, extending the combinatory proof-term stratification from proof-combinators to p-variables as well, i. e., ultimately, to the full set of combinatory proof-terms. Note also that the context-rules and conventions ($\Phi_{[\mathcal{U}]}$) insure the fact that, e. g., in rule $[\forall e]$, the $\mathcal{U}$-variable u is not free in **t** and (the types of) Φ. By rule $\langle\ \rangle$, $\vdash_c$ satisfies the axioms ($\underline{\mathrm{I}}$), ($\underline{\mathrm{I}}$), ($\underline{\mathrm{K}}$), ($\underline{\mathrm{S}}$), ($\underline{\Delta}$), ($\underline{\mathrm{K}}_{\mathcal{U}}$), ($\underline{\mathrm{K}}_{\mathcal{U}}$), ($\underline{\mathrm{S}}_{\mathcal{U}}$), ($\underline{\mathrm{B}}_{\mathcal{U}}$), ($\underline{\Phi}_{\mathcal{U}}$) of $\pi_c[\mathbf{CQ}]$, for any context. If no confusion can arise, the subscript "c" on turnstiles "$\vdash_c$" is further omitted. Also, the $\mathcal{U}$-parametrizations are ignored notationally for empty contexts. So where $\mathbf{u} \equiv [u_1, \ldots, u_n]$ contains the $\mathcal{U}$-variables that are free in A, $[\ ] \vdash X : A[\![\mathbf{u}]\!]$ means, in fact, $[\ ][u_1 :: \mathcal{U} \ldots [u_n :: \mathcal{U} \vdash_c X : A[\![\mathbf{u}]\!]$ and is equivalent to $\vdash_{\mathcal{U}} X : A[\![\mathbf{u}]\!]$.

Abstraction "algorithms" for $\pi_c[\mathbf{CQ}]$. The proofs of the so-called "deduction theorem" in "Hilbert-style" axiomatic systems for **CQ** are meant to establish the existence of appropriate "abstraction operators" over assumptions in a formal deduction. In the present setting, this is a small programming exercise (relying, in essence, on [Schönfinkel 1924]; cf. [Curry *et al.* 1958, 1972]).

2.11 Definition (*Abstraction "algorithms" for* $\pi_c[\mathbf{CQ}]$).

(1) "*Functional*" *abstraction over assumptions*. Where a is a type/proposition in $L[\supset, \forall]$, let $X[\![x]\!]$ be a combinatory proof-term such that $\Phi[\ x : A\ ] \vdash X[\![x]\!] : B$. Then $\Phi \vdash X' := \lambda_* x{:}A.X[\![x]\!]$ is defined by induction on the structure of $X[\![x]\!]$, as follows:

($\underline{\mathrm{I}}$) If $X[\![x]\!] \equiv x$ then $\lambda_* x{:}A.X[\![x]\!] \equiv \underline{\mathrm{I}}[\![A]\!]$
for $\Phi[\ x : A\ ] \vdash X[\![x]\!] \equiv x : B \equiv A$.
($\underline{\mathrm{K}}$) If $X[\![x]\!] \equiv Y$ [x is not in Y] then $\lambda_* x{:}A.X[\![x]\!] \equiv \underline{\mathrm{K}}[\![B, A]\!](Y)$
for $\Phi[\ x : A\ ] \vdash X[\![x]\!] \equiv Y : B$.

(f) If $X[\![x]\!] \equiv Fx$ [x is not in F] then $\lambda_* x{:}A.X[\![x]\!] \equiv F$
for $\Phi[\ x : A\ ] \vdash F : A \supset B$
and $\Phi[\ x : A\ ] \vdash x : A$
$\Phi[\ x : A\ ] \vdash X[\![x]\!] \equiv Fx : B$.

($\underline{S}$) If $X[\![x]\!] \equiv (F[\![x]\!])(G[\![x]\!])$ [and clause (f) does not apply]
then $\lambda_* x{:}A.X[\![x]\!] \equiv \underline{S}[\![A,C,B]\!](\lambda_* x{:}A.F[\![x]\!])(\lambda_* x{:}A.G[\![x]\!])$,
for $\Phi[\ x : A\ ] \vdash F[\![x]\!] : C \supset B$
and $\Phi[\ x : A\ ] \vdash G[\![x]\!] : C$
$\Phi[\ x : A\ ] \vdash X[\![x]\!] \equiv (F[\![x]\!])(G[\![x]\!]) : B$.

($\underline{\mathbf{@}}_\cup$) If $X[\![x]\!] \equiv (F[\![x]\!])[t]$
then $\lambda_* x{:}A.X[\![x]\!] \equiv \underline{\mathbf{@}}_\cup[\![A,B]\!](\lambda_* x{:}A.F[\![x]\!])[t]$,
for $\Phi[\ x : A\ ] \vdash F[\![x]\!] : \forall u.B[\![u]\!]$
$\Phi[\ x : A\ ] \Vdash t :: \mathcal{U}$
$\Phi[\ x : A\ ] \vdash X[\![x]\!] \equiv (F[\![x]\!])[t] : B \equiv B[\![u:=t]\!]$.

(2) *Generalization as "first-order" abstraction.* Let $X[\![u]\!]$ be a proof-term such that $\Phi[\ u :: \mathcal{U}\ ] \vdash X[\![u]\!] : A[\![u]\!]$, where A is a type/proposition in $L[\supset,\forall]$. Then $\Phi \vdash X' := !_* u.X[\![u]\!]$ is defined by induction on the structure of $X[\![u]\!]$, as follows:

($!_\cup$) If $X[\![u]\!]$ is a ground combinator then $!_* u.X[\![u]\!] \equiv !_\cup.X[\![u]\!]$.
($\underline{K}_\cup$) If $X[\![u]\!] \equiv Y$ [u is not in Y and is not free in A]
then $!_* u.X[\![u]\!] \equiv \underline{K}_\cup[\![A]\!](Y)$
for $\Phi[\ u :: \mathcal{U}\ ] \vdash X[\![u]\!] \equiv Y : A$ [u is not free in A].
($f_\cup$) If $X[\![u]\!] \equiv F[u]$ [u is not in F] then $!_* u.X[\![x]\!] \equiv F$
for $\Phi[\ u :: \mathcal{U}\ ] \vdash F : \forall u.A[\![u]\!]$
$\Phi[\ u :: \mathcal{U}\ ] \Vdash u :: \mathcal{U}$
$\Phi[\ u :: \mathcal{U}\ ] \vdash X[\![u]\!] \equiv F[u] : A[\![u]\!]$.
($\underline{S}_\cup$) If $X[\![u]\!] \equiv (F[\![u]\!])(G[\![u]\!])$ [and clause ($f_\cup$) does not apply]
then $!_* u.X[\![x]\!] \equiv \underline{S}_\cup[\![B,C]\!](!_* u.F)(!_* u.G)$
for $\Phi[\ u :: \mathcal{U}\ ] \vdash F[\![u]\!] : B[\![u]\!] \supset C[\![u]\!]$
$\Phi[\ u :: \mathcal{U}\ ] \vdash G[\![u]\!] : B[\![u]\!]$
$\Phi[\ u :: \mathcal{U}\ ] \vdash X[\![u]\!] \equiv (F[\![u]\!])(G[\![u]\!]) : C[\![u]\!] \equiv A$.
($\underline{\mathbf{@}}_\cup$) If $X[\![u]\!] \equiv (F[\![u]\!])[s]$ [u is not in s],
then $!_* u.X[\![x]\!] \equiv \underline{\mathbf{@}}_\cup[\![B]\!](!_* u.F[\![u]\!])[s]$
for $\Phi[\ u :: \mathcal{U}\ ] \vdash F : \forall v.B[\![v]\!]$
$\Phi[\ u :: \mathcal{U}\ ] \Vdash s :: \mathcal{U}$ [u is not in s]
$\Phi[\ u :: \mathcal{U}\ ] \vdash X[\![u]\!] \equiv (F[\![u]\!])[s] : B[\![v:=s]\!] \equiv A$.

(3) *Negative-abstraction:* ***reductio*** *proof-operators.* For all A, Φ and $X[\![x]\!]$ such that $\Phi[x : A^-] \vdash X[\![x]\!] : \bot$, set, in context Φ, $\not\partial_* x{:}A^-.X[\![x]\!] := \underline{\Delta}[\![A]\!](\lambda_* x{:}A^-.X[\![x]\!])$.

2.12 Remark.

(1) One can extract different "abstraction algorithms" from 2.11 (1), (2) by imposing, in each case, a *deterministic order* in the application of the clauses. The most efficient "algorithms", as regards the compactness of the generated combinatory code apply the "extensional" clauses (f), resp. ($f_\cup$) first, whenever this is possible. Conventionally, we choose this *specific sequentialization* of 2.11 (1), 2.11 (2) resp. as a *standard definition* for λ_* ($\not\partial_*$) and $!_*$, resp.

(2) Assume that $\pi_c[CQ]$ (resp. $\pi_*[CQ]$) has been extended with the additional (primitive) ground combinators $\underline{B}[A,B,C]$, $\underline{C}[A,B,C]$, $\underline{B}_\cup[A,B]$ and $\underline{C}_\cup[A,B]$, stratified by

$(\underline{B})$: $[\,] \vdash \underline{B}[A,B,C] : A \supset B \supset .C \supset A \supset (C \supset B)$,
$(\underline{C})$: $[\,] \vdash \underline{C}[A,B,C] : A \supset (B \supset C) \supset .B \supset (A \supset C)$,
$(\underline{B}_\cup)$: $[\,] \vdash \underline{B}_\cup[A,B] : A \supset B \supset .\forall u.A \supset \forall u.B$ [u not in $FV_\cup(A,B)$],
$(\underline{C}_\cup)$: $[\,] \vdash \underline{C}_\cup[A,B] : \forall u.(A \supset B[u]) \supset .A \supset \forall u.B[u]$ [u not in $FV_\cup(A)$].

[Such combinators are, in fact, definable in $\pi_c[CQ]$, $\pi_*[CQ]$. See below.] Then, applying a well-known technique, due to Haskell B. Curry, one can generate significanly more compact combinatory code, by analyzing $(\underline{S})$ and $(\underline{S}_\cup)$ above into:

$(\underline{B})$ If $X[x] \equiv F(G[x])$ [x is in G, but not in F]
then $\lambda_* x{:}A.X[x] \equiv \underline{B}[C,B,A](F)(\lambda_* x{:}A.G[x])$,
for $\Phi[\ x : A\ ] \vdash F : C \supset B$
and $\Phi[\ x : A\ ] \vdash G[x] : C$
$\Phi[\ x : A\ ] \vdash X[x] \equiv F(G[x]) : B$.

$(\underline{C})$ If $X[x] \equiv (F[x])G$ [x is in F, but not in G]
then $\lambda_* x{:}A.X[x] \equiv \underline{C}[A,C,B](\lambda_* x{:}A.F[x])(G)$
for $\Phi[\ x : A\ ] \vdash F[x] : C \supset B$
and $\Phi[\ x : A\ ] \vdash G : C$
$\Phi[\ x : A\ ] \vdash X[x] \equiv (F[x])G : B$.

$(\underline{S})$ If $X[x] \equiv (F[x])(G[x])$, [x is in both F and G]
then $\lambda_* x{:}A.X[x] \equiv \underline{S}[A,C,B](\lambda_* x{:}A.F[x])(\lambda_* x{:}A.G[x])$,
for $\Phi[\ x : A\ ] \vdash F[x] : C \supset B$
and $\Phi[\ x : A\ ] \vdash G[x] : C$
$\Phi[\ x : A\ ] \vdash X[x] \equiv (F[x])(G[x]) : B$.

$(\underline{B}_\cup)$ If $X[u] \equiv F(G[u])$ [u is not in F and is not free in A, B],
then $!_* u.X[x] \equiv \underline{B}_\cup[B,C](F)(!_* u.G[u])$
for $\Phi[\ u :: \mathcal{U}\ ] \vdash F : B \supset C$
$\Phi[\ u :: \mathcal{U}\ ] \vdash G[u] : B$ [u is not free in B, C]
$\Phi[\ u :: \mathcal{U}\ ] \vdash X[u] \equiv F(G[u]) : C \equiv A$.

$(\underline{C}_\cup)$ If $X[u] \equiv (F[u])(G)$ [u is not in G and is not free in B],
then $!_* u.X[x] \equiv \underline{C}_\cup[B,C](!_* u.F)(G)$
for $\Phi[\ u :: \mathcal{U}\ ] \vdash F[u] : B \supset C[u]$
$\Phi[\ u :: \mathcal{U}\ ] \vdash G : B$ [u is not free in B]
$\Phi[\ u :: \mathcal{U}\ ] \vdash X[u] \equiv (F[u])(G) : C[u] \equiv A$.

$(\underline{S}_\cup)$ If $X[u] \equiv (F[u])(G[u])$, [u is in both F and G]
then $!_* u.X[x] \equiv \underline{S}_\cup[B,C](!_* u.F)(!_* u.G)$
for $\Phi[\ u :: \mathcal{U}\ ] \vdash F[u] : B[u] \supset C[u]$
$\Phi[\ u :: \mathcal{U}\ ] \vdash G[u] : B[u]$
$\Phi[\ u :: \mathcal{U}\ ] \vdash X[u] \equiv (F[u])(G[u]) : C[u] \equiv A$.

The most compact version of the latter "algorithms" must, of course, apply clauses (f) resp. $(f_\cup)$ first, whenever possible.

2.13 Remark (*The ground proof-combinators*).

(1) For all types/propositions A, B, C in $\mathcal{L}[\supset,\forall]$, the (positive) ground proof-combinators are such that:

$[\ ] \vdash \underline{I}[\![A]\!] \equiv \lambda_* x{:}A.x,$
$[\ ] \vdash \underline{K}[\![A,B]\!] \equiv \lambda_* x{:}A.\lambda_* y{:}B.x,$
$[\ ] \vdash \underline{S}[\![A,B,C]\!] \equiv \lambda_* x{:}(A \supset (B \supset C)).\lambda_* y{:}(A \supset B).\lambda_* z{:}A.xz(yz),$

$[\ ] \vdash \underline{K}_\cup[\![A]\!] \equiv \lambda_* x{:}A.!_* u.x,$
[provided u is not in $FV_\cup(A)$],
$[\ ] \vdash \underline{S}_\cup[\![A,B]\!] \equiv \lambda_* x{:}(\forall u.(A \supset B)).\lambda_* y{:}\forall u.A.!_* u.x[u](y[u]),$
$[\ ] \vdash \underline{\mathfrak{B}}_\cup[\![A,B]\!] \equiv \lambda_* x{:}(A \supset \forall u.B).!_* u.\lambda_* y{:}A.(xy)[u],$
[provided u is not in $FV_\cup(A)$],
$[\ ] \vdash \underline{\Phi}_\cup[\![A]\!] \equiv \lambda_* x{:}(\forall u.\forall v.A).!_* v.!_* u.x[u][v],$

(2) If $\underline{B}[\![A,B,C]\!]$, $\underline{C}[\![A,B,C]\!]$, $\underline{B}_\cup[\![A,B]\!]$ and $\underline{C}_\cup[\![A,B]\!]$ are among the ground combinators [cf. 2.12 (2)], one has also, for all types/propositions A, B, C in $\mathcal{L}[\supset,\forall]$, with the extended definitions of λ_* and $!_*$,

$[\ ] \vdash \underline{B}[\![A,B,C]\!] \equiv \lambda_* x{:}(A \supset B).\lambda_* y{:}(C \supset A).\lambda_* z{:}C.x(yx),$
$[\ ] \vdash \underline{C}[\![A,B,C]\!] \equiv \lambda_* x{:}(A \supset (B \supset C)).\lambda_* y{:}B.\lambda_* z{:}A.xzy,$

$[\ ] \vdash \underline{B}_\cup[\![A,B]\!] \equiv \lambda_* x{:}(A \supset B).\lambda_* y{:}(\forall u.A).!_* u.x(y[u]),$
[provided u is not in $FV_\cup(A,B)$],
$[\ ] \vdash \underline{C}_\cup[\![A,B]\!] \equiv \lambda_* x{:}(\forall u.(A \supset B)).\lambda_* y{:}A.!_* u.x[u]y,$
[provided u is not in $FV_\cup(A)$].

(3) For all types/propositions A in $\mathcal{L}[\supset,\forall]$ and all contexts Φ,

$[\ ] \vdash \underline{\Delta}[\![A]\!] \equiv \lambda_* x{:}A^-.\mathfrak{d}_* y{:}A^-.xy\ [\ :\ A^- \supset A\ ].$

Proof. (1)-(2). We exemplify only the λ_*-expansion of $\underline{S}[\![A,B,C]\!]$. The other cases are similar [for $\cup$-combinators, use clause $(f_\cup)$].

$\lambda_* x{:}(A \supset .B \supset C).\lambda_* y{:}(A \supset B).\lambda_* z{:}A.xz(yz) \equiv$
$\equiv \lambda_* x{:}(A \supset .B \supset C).\lambda_* y{:}(A \supset B).(\underline{S}[\![A,B,C]\!](\lambda_* z{:}A.xz)(\lambda_* z{:}A.yz)) \equiv$
$\equiv \lambda_* x{:}(A \supset .B \supset C).\lambda_* y{:}(A \supset B).\underline{S}[\![A,B,C]\!](x)(y) \equiv$ [clause (f)]
$\equiv \lambda_* x{:}(A \supset .B \supset C).\underline{S}[\![A,B,C]\!](x) \equiv \underline{S}[\![A,B,C]\!]$ [clause (f)].

(3) By the definition of λ_* and $\mathfrak{d}_*$ one has

$[\ ] \vdash \lambda_* x{:}A^-.\mathfrak{d}_* y{:}A^-.xy \equiv \lambda_* x{:}A^-.(\underline{\Delta}[\![A]\!](\lambda_* y{:}A^-.xy)) \equiv$ [clause (f)]
$\equiv \lambda_* x{:}A^-.\underline{\Delta}[\![A]\!](x) \equiv$ [clause (f)]
$\equiv \underline{\Delta}[\![A]\!]\ [\ :\ A^- \supset A].$ □

2.14 Theorem (*The classical type-assignment: combinatory version*).

For all types/propositions A, B in $\mathcal{L}[\supset,\forall]$ and all contexts Φ,

$$(\supset i\lambda_*):\ \frac{\Phi[\ x\ :\ A\ ] \vdash Y[\![x]\!] : B}{\Phi \vdash \lambda_* x{:}A.Y[\![x]\!] : A \supset B},$$

$$(\supset i\mathfrak{d}_*):\ \frac{\Phi[x\ :\ A^-] \vdash X[\![x]\!] : \bot}{\Phi \vdash \mathfrak{d}_* x{:}A^-.X[\![x]\!] : A},$$

$$(\forall i!_*)\colon \quad \frac{\Phi[\ u\ ::\ \mathcal{U}\ ] \vdash X[\![u]\!] : A[\![u]\!]}{\Phi \vdash !_*u.X[\![x]\!] : \forall u.A[\![u]\!].}$$

Proof. $(\supset i\lambda_*)$: By induction on the structure of $Y[\![x]\!]$. $(\supset i\gamma_*)$: By $(\supset i\lambda_*)$, we have $\Phi[x : A^-] \vdash X[\![x]\!] : \bot \Longrightarrow \Phi \vdash \lambda_* x{:}A^-.X[\![x]\!] : A^-$, while $[\ ] \vdash \Delta[\![A]\!] : A^- \supset A$. So, $\Phi \vdash \underline{\Delta}[\![A]\!] : A^- \supset A$, for any context Φ such that $\Phi \vdash \varphi$, for some φ. From this, one obtains, by $(\supset e)$, $\Phi \vdash \gamma_* x{:}A^-.X[\![x]\!] \equiv \underline{\Delta}[\![A]\!](\lambda_* x{:}A^-.X[\![x]\!]) : A$. $(\forall i!_*)$: By induction on the structure of $X[\![u]\!]$. □

2.15 Theorem (*Positive extensionality*).

$(\eta\supset\lambda_*)_=$: If x is not in F, $\Phi \vdash F : A \supset B \Longrightarrow \Phi \vdash \lambda_* x{:}A.Fx \equiv F$.

$(\eta\forall!_*)_=$: If u is not in F, $\Phi \vdash F : \forall u.A \Longrightarrow \Phi \vdash !_* u.F[u] \equiv F$.

Proof. $(\eta\supset\lambda_*)_=$: From the definition of $\lambda_* x{:}A.X[\![x]\!]$, by clause (f). $(\eta\forall!_*)_=$: From the definition of $!_* u.X[\![u]\!]$, by clause $(f_\cup)$. □

2.16 Remark.

(1) If $\Phi \vdash F : A$, set $\underline{\forall}_=[\![A]\!](F) := \underline{S}[\![A^-, A, \bot]\!](\underline{I}[\![A^-]\!])(\underline{K}[\![A, A^-]\!](F))$, in context Φ. Then, for all types/propositions A in $\mathcal{L}[\supset,\forall]$, all contexts Φ and all combinatory proof-terms F such that x is not in F,

$(\eta\supset\gamma_*)_=$: $\quad \Phi \vdash F : A \Longrightarrow \Phi \vdash \gamma_* x{:}A^-.xF \equiv \underline{\Delta}[\![A]\!](\underline{\forall}_=[\![A]\!](F)) : A$.

(2) If the $\mathbf{C}[\![A,B,C]\!]$-combinators are among the primitives, set, alternatively, $\underline{\forall}_=[\![A]\!] := \underline{C}[\![A^-, A, \bot]\!](\underline{I}[\![A^-]\!])$. Then $(\eta\supset\gamma_*)_=$ holds, in the same conditions, with the new definition of $\underline{\forall}_=[\![A]\!]$.

Proof. (1) Assume that x is not in F and $\Phi \vdash F : A$. Then

$$\begin{aligned}\Phi \vdash \gamma_* x{:}A^-.xF &\equiv_{df} \underline{\Delta}[\![A]\!](\lambda_* x{:}A^-.xF) \equiv \\ &\equiv \underline{\Delta}[\![A]\!](\underline{S}[\![A^-, A, \bot]\!](\lambda_* x{:}A^-.x)(\lambda_* x{:}A^-.F)) \equiv \\ &\equiv \underline{\Delta}[\![A]\!](\underline{S}[\![A^-, A, \bot]\!](\underline{I}[\![A^-]\!])(\lambda_* x{:}A^-.F)) \equiv \\ &\equiv \underline{\Delta}[\![A]\!](\underline{S}[\![A^-, A, \bot]\!](\underline{I}[\![A^-]\!])(\underline{K}[\![A, A^-]\!](F))) : A.\end{aligned}$$

(2) In the same conditions, with ground combinators $\underline{C}[\![A,B,C]\!]$,

$$\begin{aligned}\Phi \vdash \gamma_* x{:}A^-.xF &\equiv_{df} \underline{\Delta}[\![A]\!](\lambda_* x{:}A^-.xF) \equiv \\ &\equiv \underline{\Delta}[\![A]\!](\underline{C}[\![A^-, A, \bot]\!](\lambda_* x{:}A^-.x)(F)) \equiv \\ &\equiv \underline{\Delta}[\![A]\!](\underline{C}[\![A^-, A, \bot]\!](\underline{I}[\![A^-]\!])(F)) : A.\ \square\end{aligned}$$

Standard proof-combinators: examples. In the above, one could have defined, e. g., $\underline{I}_*[\![A]\!] := \underline{S}[\![A, B\supset A, A]\!](\underline{K}[\![A, B\supset A]\!])(\underline{K}[\![A, B]\!])$, along the usual "type-free" pattern, in which case $[\ ] \vdash \underline{I}_*[\![A]\!] : A \supset A$, too. In the end, $\underline{\mathbf{B}}[\![A,B]\!]$ and $\underline{\Phi}[\![A]\!]$ are also eliminable in the context of $\pi_c[\mathbf{CQ}]$ [*exercise*]. (This leads to technical complications in the formulation of $\pi_{\mathbf{c}}[\mathbf{CQ}]$.)

On the other hand, it is most convenient to define other Boolean combinators, in a uniform way, by standard λ_*- and $!_*$-expansions. In what follows, we exemplify a few relevant cases.

Note that most $[\top,\bot,\supset,\forall]$-*proof-combinators* appearing below should be already familiar from "type-free" λ-calculus or, alternatively, from the proof-theory of Heyting's logic.

The $[\wedge\text{-}\vee\text{-}\exists]$-*proof-combinators* displayed next are, however, *genuinely classical* ("Boolean").

2.17 Definition.

For all types/propositions A, B, C, in $\mathcal{L}[\supset,\forall]$,

(1) $[\top,\bot,\supset$-*proof-combinators*$]$.

$$\begin{array}{ll}
\underline{K}_*[\![A]\!] & := \lambda_*x{:}\top.\lambda_*y{:}A.\Omega,\\
\underline{K}'[\![A,B]\!] & := \lambda_*x{:}A.\lambda_*y{:}B.y,\\
\underline{B}[\![A,B,C]\!] & := \lambda_*x{:}(A\supset B).\lambda_*y{:}(C\supset A).\lambda_*z{:}C.x(yz),\\
\underline{B}'[\![A,B,C]\!] & := \lambda_*x{:}(A\supset B).\lambda_*y{:}(B\supset C).\lambda_*z{:}A.y(xz),\\
\underline{C}[\![A,B,C]\!] & := \lambda_*x{:}(A\supset.B\supset C).\lambda_*y{:}B.\lambda_*z{:}A.xzy,\\
\underline{C}_*[\![A,B]\!] & := \lambda_*x{:}A.\lambda_*y{:}A\supset B.yx,\\
\underline{\forall}[\![A]\!] & := \underline{C}_*[\![A,\bot]\!]\ [\ \equiv \lambda_*x{:}A.\lambda_*y{:}A^-.yx\ ],\\
\underline{W}[\![A,B]\!] & := \lambda_*x{:}(A\supset.A\supset B).\lambda_*y{:}A\supset B.xyy,\\
 & \\
\underline{\omega}[\![A]\!] & := \lambda_*x{:}\bot.\,\eth_*y{:}A^-.x.
\end{array}$$

(2) $[\wedge$-*proof-combinators*$]$:

$$\begin{array}{ll}
\underline{P}[\![A,B]\!] & := \lambda_*x{:}A.\lambda_*y{:}B.\lambda_*z{:}(A\supset B^-).zxy,\\
\underline{U}[\![A,B,C]\!] & := \lambda_*x{:}(A\supset.B\supset C).\lambda_*z{:}(A\wedge B).x(\mathbf{1}_{A,B}[\![z]\!])(\mathbf{2}_{A,B}[\![z]\!]),\\
\underline{P}_1[\![A,B]\!] & := \lambda_*z{:}(A\wedge B).\mathbf{1}_{A,B}[\![z]\!],\\
\underline{P}_2[\![A,B]\!] & := \lambda_*z{:}(A\wedge B).\mathbf{2}_{A,B}[\![z]\!],\\
\multicolumn{2}{l}{\text{where } \mathbf{1}_{A,B}[\![z]\!] \equiv \eth_*z'{:}A^-.z(\lambda_*x'{:}A.\lambda_*y'{:}B.z'x')}\\
\multicolumn{2}{l}{\text{and}\quad \mathbf{2}_{A,B}[\![z]\!] \equiv \eth_*z'{:}B^-.z(\lambda_*x'{:}A.z').}
\end{array}$$

(3) $[\vee$-*proof-combinators*$]$:

$$\begin{array}{ll}
\underline{J}_1[\![A,B]\!] & := \lambda_*x{:}A.\lambda_*z{:}(A^-\wedge B^-).\underline{P}_1[\![A^-,B^-]\!](z)(x),\\
\underline{J}_2[\![A,B]\!] & := \lambda_*y{:}B.\lambda_*z{:}(A^-\wedge B^-).\underline{P}_2[\![A^-,B^-]\!](z)(y),\\
\underline{D}[\![A,B,C]\!] & := \lambda_*x{:}(A\supset B).\lambda_*y{:}(A\supset C).\lambda_*h{:}(A\vee B).\,\eth_*z{:}C^-.h(F[\![x,y,z]\!]),\\
\multicolumn{2}{l}{\text{where } F[\![x,y,z]\!] \equiv \underline{P}[\![A^-,B^-]\!](\lambda_*x'{:}A.z(xx'))(\lambda_*y'{:}B.z(yy')).}
\end{array}$$

(4) $[\forall$-*proof-combinators*$]$:

$$\begin{array}{ll}
\underline{B}_u[\![A,B]\!] & := \lambda_*x{:}(A\supset B).\lambda_*y{:}(\forall u.A).!_*u.x(y[u]),\\
 & \quad [\text{provided u is not in } FV_u(A,B)],\\
\underline{C}_u[\![A,B]\!] & := \lambda_*x{:}(\forall u.(A\supset B)).\lambda_*y{:}A.!_*u.x[u]y,\\
 & \quad [\text{provided u is not in } FV_u(A)].
\end{array}$$

(5) [∃-*proof-combinators*]:

$$\underline{\pi}[\![A]\!] := !_*u.\lambda_*x{:}A[\![u]\!].\lambda_*y{:}(\forall v.(A[\![u:=v]\!])^-).y[u]x,$$
[provided v is not in $FV_\cup(A)$],
$$\underline{\tau}[\![A,B]\!] := \lambda_*x{:}(\forall u.(A[\![u]\!]\supset B)).\lambda_*y{:}(\exists u.A[\![u]\!]).\,\partial_*z{:}B^-.y(F[\![x,z]\!]),$$
[provided u is not in $FV_\cup(B)$],
where $F[\![x,z]\!] \equiv !_*v.\lambda_*y'{:}(A[\![u:=v]\!]).z(x[v]y')$.

Then, one can establish easily the following

2.18 Fact.

For all types/propositions A, B, C, in $\mathcal{L}[\supset,\forall]$,

$(\underline{K}_*)$: $[\ ] \vdash \underline{K}_*[\![A]\!] : \top \supset .A \supset \top$,
$(\underline{K}')$: $[\ ] \vdash \underline{K}'[\![A,B]\!] : A \supset .B \supset B$,
$(\underline{B})$: $[\ ] \vdash \underline{B}[\![A,B,C]\!] : A \supset B \supset .C \supset A \supset (C \supset B)$,
$(\underline{B}')$: $[\ ] \vdash \underline{B}'[\![A,B,C]\!] : A \supset B \supset .B \supset C \supset (A \supset C)$,
$(\underline{C}_*)$: $[\ ] \vdash \underline{C}_*[\![A,B]\!] : A \supset .A \supset B \supset B$,
$(\underline{\nabla})$: $[\ ] \vdash \underline{\nabla}[\![A]\!] : A \supset A^-$,
$(\underline{W})$: $[\ ] \vdash \underline{W}[\![A,B]\!] : A \supset (A \supset B) \supset .A \supset B$,

$(\underline{\omega})$: $[\ ] \vdash \underline{\omega}[\![A]\!] : \bot \supset A$,

$(\underline{P})$: $[\ ] \vdash \underline{P}[\![A,B]\!] : A \supset .B \supset (A \wedge B)$,
$(\underline{U})$: $[\ ] \vdash \underline{U}[\![A,B,C]\!] : A \supset .B \supset C \supset (A \wedge B \supset C)$,
$(\underline{P}_1)$: $[\ ] \vdash \underline{P}_1[\![A,B]\!] : (A \wedge B) \supset A$,
$(\underline{P}_2)$: $[\ ] \vdash \underline{P}_2[\![A,B]\!] : (A \wedge B) \supset B$,

$(\underline{J}_1)$: $[\ ] \vdash \underline{J}_1[\![A,B]\!] : A \supset (A \vee B)$,
$(\underline{J}_2)$: $[\ ] \vdash \underline{J}_2[\![A,B]\!] : B \supset (A \vee B)$,
$(\underline{D})$: $[\ ] \vdash \underline{D}[\![A,B,C]\!] : A \supset C \supset .B \supset C \supset (A \vee B \supset C)$.

$(\underline{B}_\cup)$: $[\ ] \vdash \underline{B}_\cup[\![A,B]\!] : A \supset B \supset .\forall u.A \supset \forall u.B$,
[provided u is not in $FV_\cup(A,B)$],
$(\underline{C}_\cup)$: $[\ ] \vdash \underline{C}_\cup[\![A,B]\!] : \forall u.(A \supset B[\![u]\!]) \supset .A \supset \forall u.B[\![u]\!]$,
[provided u is not in $FV_\cup(A)$],

$(\underline{\pi})$: $[\ ] \vdash \underline{\pi}[\![A]\!] : \forall u.(A[\![u]\!] \supset \exists v.A[\![u:=v]\!])$,
[provided v is not in $FV_\cup(A[\![u]\!])$],
$(\underline{\tau})$: $[\ ] \vdash \underline{\tau}[\![A,B]\!] : \forall u.(A[\![u]\!] \supset B) \supset .(\exists u.A[\![u]\!]) \supset B$,
[provided u is not in $FV_\cup(B)$].

Proof. Straightforward [*exercise*]. □

2.19 Remark.

(1) Although the "pairing" simulation $\underline{P}[\![A,B]\!]$ uses a well-known pattern (from A. Church), the associated projectors $\underline{P}_i[\![A,B]\!]$ [i := 1,2] are genuinely "Boolean". $\underline{U}[\![A,B,C]\!]$ is a *uncurry*-ing proof-combinator (it stands for a proof of the "importation law").

(2) Notably, the primitive/ground [⊃,∀]-group (without $\underline{\Delta}[\![A]\!]$, but with say $\underline{\omega}[\![A]\!]$), together with the [∧-∨-∃]-group are also known to yield a complete axiomatization for the Heyting [1930] logic [*exercise*]. (The [⊥-∧-∨-∃]-intuitionistic principles are, of course, *simulated* in terms of *reductio ad absurdum*.)

2.20 Definition

(1) For all types/propositions A, B, C in $\mathcal{L}[\supset,\forall]$, any context Φ and all combinatory proof-terms X, Y such that $\Phi \vdash X : A \supset B$ and $\Phi \vdash Y : C \supset A$, set, in context Φ,

$$X \circ Y := \underline{S}[\![C,A,B]\!](\underline{K}[\![A\supset B,C]\!](X))(Y).$$

(2) For all types/propositions A, B in $\mathcal{L}[\supset,\forall]$ such that u is not in $FV_u(A,B)$, any context Φ and all combinatory proof-terms X, Y such that $\Phi \vdash X : A \supset B$ and $\Phi \vdash Y : \forall u.B$, set, in context Φ,

$$X \circ_u Y := \underline{S}_u[\![A,B]\!](\underline{K}_u[\![A\supset B]\!](X))(Y).$$

2.21 Remark.

(1) For all types/propositions A, B, C in $\mathcal{L}[\supset,\forall]$ and any context Φ,

$$[\circ]: \quad \frac{\Phi \vdash X : A \supset B,\ \Phi \vdash Y : C \supset A}{\Phi \vdash X \circ Y \equiv \lambda_* x{:}C.X(Yx) : C \supset B} \ [x \text{ is not in } X, Y].$$

(2) For all types/propositions A, B in $\mathcal{L}[\supset,\forall]$, such that u is not in $FV_u(A,B)$, and any context Φ

$$[\circ_u]: \quad \frac{\Phi \vdash X : A \supset B,\ \Phi \vdash Y : \forall u.B}{\Phi \vdash X \circ_u Y \equiv !_* u.X(Y[u]) : \forall u.B} \ [u \text{ is not in } X, Y].$$

Proof. [∘]: If x is not in X, Y and $\Phi \vdash X : A\supset B$, $\Phi \vdash Y : C\supset A$ then

$$\Phi \vdash \lambda_* x{:}C.X(Yx) \equiv \underline{S}[\![C,A,B]\!](\lambda_* x{:}C.X)(\lambda_* x{:}C.Yx) \equiv$$
$$\equiv \underline{S}[\![C,A,B]\!](\underline{K}[\![A\supset B,C]\!](X))(Y) \equiv_{df} X \circ Y : C \supset B.$$

[$\circ_u$]: Analogously. □

2.22 Exercises.

(1) There are, certainly, many alternative definitions for the Boolean combinators above. For instance, one could have had, for all types/propositions A, B, C, in $\mathcal{L}[\supset,\forall]$,

(11) $\underline{K}_*[\![A]\!] := \underline{K}[\![\top,A]\!]$.

(12) $\underline{K}'[\![A]\!] := \underline{K}[\![B\supset B,A]\!]\underline{I}[\![B]\!]$.

(13) $\underline{B}[\![A,B,C]\!] := \mathbf{S}_1(\mathbf{K}_1\mathbf{S}_2)\mathbf{K}_2$,
where the most general type-parametrizations are:
$\mathbf{S}_1 \equiv \underline{S}[\![A\supset B, C\supset. A\supset B, C\supset A\supset. C\supset B]\!]$
$\mathbf{S}_2 \equiv \underline{S}[\![C, A, B]\!]$
$\mathbf{K}_1 \equiv \underline{K}[\![C\supset(A\supset B)\supset. (C\supset A)\supset(C\supset B), A\supset B]\!]$
$\mathbf{K}_2 \equiv \underline{K}[\![A\supset B, C]\!]$

(14) $\underline{B}'[\![A,B,C]\!] := \mathbf{B}_1(\mathbf{S}_1\mathbf{B}_2)\mathbf{K}_1$,
with most general type-parametrizations:
$\mathbf{B}_1 \equiv \underline{B}[\![B\supset C\supset. A\supset B, B\supset C\supset. A\supset C, A\supset B]\!]$
$\mathbf{B}_2 \equiv \underline{B}[\![B, C, A]\!]$
$\mathbf{S}_1 \equiv \underline{S}[\![B\supset C, A\supset B, A\supset C]\!]$.
$\mathbf{K}_1 \equiv \underline{K}[\![A\supset B, B\supset C]\!]$.

(15) $\underline{C}_*[\![A,B]\!] := \underline{B}[\![A\supset B\supset A, A\supset B\supset B, A]\!](\underline{S}[\![C, A, B]\!]\underline{I}[\![A\supset B]\!])(\underline{K}[\![A, A\supset B]\!])$,
with, in particular,
$\underline{v}[\![A]\!] := \underline{B}[\![A^-\supset A, A^-, A]\!](\underline{S}[\![C, A, \bot]\!](\underline{I}[\![A^-]\!]))(\underline{K}[\![A, A^-]\!])$ $[\equiv \underline{C}_*[\![A, \bot]\!]]$.

(16) $\underline{B}_\cup[\![A,B]\!] := \underline{B}[\![C, D, E]\!](\underline{S}_\cup[\![A, B]\!])(\underline{K}_\cup[\![A\supset B]\!])$, [u not in $FV_\cup(A,B)$],
where $C \equiv \forall u.(A\supset B)$, $D \equiv [\forall u.A \supset \forall u.B]$ and $E \equiv [A \supset B]$.

(17) $\underline{C}_\cup[\![A,B]\!] := \underline{B}[\![C, D, E]\!](\underline{B}'[\![A, \forall u.A, \forall u.B]\!](\underline{K}_\cup[\![A]\!]))(\underline{S}_\cup[\![A, B]\!])$,
[u not in $FV_\cup(A)$],
where $C \equiv [\forall u.A\supset\forall u.B]$, $D \equiv [A \supset \forall u.B]$ and $E \equiv \forall u(A\supset B)$.

Show that (11)–(17) above yield the same typings as earlier.

(2) The standard definition of $\underline{C}[\![A,B,C]\!]$ is the one given by λ_*-expansion. Alternatively, one could have had, for instance, $\underline{C}[\![A,B,C]\!] := \mathbf{S}_1(\mathbf{B}_1 \circ \mathbf{S}_2)(\mathbf{K}_1\mathbf{K}_2)$, say, for appropriate type-instances $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{K}_1$, $\mathbf{K}_2$, $\mathbf{B}_1$ of $\underline{S}[\![A,B,C]\!]$, $\underline{K}[\![A,B]\!]$, $\underline{B}[\![A,B,C]\!]$. Establish the most general type-parametrization for $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{K}_1$, $\mathbf{K}_2$, $\mathbf{B}_1$ (and $\circ$), in the above.

(3) On the other hand, with both $\underline{B}[\![A,B,C]\!]$ and $\underline{C}[\![A,B,C]\!]$ among the primitives, one could have defined, in a straightforward way, e. g., $\underline{B}'[\![A,B,C]\!] := \underline{C}[\![B\supset C, A\supset B, A\supset C]\!](\underline{B}[\![B,C,A]\!])$, with the same typing as before.

(4) If $\underline{B}[\![C,A,B]\!]$ were among the primitives, one could have defined, alternatively, in a context Φ, $X \circ Y := \underline{B}[\![C,A,B]\!](X)(Y)$, for all types/propositions A, B, C in $\mathcal{L}[\supset,\forall]$, and all combinatory proof-terms X, Y such that $\Phi \vdash X : A \supset B$ and $\Phi \vdash Y : C \supset A$, Analogously, if $\underline{B}_\cup[\![A,B]\!]$ were among the primitives, one could have had, in any context Φ, $X \circ_\cup Y := \underline{B}_\cup[\![A,B]\!](X)(Y)$, for all types/propositions A, B in $\mathcal{L}[\supset,\forall]$ such that u is not in $FV_\cup(A,B)$, and all combinatory proof-terms X, Y such that $\Phi \vdash X : A \supset B$ and $\Phi \vdash Y : \forall u.B$. In such conditions, both properties [$\circ$], [$\circ_\cup$] above are preserved, with the extended definition of λ_* and $!_*$.

3. The type-theory of first-order classical logic. The set of primitives for the intended classical proof-notation consists of the following collection of symbols and syntactic proof-operators:

- ground forms:
  - x, y, z, ... proof-variables, possibly decorated,
  - Ω a primitive "proof" (a proof-constant),
- abstraction-forms:
  - λ(...) positive (λ-) abstraction (**while**),
  - ϑ(...) negative (ϑ-) abstraction (**until**),
  - !(...) *U*-abstraction (*U-generalization*),
- application-forms:
  - @(...,...) application (*message passing*),
  - @(...,...) *U*-*instantiation* (*U-parameter-passing*),
- separators: ( ) [ ] : . ,

The proof-variables make up a set Var_λ. The application operators @, @ are never displayed [see 3.3 (2)]. Ω represents, in intention, a primitive "proof" of an arbitrary non-logical "axiom" (in practice, we cope with particular first-order theories, not with "pure logic").

3.1 Definition.

The *proof-terms* (for short: *p-terms* or just *terms*), ranged over by a, b, c, ..., f, g, h, ..., possibly decorated, are then [only] of the form:

- x_n, y_n, z_n, ... p[roof]-variables [n ∈ ℕ],
- Ω a "primitive proof",
- λ([x:A].a[[x]]) positive abstractions ("deduction"),
- ϑ([x:A].a[[x]]) negative abstractions ("*reductio*"),
- @(f,a) applications ("message-passing"),
- !([u::*U*].a[[u]]) !-abstractions ("*U*-generalization"),
- @(f,t) *U*-applications ("*U*-parameter-passing"),

where A is a type, t is a *U*-term and, [[...]] is used to display (possibly vacuous) occurrences of a variable in a proof-term.

Terminology. Further, the *subterms* of proof-terms, the *free* and the *bound p-* resp. *U-variables* are supposed to be introduced in the expected way (here λ and ϑ are p-variable binders and ! is a *U*-variable binder). In particular, for p-terms a, $FV_\lambda(a)$, $BV_\lambda(a)$ stand resp. for the corresponding sets of p-variables. Analogously, $FV_U(a)$, $BV_U(a)$ refer to *U*-variables. It is convenient to have $FV(a) = FV_\lambda(a) \cup FV_U(a)$. A *closed p-term* (or a *proof-combinator*) is then a p-term a with $FV(a) = \emptyset$.

3.2 Remark (*α-conversion*).

As usual in languages with abstractors, proof-terms diferring in their bound variables only are identified. We ignore thus α-conversion subtleties and assume that the reader is able to perform correctly systematic α-reletterings. Formally, one must assume that $\equiv_\alpha \subseteq \equiv$, where $\equiv$ is syntactic identity and $\equiv_\alpha$ is a p-term congruence generated by:

(α_λ): $\lambda x{:}A.a[\![x]\!] \equiv_\alpha \lambda y{:}A.a[\![x:=y]\!]$, (y fresh for a),
$(\alpha_\forall)$: $\forall x{:}A.a[\![x]\!] \equiv_\alpha \forall y{:}A.a[\![x:=y]\!]$, (y fresh for a),
$(\alpha_!)$: $!u.a[\![u]\!] \equiv_\alpha !v.a[\![u:=v]\!]$, (v fresh for a).

3.3 Notation.

(1) *The p-term spelling.* We spare on parentheses by associating to the left and by omitting oft the outermost pair.

(2) We use also the following shorthand:

- $(\lambda x{:}A.a[\![x]\!]) := \lambda([x{:}A].a[\![x]\!])$,
- $(\forall x{:}A.a[\![x]\!]) := \forall([x{:}A].a[\![x]\!])$,
- $(ab) := @(a,b)$,

- $(!u.a[\![u]\!]) := !([u::\mathcal{U}].a)$,
- $(a[t]) := @(a,t)$.

(3) The substitution "operators" on proof-terms are "...$[\![x:=a]\!]$" and "...$[\![u:=t]\!]$"; they have the usual meaning ("x becomes a in ...", etc.).

3.4 Remark.

(1) In a first-order "application"-term of the form $a[t] := @(a,t)$ [*$\mathcal{U}$-instantiation at* t], the term t appearing in brackets is always a $\mathcal{U}$-term.

(2) $\mathcal{U}$ is not a type and is *always* discarded in notation. But the usual type/propositional labels on *p*-variable occurring within the scope of an abstractor (here: λ, ∀) are *always* mentioned explicitly.

Proof-contexts and stratification (*type-assignment*). The definition of the basic type-assignments (proof-contexts) makes up the initial clause of a recursion.

A proper *type-assignment* is an extension [as a map] of a context to a (finite) set of p-terms. So, ultimately, a type-assignment is *a partial map from p-terms to types/propositions.*

The *admissible type-assignments* for **CQ** are given by *typing rules.*

Syntactically, a typing rule displays the graph of type-assignments. One has, essentially, *t-statements* of the form $\Phi \vdash a : A$, where Φ is a context, a is a p-term and A is a type/proposition, such that a typing rule can be also seen as an operation on t-statements. The general form of a typing rule **(r)** is:

$$\textbf{(r)}: \quad (\Phi_1 \vdash a_1 : A_1) \;\&\; \ldots \;\&\; (\Phi_n \vdash a_n : A_n) \Longrightarrow (\Phi \vdash a : A)$$

where **&** and ==> stand for the metalinguistic conjunction and the metalinguistic conditional resp.

Alternatively, the previous display of rule (r) can be arranged vertically:

$$\textbf{(r)} \qquad \begin{array}{l} \Phi_1 \vdash a_1 : A_1 \\ \ldots \\ \Phi_n \vdash a_n : A_n \\ \hline \Phi \vdash a : A. \end{array} \quad (n \geq 0)$$

The *global context-rules* are similar to some of the so-called *structuhal rules* of a Gentzen *L*-system ("sequent system", [Gentzen 1935]), except for the fact that the latter ones apply to *sequences*.

3.5 Definition (*Context rules*).

Classical context rules. Let φ be "c : C".

$$\langle\Omega\rangle: \qquad [\;] \vdash \Omega : \mathbf{T},$$

$$\langle\mathbf{I}\rangle: \qquad [x : A] \vdash x : A,$$

$$\langle\mathbf{K}\rangle: \qquad \frac{\Phi \vdash \varphi}{\Phi[\; x : A \;] \vdash \varphi},$$

$$\langle\mathbf{K}\forall\rangle: \qquad \frac{\Phi \vdash \varphi}{\Phi[\; u :: \mathit{U} \;] \vdash \varphi}.$$

3.6 Remark.

(1) The last rule **<K∀>** has been added only in order to accomodate the *ad hoc* notations $\Phi[\; u :: \mathit{U} \;]$.

(2) The analogues of the so-called *Contraction* and *Exchange* (or *Permutation*) rules - occurring in Gentzen *L*-systems - are somewhat pointless in the present setting. Indeed, the usual *Contraction* rule

$$\langle\mathbf{W}\rangle: \qquad \frac{\Phi[\; x : A \;][\; x : A \;] \vdash \varphi}{\Phi[\; x : A \;] \vdash \varphi},$$

say, becomes a mere *notational diversion*, since Φ is a set and $\Phi[\ x : A\ ] = \Phi \cup \{\ [x : A]\ \} = \Phi \cup \{\ [x : A]\ \} \cup \{\ [x : A]\ \} = \Phi[\ x : A\ ][\ x : A\ ]$. The same is true about the *Exchange* rule.

(3) *Context redundancy*. The "context-redundancy elimination" rule

$$\langle \mathbf{Ke}\vdash\rangle: \qquad \frac{\Phi[\ x : A\ ] \vdash c : C}{\Phi \vdash c : C}, \quad (x \text{ not free in } c),$$

can be shown to be *admissible* in systems without such rules ("admissible" is taken, *mutatis mutandis*, in the Lorenzen-Curry sense, cf. [Lorenzen 1955], [Curry 1963]); see also [Rezus 1983]). In order to accomodate the $\Phi[\ u :: U\ ]$-notation, one could have had, analogously:

$$\langle \mathbf{K}\forall\mathbf{e}\vdash\rangle: \qquad \frac{\Phi[\ u :: U\ ] \vdash \varphi}{\Phi \vdash \varphi}, \quad (u \text{ not free in } \varphi),$$

3.7 Remark (*Context substitution rules*).

Substitution rules analogous to some **Automath** context-rules ([Rezus 1983,1986], [Barendregt & Rezus 1983]) can be easily shown to be *admissible*. *Examples*:

$$\langle \$\rangle: \qquad \frac{\Phi_1 \vdash a : A \qquad \Phi_2[\ x : A\ ] \vdash b[\![x]\!] : B}{\Phi_1\Phi_2 \vdash b[\![x:=a]\!] : B} \quad (x \text{ fresh for } \Phi_1,\Phi_2),$$

$$\langle \$_U\rangle: \qquad \frac{\Phi_1 \Vdash t :: U \quad (\equiv \text{"}t \text{ is a } U\text{-term"}) \qquad \Phi_2[\ u :: U\ ] \vdash b[\![u]\!] : A[\![u]\!]}{\Phi_1\Phi_2 \vdash b[\![u:=t]\!] : A[\![u:=t]\!]} \quad (u \text{ fresh for } \Phi_1,\Phi_2),$$

or, more generally,

$$\langle \$_U\rangle: \qquad \frac{\Phi_1 \Vdash t :: U \quad (\equiv \text{"}t \text{ is a } U\text{-term"}) \qquad \Phi_2[\![u]\!] \vdash b[\![u]\!] : A}{\Phi_1(\Phi_2[\![u:=t]\!]) \vdash b[\![u:=t]\!] : A[\![u:=t]\!]} \quad (u \text{ fresh for } \Phi_1).$$

Stratification in the proof-calculus of first-order classical logic. We introduce the *stratification structure* for a *typed* λ-*calculus* $\lambda\not!!$. $\lambda\not!!$ is *a proof-calculus for the first-order classical logic* **CQ**, formulated in the provability language $L[\supset,\forall]$ and can be viewed as an extension of the usual (extensional) typed λ-calculus λ. The "!"-less label $\lambda\not!$ identifies the *propositional* (quantifier-free) segment of the proof-calculus.

We display next only the derivation/proof rules meant to define a *stratification* ("*typing*") of p-terms. These rules consist of the former *context rules*, together with a set of *type-assignment rules*. They define a *concept of proof for* **CQ**, *via* a *type-assignment* **π[CQ]**. Essentially, **π[CQ]** is thus a "stratified" or a "typed" language, where the types are the "propositions" of **CQ**. In extension, it can be viewed as a set of *t-statements*, i. e., "typing" statements of the form $\Phi \vdash a : A$ (read "a proves A in context Φ").

Proof-equality (or *proof-conversion*) *for* **CQ** *relative to* **π[CQ]** is introduced effectively later, by means of the typed λ-calculus **λϑ!**, i. e., *via* a specific *notion of reduction* **r(π[CQ])**. As in the case of the ordinary (even "type-free") λ-calculus, the latter concept is described as a (binary) relation, defined explicitly on the "legal" ("stratifiable", "correct", "typable", etc.) terms of **π[CQ]**. So, as usual, equality/conversion in **λϑ!** is generated by *reduction rules*.

3.8 Definition.

The *derivation rules of the proof-system* **π[CQ]** are:

1 ***Context rules***: **<Ω>**, **<I>**, **<K>**, **<K∀>** (see above).

2 ***Type-assignment rules*** (*classical type-assignment*).

2.1 Basic type-assignment.

$$(\supset i\lambda):\ \frac{\Phi[\ x : A\ ] \vdash a[\![x]\!] : B}{\Phi \vdash \lambda x{:}A.a[\![x]\!] : A \supset B}, \qquad (\supset i\vartheta):\ \frac{\Phi[\ x : A^{-}\ ] \vdash a[\![x]\!] : \bot}{\Phi \vdash \vartheta x{:}A^{-}.a[\![x]\!] : A},$$

$$(\supset e):\ \frac{\Phi_1 \vdash f : A \supset B \quad \Phi_2 \vdash a : A}{\Phi_1,\Phi_2 \vdash fa : B},$$

2.2 First-order type-assignment. If u is not free in f [(∀e)],

$$(\forall i):\ \frac{\Phi[\ u :: U\ ] \vdash a[\![u]\!] : A[\![u]\!]}{\Phi \vdash !u.a[\![u]\!] : \forall u.A[\![u]\!]}, \qquad (\forall e):\ \frac{\Phi_1 \Vdash t :: U,\quad \Phi_2 \vdash f : \forall u.A[\![u]\!],}{\Phi_1,\Phi_2 \vdash f[\,t]: A[\![u:=t]\!].}$$

3.9 Remark.

It is appropriate to state explicitly the *restrictions on the use of variables appearing in the rules* above. Note that the *provisoes on contexts* in proper context rules have been stated separately. *Provisoes on contexts*. Concerning the statement of rules that are not proper context rules: the notation agrees with **Convention** $\Phi_{[U]}$ above. That is: the assumption on context extensions of the form $\Phi[\ x : C\ ]$ (resp. $\Phi[\ u :: U\ ]$) is - in such rules - that $[\ x : C\ ]$ (resp. $[\ u :: U\ ]$) does not occur in Φ. In detail, this should give:

in ($\supset i\lambda$): $[\ x : A\ ]$ is not in Φ,
in ($\supset i\not{}$): $[\ x : A^{-}]$ is not in Φ,
in ($\forall i$): $[\ u :: U\ ]$ is not in Φ.

3.10 Remark.

One can eliminate both Ω and $\top$ from $\pi[CQ]$, by setting:

$$\top := [\bot \supset \bot] \text{ and } \Omega := \lambda x{:}\bot.x.$$

Then rule $\langle\Omega\rangle$ is obtained from $\langle I\rangle$ and an instance of ($\supset i\lambda$):

$\langle I\rangle$: $[\ ][x : \bot] \vdash x : \bot$
($\supset i\lambda$): ---

$$[\ ] \vdash \lambda x{:}\bot.x \equiv_{\alpha\beta} \Omega : [\bot \supset \bot] \equiv_{\alpha\beta} \top.$$

[*Note*. This strategy does not preserve the reduction/equational behavior of classical proofs.]

3.11 Remark.

The usual logic text-book terminology identifies ($\supset i\lambda$) as "$\supset$-introduction" [or yet, in a different setting, as a "deduction metatheorem"], whereas ($\supset e$) is also known as "$\supset$-elimination" [as *modus ponens* or as "detachment rule"]. Further, ($\forall i$) and ($\forall e$) are known as "$\forall$-introduction" and "$\forall$-elimination" resp. [or as "universal generalization" and "instantiation" resp.]. ($\supset i\lambda$), ($\supset e$) are part of the syntax of the usual typed λ-calculus (Curry's *basic functionality theory*). Rule ($\supset i\not{}$) is *new* and formalizes the so called *reductio ad absurdum* or the principle of *indirect reasoning*.

Proof-term "correctness"; "type-checking". Recall that, in the above, $\pi[C]$ is the purely "propositional" fragment of $\pi[CQ]$.

As expected, the "type-checking" of a proof-term, i. e., the attempt to establish the "correctness" of its typing relative to $\pi[CQ]$, should correspond actually to "logic theorem proving", in traditional proof-theoretic terms.

We discuss next the theoretical basis of "type-checking" for the first-order classical proof-system **π[CQ]**. Mentioned first are a few elementary properties of "correct" proof-terms, i. e., those proof-terms that respect the type-assignment rules of **π[CQ]**.

3.12 Notation.

Let "Φ ⊢ a" be shorthand for "Φ ⊢ a : A, for some type A". *Note.* The intuitive [intended] reading of "Φ ⊢ a" is "a is a correct proof in **π[C(Q)]**". This notion is *relative to* the underlying provability language (here: $L[\supset,(\forall)]$ resp.).

One can easily prove the following facts, by inspecting the form of the proof-terms appearing in the conclusion of the type-assignment rules of **π[C(Q)]**.

3.13 Lemma (*Behavior of abstractions under typing*).

(1) Φ ⊢ λx:A.a[[x]] ==> Φ ⊢ λx:A.a[[x]] : (A ⊃ B), for some B,
(2) Φ ⊢ ∂x:C.a[[x]] ==> C ≡ A^{-}, for some A, with Φ ⊢ ∂x:A^{-}.a[[x]] : A,
(3) Φ ⊢ !u.a[[u]] ==> Φ ⊢ !u.a[[u]] : ∀u.A[[u]], for some A[[u]], (with u free in A[[u]]).

Proof. By a direct inspection of the typing rules. □

3.14 Lemma (*Behavior of application/instantiation under typing*).

(1) Φ ⊢ fa ==> Φ ⊢ f : A ⊃ B, Φ ⊢ a : A, for some A, B such that Φ ⊢ fa : B,
(2) Φ ⊢ f[t] ==> Φ ⊢ f[t] : A[[u:=t]], for some A[[u]] (u free in A[[u]] and t free for u in A[[u:=t]]) and some U-term t, with Φ ⊩ t :: U.

Proof. By a straightforward inspection of the typing rules. □

3.15 Theorem ("*Subterm correctness*").

(1) Φ ⊢ fc : A ==> Φ ⊢ f : C ⊃ A and Φ ⊢ c : C, for some C,
(2) Φ ⊢ λx:A.a[[x]] : (A ⊃ B) ==> Φ[x:A] ⊢ a[[x]] : B,
(3) Φ ⊢ ∂x:A^{-}.a[[x]] : A ==> Φ[x:A^{-}] ⊢ a[[x]] : ⊥,
(4) Φ ⊢ f[t] : A[[u:=t]] ==> Φ ⊢ f : ∀v.A[[v:=u]], Φ ⊩ t :: U, where t is free for u in A[[u]] and v is fresh for Φ and not free in A[[u:=t]], and f,
(5) Φ ⊢ !u.a[[u]] : ∀u.A[[u]] ==> Φ[u:U] ⊢ a[[u]] : A[[u]], where v is fresh for Φ.

Proof. In each case, by induction on the [length of] derivation of the premiss, using the preceeding Lemmas. □

3.16 Corollary.

Let a, b be proof-terms such that b is a subterm of a. If Φ ⊢ a then Φ' ⊢ b, for some Φ'.

Proof. By induction on the subterm structure of a, using **3.15**. [A *minimal* such Φ' can be determined explicitly.] □

3.17 Remark.

The *Subterm Correctness Theorem* and its corollary guarantee formally the existence of an appropriate *type-checking algorithm for first-order classical logic proofs*. [**NB**: For λ-free typed-languages, this is widely known from the **Automath** literature, as well as from work on so-called "polymorphic" typed functional programming languages, as, e. g., the Hindley-Milner-Cardelli **ML**-series. It has become popular to contrast "Church *vs* Curry" in this respect, although both points of view use the same "type-checking" strategies.]

3.18 Theorem (*Unicity of types*: [**UT**]).

If $\Phi \vdash a : A_1$ and $\Phi \vdash a : A_2$ then $A_1 \equiv A_2$.

Proof. Induction on the (subterm) structure of a, using the *Subterm Correctness Theorem*. □

Notation. So, we can write unambiguously, **type**(Φ,a) for A, whenever $\Phi \vdash a : A$, whence $\Phi \vdash a :$ **type**(Φ,a). In particular, if Φ is empty or can be retrieved from the immediate environment, we set **type**(a) $\equiv$ A.

"*Type-checking*": *heuristics*. We illustrate next a practical "type-checking" technique, based on traditional proof-methods (S. Jaśkowski, F. B. Fitch, N. G. de Bruijn). Roughly speaking, the meaning of the procedure is that, once a proof-term is *given*, the fact that it is/represents, indeed, a classical proof - in the technical sense of π[CQ] - can be also *recognized by automatic means*. The Jaśkowski/Fitch/de Bruijn technique consists of visualizing proof-structures *via* "nested blocks".

The closed proof-terms stratified by the typing rules of π[CQ] are called *Boolean combinators*. In order to see how the language works, some *examples* are useful.

3.19 Remark.

The "type-checking" of (first-order) Boolean proof-terms admits of a "reversal" ("Beth search"), refining the familiar *Beth tableaux* method (or, alternatively, the so-called "dialogic" approach, cf. [Lorenzen & Lorenz 1978]) into an *algorithmic technique*. The latter method aims at *finding* a proof-term, for a *given* type/proposition.

As ever, in what follows, type-parametrizations are shown explicitly.

Since, for λ-free languages, the method should be familiar from the ordinary typed λ-calculus, we display, in detail, only "genuinely classical" closed proof-terms. More examples will occur subsequently.

3.20 Examples (*Boolean proof-combinators*).

(1°) *The Łukasiewicz axiom for material implication*. Consider the type/proposition [Łukasiewicz 1948]:

(**Ł**) $A \supset B \supset C \supset .C \supset A \supset (D \supset A)$.

This is also known to be the *shortest* axiom for the "pure material implication" (in axiomatizations with $[\supset e]_o$, i. e., *modus ponens*, as sole derivation rule). Define:

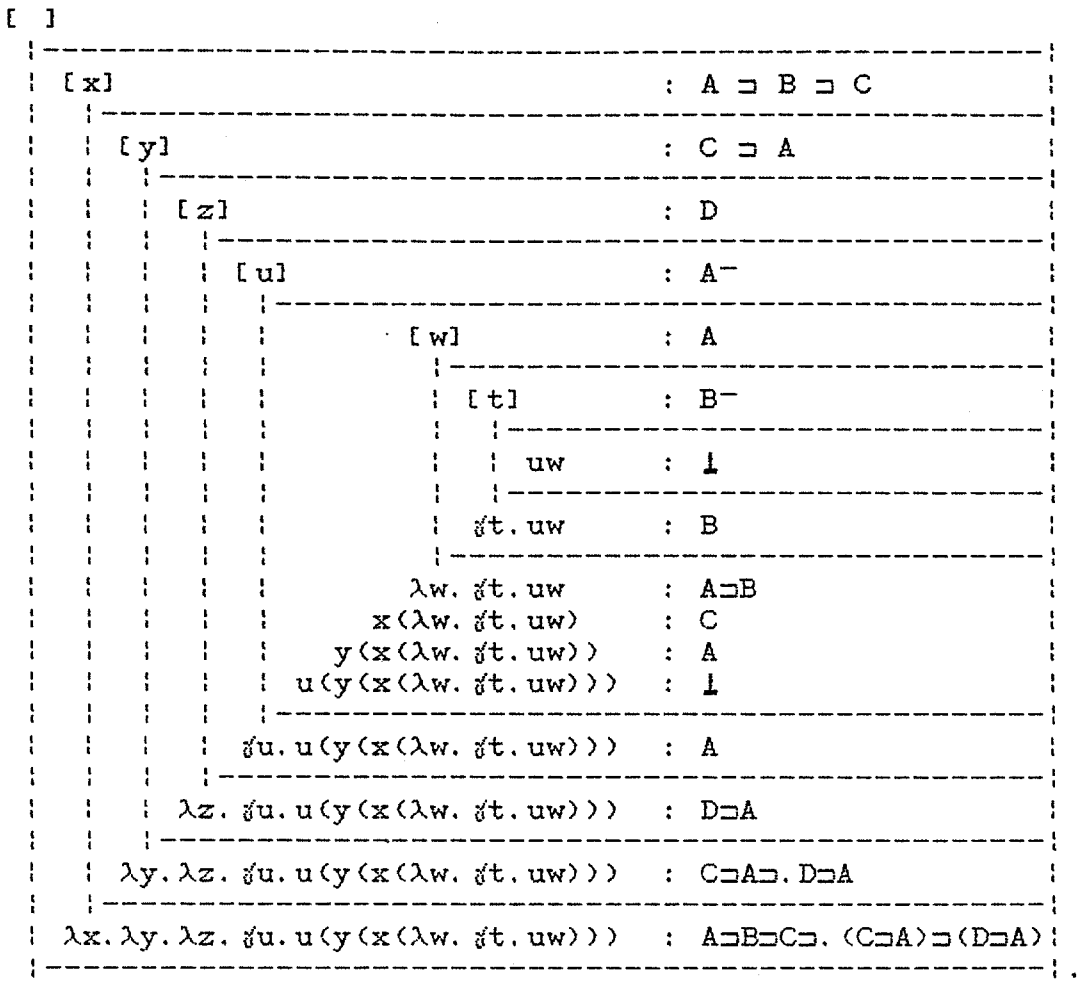
$\mathbf{Ł}[\![A,B,C,D]\!] := \lambda x.\lambda y.\lambda z.\,\eth u.u(y(x(\lambda w.\,\eth t.uw)))$

with $[x : A \supset B \supset C][y : C \supset A][z : D][u : A^-][w : A][t : B^-]$,

where the types of bound p-variables occurring within the scope of a λ- or a $\eth$-abstractor have been extracted under a *local* **with**-declaration. One can check:

$[\,] \vdash \mathbf{Ł}[\![A,B,C,D]\!] : A \supset B \supset C \supset .C \supset A \supset (D \supset A)$.

The standard "type-checking" of this proof-term is a *natural deduction proof* of (**Ł**), using only *modus ponens*, the *deduction theorem* and *reductio ad absurdum* (the rules of $\pi[C]$). The so-called "Fitch-style" ["nested-blocks"] presentation of the proof [Fitch 1952] is an elliptic form of the type-checking.



One should realize that the proof-term $\mathfrak{L}[\![A,B,C,D]\!]$ delivers *complete information* as to the way of recovering the full display.

Note. This display "style" is an intuitive visualization of a technique due, independently, to S. Jaśkowski (1926), F. B. Fitch (*circa* 1950) and N. G. de Bruijn (1967-1968). In the latter case, it reflects also a specific method of codifying *mathematical proof-texts*, in view of *computer-assisted proof-checking*, as used in **Automath** (see, e.g., [Rezus 1983], for background and further details).

A $\mathfrak{J}$-free variant of the "nested blocks" method has been also implemented in a specific **Automath** language; cf. [Jutting 1979] and [de Bruijn 1980], for instance. N. G. de Bruijn [1987] has proposed a *pure combinatorial representation* of the "nested blocks" structuring, in terms of labelled trees, best suited for the *automatic processing of proof-operations*.

(2°) *Peirce's Law*. The following type/proposition, known also as *Peirce's Law* (C. S. Peirce 1885),

($P_\bot$) [*Peirce*] $A \supset B \supset A \supset A$,

has a "proof" $P_\bot[\![A,B]\!] := \lambda x.\, \mathcal{J} y.\, y(x(\lambda z.\, \mathcal{J} u.\, yz))$, **with** [x:A⊃B⊃A] [y:A⁻][z:A][u:B⁻] and the following standard "type-checking" [printed, more economically, as]:

```
[x]                          : A ⊃ B ⊃ A
 ¦ [y]                       : A⁻
 ¦  ¦      [z]               : A
 ¦  ¦       ¦ [u]            : B⁻
 ¦  ¦       ¦  ¦ yz          : ⊥
 ¦  ¦       ¦ 𝒥u.yz          : B
 ¦  ¦      λz.𝒥u.yz          : A ⊃ B
 ¦  ¦   x(λz.𝒥u.yz)          : A
 ¦  ¦   x(λz.𝒥u.yz)          : A
 ¦  ¦ y(x(λz.𝒥u.yz))        : ⊥
 ¦ 𝒥y.y(x(λz.𝒥u.yz))        : A
λx.𝒥y.y(x(λz.𝒥u.yz))        : A ⊃ B ⊃ A ⊃ A.
```

(3°) *Ex falso quodlibet*. The λ𝒥-term $\omega[\![A]\!] := \lambda x{:}\bot.\, \mathcal{J} y{:}A^-.\, x$ "proves" the Heyting-specific proposition (beyond Johansson's [1936] *Minimalkalkül*) (ω) [*ex falso quodlibet*]: $\bot \supset A$.

```
[x]                          : ⊥
 ¦ [y]                       : A⁻
 ¦  ¦ x                      : ⊥
 ¦ 𝒥y.x                      : A
λx.𝒥y.x                      : ⊥ ⊃ A.
```

(4°) *Duplex negatio affirmat*. With $\Delta[\![A]\!] := \lambda f{:}A^{--}.\, \mathcal{J} x{:}A^-.\, fx$, for all types/propositions A, one has $[\,] \vdash \Delta[\![A]\!] : A^{--} \supset A$. Contrast this with $I[\![A]\!]$ and $1[\![A,B]\!]$ (where $I[\![A]\!] := \lambda x{:}A.\, x$ with $[\,] \vdash I[\![A]\!] : A \supset A$ and $1[\![A,B]\!] := \lambda f{:}(A \supset B).\, \lambda x{:}A.\, fx$, with $[\,] \vdash 1[\![A,B]\!] : A \supset B \supset .A \supset B$).

3.21 Remark.

So **π[C]**, the "basic" fragment of **π[CQ]** contains actually the full classical *propositional* logic **C**, in the language $L[\supset]$. Indeed, **T** [as a **CQ**-theorem, taken care by (Ω)], together with ($\mathcal{J}$), (ω) and $[\supset e]_o$ are also known to suffice for a complete axiomatization of **C** in $L[\supset]$. However, **π[C]** "contains" more than mere "provability", it *codifies the proofs themselves*.

The lists of proof-combinators displayed next have a technical use later. For the moment being, they can be taken as *examples* or as training material [*exercises*] in type-checking closed proof-terms.

Boolean proof-combinators. The classical proof-system **π[CQ]** admits of a *finitary formulation* in terms of proof-combinators.

In traditional proof-theoretic parlance, this points out to specific *axiomatizability results*. In combinatory (logic) terms, one has a "*theory of Meredith proofs*" [Rezus 1982].

The rest of the section shows that the proof-system $\pi[CQ]$ *contains* the corresponding first-order logic **CQ**, with implicit reference to appropriate axiomatics. The fact that it is not "more" will be established later, *via* the combinatory formulation $\pi_{\omega}[CQ]$.

3.22 Definition (*Ground Boolean proof-combinators*).

For all types/propositions A, B, C,

(1) (*Basic Boolean combinators*)

$$\begin{aligned}
&\mathbf{I}[\![A]\!] && := \lambda x{:}A.\,x,\\
&\mathbf{K}[\![A,B]\!] && := \lambda x{:}A.\lambda y{:}B.\,x,\\
&\mathbf{S}[\![A,B,C]\!] && := \lambda x{:}(A\supset.\,B\supset C).\lambda y{:}(A\supset B).\lambda z{:}A.\,xz(yz),\\
&\Delta[\![A]\!] && := \lambda x{:}A^{-}.\,?y{:}A^{-}.\,xy,
\end{aligned}$$

(2) (*First-order Boolean combinators*)

$$\begin{aligned}
&\mathbf{K}_{\cup}[\![A]\!] && := \lambda x{:}A.\,!u.\,x,\ [u \text{ not in } FV_{\cup}(A)],\\
&\mathbf{S}_{\cup}[\![A,B]\!] && := \lambda x{:}(\forall u.\,(A\supset B)).\lambda y{:}(\forall u.\,A).\,!u.\,x[u](y[u]),\\
&\mathbf{\Theta}_{\cup}[\![A,B]\!] && := \lambda x{:}(A\supset\forall u.\,B).\,!u.\lambda y{:}A.\,(xy)[u],\ [u \text{ not in } FV_{\cup}(A)],\\
&\mathbf{\Phi}_{\cup}[\![A]\!] && := \lambda x{:}(\forall u.\forall v.\,A).\,!v.\,!u.\,x[u][v].
\end{aligned}$$

3.23 Remark.

(1) In the preceeding list, some ground Boolean combinators can be obtained from other ones, also present in the list, by explicit definitions. A typical case in point is $\mathbf{I}[\![A]\!]$.

(2) The following proof-combinators can be also defined in terms of ground Boolean proof-combinators (for all types/propositions A, B, C) and $\cup$-terms t:

$$\begin{aligned}
&\mathbf{K}_{*}[\![A]\!] && := \lambda x{:}\top.\lambda y{:}A.\,\Omega,\\
&\mathbf{K}'[\![A,B]\!] && := \lambda x{:}A.\lambda y{:}B.\,y,\\
&\mathbf{B}[\![A,B,C]\!] && := \lambda x{:}(A\supset B).\lambda y{:}(C\supset A).\lambda z{:}C.\,x(yz),\\
&\mathbf{B}'[\![A,B,C]\!] && := \lambda x{:}(A\supset B).\lambda y{:}(B\supset C).\lambda z{:}A.\,y(xz),\\
&\mathbf{C}[\![A,B,C]\!] && := \lambda x{:}(A\supset.\,B\supset C).\lambda y{:}B.\lambda z{:}A.\,xzy,\\
&\mathbf{C}_{*}[\![A,B]\!] && := \lambda x{:}A.\lambda y{:}(A\supset B).\,yx,\\
&\mathbf{W}[\![A,B]\!] && := \lambda x{:}(A\supset.\,A\supset B).\lambda y{:}A.\,xyy,\\
&\mathbf{B}_{\cup}[\![A,B]\!] && := \lambda x{:}(A\supset B).\lambda y{:}(\forall u.\,A).\,!u.\,x(y[u])\ [u \text{ not in } FV_{\cup}(A,B)],\\
&\mathbf{C}_{\cup}[\![A,B]\!] && := \lambda x{:}(\forall u.\,(A\supset B)).\lambda y{:}A.\,!u.\,x[u]y\ [u \text{ not in } FV_{\cup}(A)],\\
&\mathbf{\Theta}_{*\cup}[\![A]\!] && := !u.\lambda x{:}(\forall v.\,A[\![v]\!]).\,x[u],\ [u \text{ not in } FV_{\cup}(A[\![v]\!])],\\
&\mathbf{\Theta}_{t}[\![A]\!] && := \lambda x{:}(\forall u.\,A[\![u]\!]).\,x[t],\ [u \text{ free for } t \text{ in } A[\![u]\!]].
\end{aligned}$$

3.24 Notation.

(1) For all types A, $\nabla[\![A]\!] := \mathbf{C}_*[\![A, \bot]\!]$ [$\equiv \lambda x{:}A.\lambda y{:}A^-.yx$].

(2) For all types A, B, C, any context Φ, and proof-terms a, b such that $\Phi \vdash a : A \supset B$, $\Phi \vdash b : C \supset A$, set, in context Φ,

$$a \circ b := \lambda z{:}C.a(bz),$$

provided z is not in $FV_\lambda(a) \cup FV_\lambda(b)$.

(3) For all types A, B, any context Φ, and proof-terms a, b such that $\Phi \vdash a : A \supset B$ and $\Phi \vdash b : \forall u.A$, set, in context Φ,

$$a \circ_\cup b := !u.a(b[u]),$$

provided u is not in $FV_\lambda(a) \cup FV_\lambda(b) \cup FV_\cup(A) \cup FV_\cup(B)$.

3.25 Fact (*"Type-checking" Boolean proof-combinators*).

(1) For all types A, B, C, one has, in the empty context,

(**I**) $\vdash \mathbf{I}[\![A]\!] : A \supset A$,
(**K**) $\vdash \mathbf{K}[\![A,B]\!] : A \supset .B \supset A$,
(**S**) $\vdash \mathbf{S}[\![A,B,C]\!] : A \supset (B \supset C) \supset .A \supset B \supset (A \supset C)$,
(Δ) $\vdash \Delta[\![A]\!] : A^- \supset A$ [$\equiv A \supset \bot \supset \bot \supset A$],

($\mathbf{K}_\cup$) $\vdash \mathbf{K}_\cup[\![A]\!] : A \supset \forall u.A$ [u not in $FV_\cup(A)$],
($\mathbf{S}_\cup$) $\vdash \mathbf{S}_\cup[\![A,B]\!] : \forall u.(A \supset B) \supset .\forall u.A \supset \forall u.B$,
($\Phi_\cup$) $\vdash \Phi_\cup[\![A]\!] : \forall u.\forall v.A \supset \forall v.\forall u.A$,
($\Theta_\cup$) $\vdash \Theta_\cup[\![A,B]\!] : A \supset \forall u.B \supset \forall u.(A \supset B)$ [u not in $FV_\cup(A)$].

(2) Similarly, for all types A, B, C, in the empty context,

($\mathbf{K}_*$) $\vdash \mathbf{K}_*[\![A]\!] : \mathbf{T} \supset .A \supset \mathbf{T}$,
(**K'**) $\vdash \mathbf{K}'[\![A,B]\!] : A \supset .B \supset B$,

(**B**) $\vdash \mathbf{B}[\![A,B,C]\!] : A \supset B \supset .C \supset A \supset (C \supset B)$,
(**C**) $\vdash \mathbf{C}[\![A,B,C]\!] : A \supset (B \supset C) \supset .B \supset (A \supset C)$,

(**B'**) $\vdash \mathbf{B}'[\![A,B,C]\!] : A \supset B \supset .B \supset C \supset (A \supset C)$,
($\mathbf{C}_*$) $\vdash \mathbf{C}_*[\![A,B]\!] : A \supset .A \supset B \supset B$,
(∇) $\vdash \nabla[\![A]\!] : A \supset A^-$ [$\equiv A \supset .A \supset \bot \supset \bot$],
(**W**) $\vdash \mathbf{W}[\![A,B]\!] : (A \supset .A \supset B) \supset .A \supset B$.

($\mathbf{B}_\cup$) $\vdash \mathbf{B}_\cup[\![A,B]\!] : A \supset B \supset .\forall u.A \supset \forall u.B$ [u not in $FV_\cup(A,B)$].
($\mathbf{C}_\cup$) $\vdash \mathbf{C}_\cup[\![A,B]\!] : \forall u.(A \supset B[\![u]\!]) \supset .A \supset \forall u.B[\![u]\!]$ [u not in $FV_\cup(A)$].

(3) For all types A[u], if v is not free in A[u],

(Θ_*) [ ] $\vdash \Theta_{*\cup}[\![A]\!] : \forall u.(\forall v.A[\![u:=v]\!] \supset A[\![u]\!])$.

(4) For all types A[u], any context Φ, and all $\mathcal{U}$-terms t such that u is free for t in A[u], and v is not free in A[u],

(Θ_t) $\Phi \Vdash t :: \mathcal{U} \Longrightarrow \Phi \vdash \Theta_t[A] : \forall v.A[v] \supset A[u:=t]$, whence also

(Θ_t) $\Phi \Vdash t :: \mathcal{U}$, $\Phi \vdash f : \forall v.A[v] \Longrightarrow \Phi \vdash \Theta_t[A](f) : A[u:=t]$.

(5) For all types A, B, C and any context Φ,

$$(\circ)\quad \frac{\Phi \vdash a : A \supset B,\ \Phi \vdash b : C \supset A}{\Phi \vdash a \circ b : C \supset B},$$

$$(\circ_{\mathcal{U}})\quad \frac{\Phi \vdash a : A \supset B,\ \Phi \vdash b : \forall u.A}{\Phi \vdash a \circ_{\mathcal{U}} b : \forall u.B}\ [u \text{ not in } FV_{\mathcal{U}}(A,B)].$$

Proof. [*Exercise*]. □

4. Completeness relative to classical provability. We can now establish the fact that both points of view of *provability*: the *combinatory* formalism π_c[CQ] and the $\lambda\gamma$-formalism π[CQ] are equivalent. Moreover, both formalisms are *complete relative to the classical concept of first-order provability.*

Let next Term(π[CQ]) be the set of $\lambda\gamma!$-proof-terms and Term(π_c[CQ]) be the set of combinatory proof-terms.

We assume, for convenience, that the set Var_λ of p-variables of π[CQ] and the set Var_c of p-variables of π_c[CQ] coincide and that π[CQ] and π_c[CQ] have the same sets of U-terms.

Also, for any primitive Boolean proof-combinator $\underline{X}$ in π_c[CQ] (viz., $\underline{I}[\![A]\!]$, $\underline{K}[\![A,B]\!]$, $\underline{S}[\![A,B,C]\!]$, $\underline{\Delta}[\![A]\!]$, $\underline{K}_U[\![A]\!]$, $\underline{S}_U[\![A,B]\!]$, $\underline{@}_U[\![A,B]\!]$ or $\underline{\Phi}_U[\![A]\!]$), let $\underline{X}^\wedge$ be the corresponding $\lambda\gamma!$-proof-term, as above. E. g., if $\underline{X}$ is $\underline{I}[\![A]\!]$, then $\underline{I}[\![A]\!]^\wedge$ is the p-term $I[\![A]\!] \equiv \lambda x{:}A.x$.

What follows is similar to (a part of) the standard equivalence (type-free) argument: combinatory logic <==> $\lambda\beta\eta$-calculus (cf., *mutatis mutandis*, for instance, [Stenlund 1972], [Rezus 1981] or [Hindley & Seldin 1986]).

4.1 Definition.

(1) Let $(\ldots)^c$: Term(π[CQ]) --> Term(π_c[CQ]) be a map defined by:

- $(x)^c = x$, if x is a p-variable
- $(\Omega)^c = \underline{\Omega}$,
- $(fc)^c = (f)^c(c)^c$,
- $(\lambda x{:}A.b)^c = \lambda_* x{:}A.(b)^c$,
- $(\gamma x{:}A.b)^c = \gamma_* x{:}A.(b)^c\ [\ \equiv_{df} \underline{\Delta}[\![A]\!](\lambda_* x{:}A.(b)^c)\ ]$,
- $(f[t])^c = (f)^c[t]$,
- $(!u.a)^c = !_* u.(a)^c$.

(2) Let $(\ldots)^\llcorner$: Term(π_c[CQ]) --> Term(π_c[CQ]) be a map defined by:

- $(x)^\llcorner = x$, if x is a p-variable,
- $(\underline{\Omega})^\llcorner = \Omega$,
- $(\underline{X})^\llcorner = \underline{X}^\wedge$, for any primitive proof-combinator $\underline{X}$,
- $(!_\sqcup.X)^\llcorner = !u.(X)^\llcorner$, for any ground proof-combinator $!_\sqcup.X$,
- $(XY)^\llcorner = (X)^\llcorner(Y)^\llcorner$,
- $(X[t])^\llcorner = (X)^\llcorner[t]$.

4.2 Lemma

(1) For any primitive proof-combinator $\underline{X}$ in π_c[CQ], $[\ ] \vdash (\underline{X}^\wedge)^c \equiv \underline{X}$.

(2) For any ground proof-combinator X in π_c[CQ], $[\ ] \vdash (X^\llcorner)^c \equiv X$, [viz., $(X^\llcorner)^c$ is also a ground proof-combinator].

Proof. (1) We need only check the individual cases:

$$[\ ] \vdash (\mathbf{I}[\![A]\!])^c \equiv (\lambda x{:}A.x)^c \equiv \lambda_* x{:}A.x \equiv \underline{\mathbf{I}}[\![A]\!].$$

$$[\ ] \vdash (\mathbf{K}[\![A,B]\!])^c \equiv (\lambda x{:}A.\lambda y{:}B.x)^c \equiv \lambda_* x{:}A.\lambda_* y{:}B.x \equiv \underline{\mathbf{K}}[\![A,B]\!].$$

$$\begin{aligned}[\ ] \vdash (\mathbf{S}[\![A,B,C]\!])^c &\equiv (\lambda x{:}(A\supset.B\supset C).\lambda y{:}(A\supset B).\lambda z{:}A.xz(yz))^c \equiv \\ &\equiv \lambda_* x{:}(A\supset(B\supset C)).\lambda_* y{:}(A\supset B).\lambda_* z{:}A.xz(yz) \equiv \\ &\equiv \underline{\mathbf{S}}[\![A,B,C]\!].\end{aligned}$$

If u is not in $FV_\cup(A)$:

$$[\ ] \vdash (\mathbf{K}_\cup[\![A]\!])^c \equiv (\lambda x{:}A.!u.x)^c \equiv \lambda_* x{:}A.!_* u.x \equiv \underline{\mathbf{K}}_\cup[\![A]\!].$$

$$\begin{aligned}[\ ] \vdash (\mathbf{S}_\cup[\![A,B]\!])^c &\equiv (\lambda x{:}(\forall u.(A\supset B)).\lambda y{:}(\forall u.A).!u.x[u](y[u])) \equiv \\ &\equiv \lambda_* x{:}(\forall u.(A\supset B)).\lambda_* y{:}(\forall u.A).!_* u.x[u](y[u]) \equiv \\ &\equiv \underline{\mathbf{S}}_\cup[\![A,B]\!].\end{aligned}$$

If u is not in $FV_\cup(A)$:

$$\begin{aligned}[\ ] \vdash (\mathbf{@}_\cup[\![A,B]\!])^c &\equiv (\lambda x{:}(A\supset\forall u.B).!u.\lambda y{:}A.(xy)[u])^c \equiv \\ &\equiv \lambda_* x{:}(A\supset\forall u.B).!_* u.\lambda_* y{:}A.(xy)[u] \equiv \\ &\equiv \underline{\mathbf{@}}_\cup[\![A,B]\!].\end{aligned}$$

$$\begin{aligned}[\ ] \vdash (\mathbf{\Phi}_\cup[\![A]\!])^c &\equiv (\lambda x{:}(\forall u.\forall v.A).!v.!u.x[u][v])^c \equiv \\ &\equiv \lambda_* x{:}(\forall u.\forall v.A).!_* v.!_* u.x[u][v] \equiv \\ &\equiv \underline{\mathbf{\Phi}}_\cup[\![A]\!].\end{aligned}$$

(2) If X is a ground Boolean combinator then $X \equiv !_\cup.Y[\![u]\!]$, where $Y[\![u]\!]$ is either a primitive (ground) proof-combinator or a ground proof-combinator of the form $!_\vee.Z[\![u,v]\!]$, for some $Z[\![u,v]\!]$. The result follows from (1) by induction on the length of the !-prefix of X. □

4.3 Theorem.

> For any (stratified) combinatory proof-term X in $\pi_*[\mathbf{CQ}]$, such that $\Phi \vdash_c X : A$, for some context Φ and some type/proposition A, one has $\Phi \vdash_c (X^\llcorner)^c \equiv X$.

Proof. By induction on the structure of X. Clearly, by definition, $(x^\llcorner)^c \equiv (x)^c \equiv x$, for any p-variable. By definition, $(\underline{\Omega}^\llcorner)^c \equiv \underline{\Omega}$. By **4.2** (1) above, if X is a primitive Boolean combinator $\underline{X}$, then $[\ ] \vdash (\underline{X}^\llcorner)^c \equiv (\underline{X}^\wedge)^c \equiv \underline{X}$, since, by the definition of $(\ldots)^\llcorner$, one has $\underline{X}^\llcorner \equiv \underline{X}^\wedge$. If $!_\cup.X$ is a ground Boolean combinator then so is X and $((X)^\llcorner)^c$, whence

$$((!_\cup.X)^\llcorner)^c \equiv (!u.(X)^\llcorner)^c \equiv !_* u.((X)^\llcorner)^c \equiv_{\mathrm{df}} !_\cup.((X)^\llcorner)^c \equiv !_\cup.X,$$

by **4.2** (2).

The (IH) yields the remaining cases:

$$((XY)^\llcorner)^c \equiv ((X)^\llcorner(Y)^\llcorner)^c \equiv ((X)^\llcorner)^c)((Y)^\llcorner)^c,$$

$$((X[t])^\llcorner)^c \equiv ((X)^\llcorner[t])^c \equiv ((X)^\llcorner)^c[t]. \ \square$$

4.4 Theorem (*Combinatory Embedding for* $\pi[CQ]$).

For all types/propositions A in $L[\supset,\forall]$, all proof-terms a in Term($\pi[CQ]$), and all contexts Φ,

$\Phi \vdash a : A \Longrightarrow \Phi \vdash_c (a)^c : A$.

Proof. By induction on the [length of] derivation of the premiss. □

4.5 Theorem ($\lambda\beta!$*-Embedding for* $\pi_c[CQ]$).

For all types/propositions A in $L[\supset,\forall]$, all combinatory proof-terms X in $\pi_c[CQ]$, and all contexts Φ,

$\Phi \vdash_c X : A \Longrightarrow \Phi \vdash (X)^\vdash : A$,

Proof. By induction on the [length of] derivation of the premiss. □

Finally, this yields the following (expected) completeness result.

4.6 Theorem (*Proof-completeness*).

For all types/propositions A in $L[\supset,\forall]$, the following meta-statements are equivalent:

(1) $CQ \Vdash A$ [= A is provable in first-order classical logic].

(2) $\vDash X : A$, in $\pi_c[CQ]$, for some Boolean proof-combinator X.

(3) $[\,] \vdash a : A$, in $\pi[CQ]$, for some (closed) p-term a.

Proof. (1) <==> (2) is straightforward, provided one realizes the fact that $\pi_c[CQ]$ is nothing but an explicit standard axiomatization of CQ. (2) ==> (3) follows from the *Combinatory Embedding Theorem* above, with $a \equiv (X)^\vdash$. The converse implication (3) ==> (2) follows by the $\lambda\beta!$*-Embedding Theorem* above, with $X \equiv (a)^c$. □

5. A proof-calculus for first-order classical logic. We introduce next a concept of *proof-equality* (or *proof-conversion*) *for* **CQ** *relative to the concept of proof* (formalized in) **π[CQ]**, by means of a typed λ-calculus **λ∂!**, that is, *via* a specific *notion of reduction* **r(π[CQ])**. The latter one is a formal counterpart of the intuitive notion of a *proof-détour elimination* in **CQ**.

The *détour eliminations* are analyzed explicitly by (1) *evaluation rules*, (2) ***reductio*** *rules* and (3) *compatibility rules*.

The first group (1) contains rules that are similar to the usual reduction ["evaluation"] rules of the typed λ-calculus: in fact, they are identical to the proof-rules needed to formalize the [⊃,∀]-fragment of the Heyting calculus.

The ***reductio*** rules of group (2) characterize the "classical" notion of proof-reduction. They are, essentially, meant to estimate the complexity of proofs by *reductio ad absurdum* in classical logic.

Finally, the *compatibility* rules (3) are technical (but trivial) in nature and are intended - as usual in λ-calculi - to insure the fact that the equality relation generated by the reduction rules is a congruence relative to the syntactic operations of the language.

In particular, **λ∂!** appears to be *a typed λ-calculus*, namely it can be viewed as a proper (and conservative) extension of the ordinary typed λ-calculus **λ**. If = is the equality generated by the reduction relation of **λ∂!**, *the proof-theory* (**λ∂!**, =) is *a Post-consistent extension* of **λ**, [i. e., using the standard λ-calculus terminology, **λ∂!** is a *typed λ-theory*], whose types are the "propositions" of the first-order classical logic **CQ**.

At a later stage, the "***reductio***" rules are shown to be - roughly speaking - equivalent to *a systematic abbreviation device*. This motivates the restriction to "type-normal" proof-theories, i. e., to proof-calculi where *reductio ad absurdum* is only applied to atomic types.

5.1 Note.

(1) The standard notion of reduction of **λ∂!** is denoted by =», whereas the equality generated by =» is denoted by =. If necessary, the non-transitive counterpart of =» (*proof-contraction*) is denoted by -».

(2) With **&** ("and") in metalanguage, we write Φ ⊢ a =» b : A for

(Φ ⊢ a : A) & (Φ ⊢ b : A) & (Φ ⊢ a =» b)

The parenthesized notation Φ ⊢ a =» b [: A] means that the information appearing in brackets is redundant (viz., the actual form of A can be restored by type-assignment rules).

(3) *Proof-contraction, proof-reduction and proof-equality/conversion.* *Proof-reduction* [=»] is reflexive and transitive as a relation on proof-terms (relative to typing), while *proof-contraction* [-»] is reflexive but not transitive. That is, formally:

$$(\rho):\ \frac{\Phi \vdash a : A}{\Phi \vdash a \twoheadrightarrow a\ [: A]},\qquad \frac{\Phi \vdash a : A}{\Phi \vdash a \Rightarrow a\ [: A]},\qquad (\tau):\ \frac{\Phi_1 \vdash a \Rightarrow b : A \quad \Phi_2 \vdash b \Rightarrow c : A}{\Phi_1,\Phi_2 \vdash a \Rightarrow c\ [: A]}.$$

Further, (τ) excepted, -» is assumed to satisfy the same rules as =». This means that, with -» in the primitive setting, one could have introduced =», by stating only -»-rules of the form below, together with, as specific assumptions on =», (τ) and the following "-»,=»-containment" rule:

$$(\Rightarrow):\ \frac{\Phi \vdash a \twoheadrightarrow b : A}{\Phi \vdash a \Rightarrow b\ [: A]}.$$

Analogously, *proof-equality* or *proof-conversion* = [that is: the equivalence generated by -»] satisfies the same rules as -», together with (τ) and

$$(\sigma):\ \frac{\Phi \vdash a = b : A}{\Phi \vdash b = a\ [: A]},$$

whence, with -» [resp. -» and =»] in the primitive setting, one could have introduced =, by stating only -»-rules [resp. -»-rules and the specific =»-rules (τ), (-»)], of the form above, together with, as specific assumptions on =, (σ), (τ), and a "-»,=-containment" rule of the form:

$$(=):\ \frac{\Phi \vdash a \twoheadrightarrow b : A}{\Phi \vdash a \Rightarrow b\ [: A]}.$$

[resp. the obvious "=»,=-containment" rule].

The ***derivation rules of the classical proof-calculus*** **λ∀!** are:

5.2 Definition. (*First-order classical proof-rules*).

1 ***Evaluation rules.***

1.1 Basic evaluation.

$$(\beta\supset\lambda):\ \frac{\Phi_1 \vdash a : A \quad \Phi_2[\ x : A\ ] \vdash b[\![x]\!] : B}{\Phi_1,\Phi_2 \vdash (\lambda x{:}A.\, b[\![x]\!])a \Rightarrow b[\![x:=a]\!]\ [: B]}.$$

$$(\eta\supset\lambda):\quad \frac{\Phi \vdash f : A \supset B}{\Phi \vdash \lambda x{:}A.fx \Rightarrow f},\ (x \text{ not free in } f),$$

1.2 First-order evaluation.

$$(\beta\forall):\quad \frac{\Phi_1 \Vdash t :: U \qquad \Phi_2[\, u :: U\,] \vdash a[\![u]\!] : A[\![u]\!]}{\Phi_1,\Phi_2 \vdash (!u.a[\![u]\!])[t] \Rightarrow a[\![u:=t]\!]\ [:\ A[\![u:=t]\!]]},$$

$$(\eta\forall):\quad \frac{\Phi \vdash f : \forall u.A[\![u]\!]}{\Phi \vdash !u.f[u] \Rightarrow f},\ (u \text{ not free in } f),$$

2 Reductio rules.

2.1 Basic type-reduction.

$$(\beta\partial T):\quad \frac{\Phi[\, x : T^{-}\,] \vdash b[\![x]\!] : \bot}{\Phi \vdash \partial x{:}T^{-}.b[\![x]\!] \Rightarrow \Omega\ [:\ T]},$$

$$(\beta\partial\bot):\quad \frac{\Phi[\, x : \bot^{-}\,] \vdash b[\![x]\!] : \bot}{\Phi \vdash \partial x{:}\bot^{-}.b[\![x]\!] \Rightarrow b[\![x:=\lambda z{:}\bot.z]\!]\ [:\ \bot]},$$

$$(\beta\partial\supset):\quad \frac{\Phi_1 \vdash a : A \qquad \Phi_2[\, x : (A\supset B)^{-}\,] \vdash b[\![x]\!] : \bot}{\Phi_1,\Phi_2 \vdash (\partial x{:}(A\supset B)^{-}.b[\![x]\!])a \Rightarrow \partial x{:}B^{-}.b'[\![x]\!]\ [:\ B]},$$

where $b'[\![x]\!] \equiv b[\![x:=\lambda z{:}(A\supset B).x(za)]\!]$.

2.2 First-order type-reduction. If u is not free in b[[x]],

$$(\beta\partial\forall):\quad \frac{\Phi_1 \Vdash t :: U \qquad \Phi_2[\, x : (\forall u.A[\![u]\!])^{-}\,] \vdash b[\![x]\!] : \bot}{\Phi_1,\Phi_2 \vdash (\partial x{:}(\forall u.A[\![u]\!])^{-}.b[\![x]\!])[t] \Rightarrow \partial x{:}A^{-}[\![u:=t]\!].b'[\![x]\!]\ [:\ A']},$$

where $b'[\![x]\!] \equiv b[\![x:=\lambda z{:}(\forall u.A[\![u]\!]).x(z[t])]\!]$
[**and** $A' \equiv A[\![u:=t]\!]$].

2.3 Extensionality.

$$(\eta\supset\partial):\quad \frac{\Phi \vdash f : A}{\Phi \vdash \partial x{:}A^{-}.xf \Rightarrow f},\ (x \text{ not free in } f),$$

3 *Compatibility rules.*

3.1 Basic compatibility.

$$(\mu\supset):\quad \frac{\Phi_1 \vdash f \Rightarrow g : A \supset B \qquad \Phi_2 \vdash a : A}{\Phi_1\Phi_2 \vdash fa \Rightarrow ga\ [: B]},$$

$$(\nu\supset):\quad \frac{\Phi_1 \vdash a \Rightarrow b : A \qquad \Phi_2 \vdash f : A \supset B}{\Phi_1\Phi_2 \vdash fa \Rightarrow fb\ [: B]},$$

$$(\wr\supset\lambda):\quad \frac{\Phi[\ x : A\ ] \vdash a[\![x]\!] \Rightarrow b[\![x]\!] : B}{\Phi \vdash \lambda x{:}A.a[\![x]\!] \Rightarrow \lambda x{:}A.b[\![x]\!] : [A \supset B]},$$

$$(\wr\supset⁀):\quad \frac{\Phi[\ x : A^- \ ] \vdash a[\![x]\!] \Rightarrow b[\![x]\!] : \bot}{\Phi \vdash ⁀x{:}A^-.a[\![x]\!] \Rightarrow ⁀x{:}A^-.b[\![x]\!]\ [: A]}.$$

3.2 First-order compatibility. If u is not free in f, g [$(\mu\forall)$],

$$(\mu\forall):\quad \frac{\Phi_1 \Vdash t :: U \qquad \Phi_2 \vdash f \Rightarrow g : \forall u.A[\![u]\!]}{\Phi_1\Phi_2 \vdash f[t] \Rightarrow g[t]\ [: A[\![u:=t]\!]]},$$

$$(\wr\forall):\quad \frac{\Phi[\ u :: U\ ] \vdash a[\![u]\!] \Rightarrow b[\![u]\!] : A[\![u]\!]}{\Phi \vdash !u.a[\![u]\!] \Rightarrow !u.b[\![u]\!]\ [: \forall u.A]}.$$

5.3 Remark.

It is appropriate to state explicitly the *restrictions on the use of variables appearing in the rules* above.

(1) *Provisoes on contexts.* Concerning the statement of rules that are not proper context rules: the notation agrees here with **Convention $\Phi_{[U]}$** above. That is to say: the assumption on contexts of the form Φ[x : C] (resp. Φ[u :: U]) is that [x : C] (resp. [u :: U]) does not occur in Φ. In detail:

in $(\wr\supset\lambda)$: [x : A] is not in Φ,
in $(\wr\supset⁀)$: [x : A^-] is not in Φ,
in $(\wr\forall)$: [u :: U] is not in Φ,
in $(\beta\supset\lambda)$: [x : A] is not in Φ_1, Φ_2,
in $(\beta\forall)$: [u :: U] is not in Φ_1, Φ_2,
in $(\beta⁀\top)$: [x : $\top^-$] is not in Φ,
in $(\beta⁀\bot)$: [x : $\bot^-$] is not in Φ,
in $(\beta⁀\supset)$: [x : $(A \supset B)^-$] is not in Φ_1, Φ_2,
in $(\beta⁀\forall)$: [x : $(\forall u.A[\![u]\!])^-$] is not in Φ_1, Φ_2.

(2) *Provisoes on proof-terms.* For extensional [η-] rules involving abstractors, one has, as usually in typed λ-calculi:

in (η⊃λ), (η⊃∀): x is not free in f,
in (η∀): u is not free in f.

5.4 Remark.

To our knowledge, the reduction rules for classical first-order logic have *never* been stated correctly in the literature. See, however, the recent work of Glen Helman [1983,1987] for good intuitions into the equational behavior of **CQ**-proofs.

[42]

6. Classical proof-consistency. We use next a simple extension of the ordinary "type-free" λ-calculus in order to give a *direct* proof of *consistency* for λ𝒥!. "Direct" means that no specific properties of reductions (as, e. g., confluence) will be actually used in the consistency proof.

In other words, we show **Cons**[^{r}λ!], for a "fictious" extension ^{r}λ! of the ordinary extensional ("type-free") λ-calculus, called, for typographical convenience, ^{r}λ here (the latter appears also as λβηK or as λ(η) in the literature [Barendregt 1984]).

The "fictious" λ-calculus ^{r}λ! is, in a sense, not "more" than ^{r}λ. This insures, by a simple translation, the *non-triviality of proof-equality* (or "*proof-conversion*") *in first-order classical logic* **CQ**. So, the *proof-consistency* of *classical logic* is obtained *by a pure* ***type-free*** *λ-calculus technique.*

For reference in technical definitions, the *type-free λ-term-syntax* (of ^{r}λ) is given, inductively, by [primitive notation in brackets]:

x_1, y_1, z_1, ...		general-variables,
(fa)	[≡ @fa]	functional applications,
(λx.a)	[≡ λxa]	λ-abstractions,

(where f, a are λ-terms), with the usual conversion rules (β), (η).

In order to obtain the "fictious" extension ^{r}λ!, we add to this a set of *U-variables*, distinct from those in Var_{λ} and term-forms:

(f[t])	[≡ @ft]	**U**-applications,
(!u.a)	[≡ !ua]	**U**-abstractions.

Beyond the familiar rules (β) [≡ (βλ)], (η) [≡ (ηλ)] of ^{r}λ, ^{r}λ! has also some trivial analogues (β!), (η!) of these. For e in an appropriate *index set*, we simulate, in terms of ^{r}λ!, additional term-forms

($\mathcal{J}_e$x.a)	[≡ $\mathcal{J}_e$xa]	$\mathcal{J}$-abstractions.

Finally, ^{r}λ! is shown to be *consistent* as an equational theory and λ𝒥! is interpreted trivially in ^{r}λ!.

Type-free $\lambda\mathfrak{f}!$*-calculi*. We formalize next the *abstract structure* of first-order classical proofs in a *pure* "type-free" λ-calculus setting.

6.1 Definition (*Structure indices*).

Consider an alphabet Σ consisting of the following symbols:

$$0\ 1\ 2\ (\ )\ [\ ]\ ,$$

[That is: Σ contains three *digits* 0, 1, 2, *parentheses*, *brackets*, and *a comma*.]

(1) The set of *structure indices* is a set of words over $\mathbb{N} \cup \Sigma$, generated by the following inductive conditions: for $n \in \mathbb{N}$,

(1°) $(n,0)$, $(n,1)$, $(n,2)$ are structure indices,
(2°) $(n,[e])$ are structure indices, if so is e.

Notation. If $e \equiv (n,o)$ then $\underline{H}e \equiv n$ is the *height of* e and $\underline{G}e \equiv o$ is the *ground of* e.

(2) A structure index e is said to be *basic*, if e does not contain brackets.

6.2 Remark (*Structure indices as fictious types*).

The structure indices reflect the *abstract structure* of certain provability languages. Consider, e. g., the first-order language $L := L[\supset,\forall]$. L is constructed, from atomic propositional constants $\top$, $\bot$, atomic propositions $P[\![u_1,\ldots,u_n]\!]$, by closing under a binary connective $\supset$ ("material implication") and a first-order ("universal") quantifier $\forall$. Then there is a correspondence

$$L\text{-formulas/propositions} \longrightarrow \textbf{indices},$$

since any formula/proposition A in L is of the form:

$A \equiv (A_1 \supset .(A_2 \supset \ldots .(A_n \supset B)\ldots)$ $(n \geq 0)$ [index (n,e)],

where B [index e] is			
either	(1°)	$\top$	[index $(0,0)$]
or	(2°)	$\bot$	[index $(0,1)$]
or	(3°)	a propositional atom P	[index $(0,2)$]
or of the form	(4°)	$\forall u.C$	[index $(n,[e])$],
		with C formula/proposition	[index e].

So, the basic indices would correspond to quantifier-free formulas/propositions of L.

[44]

6.3 Definition (*Type-free λ!-calculus*).

(1) The calculus '**λ!** has, as *syntactic categories*:

(1°) A set **U** of **U**-*terms*. The **U**-terms are ranged over by **t**, possibly with decorations. The set of **U**-terms is left unspecified, except for the fact that **U** is supposed to contain an infinite set $Var_{\mathbf{U}}$ of **U**-*variables*. The **U**-variables are thus **U**-terms.

(2°) A set Term[**λ!**] of λ!-*terms*, ranged over by a, b, c, ..., f, g, h, ... and generated inductively by:

u_i, v_i, ...		**U**-variables,
x_i, y_i, z_i, ...		g-variables [in a set Var_{λ}],
(fa)	[≡ @fa]	functional applications,
(λx.a)	[≡ λxa]	λ-abstractions,
(f[**t**])	[≡ @f**t**]	**U**-applications,
(!u.a)	[≡ !ua]	**U**-abstractions,

where f, a are λ!-terms and **t** is a **U**-term [in the above, the primitive notation appears in brackets].

6.4 Notation.

(1) In notation, economy on parentheses and brackets is made in the usual way (by associating to the left and by eliminating the outermost pair). This gives the shorthand $fa_1 \ldots a_n$ for $(\ldots(fa_1)\ldots a_n)$, as usual, but we can write $f[\mathbf{t}_1,\ldots,\mathbf{t}_n]$ as shorthand for $((\ldots(f[\mathbf{t}_1])[\mathbf{t}_2])\ldots[\mathbf{t}_n])$

(2) Following the customary practice, we write $\lambda x_1 \ldots x_n.a$ for $\lambda x_1 \ldots \lambda x_n.a$ and, analogously, $!u_1 \ldots u_n.a$ for $!u_1 \ldots !u_n.a$.

(3) For λ!-terms a, c, and **U**-terms **t**, we write also $a[\![x:=c]\!]$ and $a[\![u:=\mathbf{t}]\!]$, resp. for *the substitution operators* on λ!-terms. [These stand for formal operators that can be given a rigorous inductive definition, which we want to skip.]

(4) For convenience, "⊢ ..." stands for derivability in '**λ!**.

Subterms of a λ!-term are supposed to be introduced in the usual inductive way (such as to enable us to do induction on subterms, for instance). By similar inductions, it is assumed that it is clear what means to be *bound* and *free* for the occurrences of a g-variable resp. a **U**-variable in a λ!-term. $FV_{\mathbf{g}}(a)$ and $BV_{\mathbf{g}}(a)$, (resp. $FV_{\mathbf{U}}(a)$ and $BV_{\mathbf{U}}(a)$), stand for the set of bound and free g-variables (resp,. **U**-variables) of a term a. A λ!-term is *g-* resp. **U**-*closed* if it does not contain free g- resp. **U**-variables.

A *closed* λ!-*term* (or a λ!-*combinator*) is a λ!-term a without free variables (it is both g- and **U**-closed: $FV_{\mathbf{g}}(a) = FV_{\mathbf{U}}(A) = \emptyset$).

The "window brackets" ⟦...⟧ are part of the the meta-language and are used to display occurrences of free g- and/or **U**-variables in a λ!-term, or formal operations applied to such occurrences. As ever, syntactic identity is denoted by ≡.

It is also assumed that the reader knows how to do systematic re-letterings of bound g- and **U**-variables, i. e. that he is able to identify/distinguish terms by *α-conversion*, i. e., by using the (renaming) rules (for g- and **U**-variables):

(α_λ): $\lambda x.a⟦x⟧ \equiv_\alpha \lambda y.a⟦x:=y⟧$, (y fresh for a),
(α_U): $!u.a⟦u⟧ \equiv_\alpha !v.a⟦u:=v⟧$, (v fresh for a).

In the sequel, = stands for *conversion* (the "extensional equality") in $^r\lambda!$. It is assumed to respect α-conversion, in the usual way ($\equiv_\alpha \subseteq =$, as set-inclusion between relations).

6.5 Definition ($^r\lambda!$-*conversion rules*).

Conversion (or *equality*) = in $^r\lambda!$ is defined, as usually, by *conversion rules*:

$(\beta\lambda=)$: $(\lambda x.a)c = a⟦x:=c⟧$,
$(\eta\lambda=)$: $\lambda x.fx = f$, (x not free in f),
$(\beta!=)$: $(!u.a)[t] = a⟦u:=t⟧$,
$(\eta!=)$: $!u.f[u] = f$, (u not free in f),

6.6 Remark.

(1) Here, the λ!-terms occurring on the **LHS** of a conversion rule are said to be *détours* (or *redexes*); the corresponding **RHS** counterpart make up their *contracta*. *Détour-classification*:

β-détours ("intensional"):	η-détours ("extensional"):
$(\lambda x.a)c$	$\lambda x.fx$ [x not free in f]
$(!u.a)[t]$	$!u.f[u]$ [u not free in f]

A λ!-term is *normal* (or *in normal form*) if no subterm of it is a détour.

(2) As expected, the equality of $^r\lambda!$ is assumed to respect the primitive syntactic operations of the calculus. This gives several *compatibility conditions*, making = into a *congruence* on Term[$^r\lambda!$]. Formally, one must have also, for all λ!-terms f, g, a, b, all **U**-terms t:

(ρ): $a = a$,
(σ): $a = b \Longrightarrow b = a$,
(τ): $a = b,\ b = c \Longrightarrow a = c$,

(μ): $a = b \Longrightarrow fa = fb$,
(υ): $f = g \Longrightarrow fa = ga$,
$(\upsilon!)$: $f = g \Longrightarrow f[t] = f[t]$,
$(\xi\lambda)$: $a = b \Longrightarrow \lambda x.a = \lambda x.b$,
$(\xi!)$: $a = b \Longrightarrow !u.a = !u.b$.

(3) If necessary, the *standard reduction* relation $\Rightarrow$ of $^r\lambda!$ is defined by the same conditions as above, without (σ). One can easily see that:

$(\Rightarrow,=)$: $a \Rightarrow b \Longrightarrow a = b$,
$(\mu\upsilon)$: $f \Rightarrow g,\ a \Rightarrow b \Longrightarrow fa \Rightarrow gb$.

6.7 Lemma.

Cons$[^r\lambda!]$.

Proof. [If, for some reason, the "purity of methods" is at premium, this can be shown by using a *confluence* (Church-Rosser) *argument*, noting that the *standard Tait/Martin-Löf "residuation" technique* [Rezus 1981] applies directly to the case of concern. The required analysis is only slightly more involved than that appearing in, e. g., [Takahashi 1989]. We prefer the following, much shorter, *translation argument*.]

Where Term$[^r\lambda]$ is the set of type-free λ-terms of the ordinary λ-calculus $^r\lambda$, assume that the **U**-terms of $^r\lambda!$ are constructed from **U**-variables and function symbols in a set $\{ f_{[n],i} : i \in I \}$.

Fix a *valuation* $\#_o$ of Var_U into Var_λ, associating to every n-ary function symbol $f_{[n],i}$ of $^r\lambda!$ an arbitrary closed λ-term

$$f_{n,i} := \lambda x_1 \ldots x_n . a_{[n],i}[\![x_1, \ldots, x_n]\!].$$

Define a map $^* : \mathrm{Term}[^r\lambda!] \dashrightarrow \mathrm{Term}[^r\lambda]$, by

$(u)^* = \#_o(u) \in \mathrm{Var}_\lambda$,
$(f_{[n],i}[t_1 \ldots t_n])^* = a_{[n],i}[\![x_1 := (t_1)^*, \ldots, x_n := (t_n)^*]\!]$,
$(x)^* = x \in \mathrm{Var}_\lambda$,
$(ab)^* = (a)^*(b)^*$,
$(\lambda x.a)^* = \lambda x.(a)^*$,
$(a[t])^* = (a)^*(t)^*$,
$(!u.a)^* = \lambda x.(a[\![\#_o(u) := x]\!])^*$.

Then

$(a)^* \equiv a$, for all $a \in \mathrm{Term}[^r\lambda]$,

and for all $a, b \in \mathrm{Term}[^r\lambda!]$,

$^r\lambda \vdash (ab)^* = (a)^*(b)^*$,
$^r\lambda \vdash (\lambda x.a)^* = \lambda x.(a)^*$.

It is equally easy to check the fact that

$$\text{'}\lambda! \vdash a = b \Longrightarrow \text{'}\lambda \vdash (a)^* = (b)^*,$$

for all $\lambda!$-terms a, b, whence **Cons**['$\lambda!$]. □

Type-free ***reductio****-functionals in* '$\lambda!$. We introduce now a sequence of $\lambda!$-combinators, used next in order to simulate the equational behavior of $\lambda\beta!$ within '$\lambda!$.

6.8 Remark.

(1) (*"λ-pure" combinators.*) The following combinators are standard:

$$\begin{array}{ll} \mathbf{I} := \lambda x.x, & \\ \mathbf{K} := \lambda x.\lambda y.x, & \mathbf{K'} := \lambda x.\lambda y.y, \\ \mathbf{B} := \lambda x.\lambda y.\lambda z.x(yz), & \mathbf{B'} := \lambda x.\lambda y.\lambda z.y(xz), \\ \mathbf{C} := \lambda x.\lambda y.\lambda z.xzy, & \mathbf{C}_* := \lambda x.\lambda y.yx, \\ \mathbf{S} := \lambda x.\lambda y.\lambda z.xz(yz). & \end{array}$$

(2) ($\lambda!$-*combinators.*) There are obvious $\lambda!$-analogues of the above.

$$\begin{array}{ll} \mathbf{K}_! := \lambda x.!u.x, & \mathbf{K'}_! := !u.\lambda x.x, \\ \mathbf{B}_! := \lambda x.\lambda y.!u.x(y[u]), & \mathbf{B'}_! := \lambda x.\lambda y.!u.y(x[u]), \\ \mathbf{C}_! := \lambda x.\lambda y.!u.x[u]y, & \mathbf{\Theta}_! := \lambda x.!u.\lambda y.(xy)[u], \\ \mathbf{\Phi}_! := \lambda x.!u.!v.x[v][u], & \nabla_! := !u.\lambda x.x[u], \\ \mathbf{S}_! := \lambda x.\lambda y.!u.x[u](y[u]), & [\mathbf{t}] := \nabla_![t], \text{ for any } \mathbf{U}\text{-term } t. \end{array}$$

So **U**-application f[**t**] can be expressed by [**t**](f).

6.9 Notation.

$$\begin{array}{ll} a \circ b := \lambda x.a(bx) & [\text{for } x \text{ not free in } a,\ b]. \\ \langle a_1,\ldots,a_n\rangle := \lambda x.xa_1\ldots a_n & [\text{for } x \text{ not free in } a_1,\ldots,a_n]. \\ \nabla := \lambda xy.yx \ [\ \equiv \lambda x.\langle x\rangle \equiv \mathbf{C}_*\ ] & \end{array}$$

6.10 Remark.

(1) For x not free in a, b, $\vdash a \circ b = \mathbf{B}ab$.
(2) $\vdash \nabla = \mathbf{CI}$.

6.11 Definition (*The* ***next*** *"type-free" combinators*).

$$\begin{array}{ll} \$_{\langle\rangle} & := \lambda h.\lambda f.\lambda x.h(\lambda y.f(\lambda z.y(zx))), \\ \$_{[]} & := \lambda h.\lambda f.!u.h(\lambda y.f(\lambda z.y(z[u]))). \end{array}$$

6.12 Remark.

One has $\vdash \$_{\langle\rangle} = \lambda hfx.h(\lambda y.f(y\circ(\nabla x)))$ and, by expansion, also $\vdash \$_{\langle\rangle} = \mathbf{C}[\mathbf{B}\circ\mathbf{B}](\mathbf{C}[\mathbf{B}\circ\mathbf{B}][(\mathbf{CB})\circ\nabla])$. [*Exercise*: Find expansions for the other **next**-combinator $\$_{[]}$, in terms of **B**, **C**, **I** and basic $\lambda!$-combinators. Note that no **S** and **K** are needed. In other words, the **next**-combinators are "strictly linear" or "**BCI**-linear".]

6.13 Definition.

$$\begin{array}{lll}
\Delta_{\langle o,o\rangle} & := \lambda xy.y & [\equiv_\alpha \lambda x.I\ ],\\
\Delta_{\langle o,1\rangle} & := \lambda x.x(\lambda y.y) & [\equiv_\alpha \lambda x.xI \equiv \langle I\rangle\ ],\\
\Delta_{\langle o,2\rangle} & := \lambda xy.x(\lambda z.zy) & [\equiv_\alpha \lambda xy.x\langle y\rangle\ ],\\
\Delta_{\langle n+1,\mathbf{e}\rangle} & := \$_{\langle\rangle}\Delta_{\langle n,\mathbf{e}\rangle} & [\text{for } n \geq 0 \text{ and any structure index } \mathbf{e}],\\
\Delta_{\langle o,\lceil\mathbf{e}\rceil\rangle} & := \$_{\lceil\rceil}\Delta_{\mathbf{e}} & [\text{for any structure index } \mathbf{e}].
\end{array}$$

6.14 Lemma.

$$\begin{array}{lll}
(0) & \vdash \Delta_{\langle o,o\rangle} = \mathbf{K}' & [=\mathbf{KI}\ ],\\
(1) & \vdash \Delta_{\langle o,1\rangle} = \mathbf{CII} & [=\nabla\mathbf{I}\ ],\\
(2) & \vdash \Delta_{\langle o,2\rangle} = \mathbf{CB(CI)} & [=\mathbf{B}'\nabla\ ].
\end{array}$$

Proof. Trivial. □

6.15 Lemma.

For any structure index $\mathbf{e}$, and all $\lambda(!)$-terms f, a,

$$\begin{array}{ll}
(1) & \vdash \Delta_{\langle n+1,\mathbf{e}\rangle} = \lambda fx.\Delta_{\langle n,\mathbf{e}\rangle}(\lambda z.f(\lambda u.z(ux))) =\\
& \qquad = \lambda fx.\Delta_{\langle n,\mathbf{e}\rangle}(\lambda z.f(z\circ\nabla x)) \quad =\\
& \qquad = (\mathbf{B}\Delta_{\langle n,\mathbf{e}\rangle})\circ(\mathbf{C}[\mathbf{B}\circ\mathbf{B}][\mathbf{CB}\circ\nabla]),\\
(2) & \vdash \Delta_{\langle n+1,\mathbf{e}\rangle}(f) = \Delta_{\langle n,\mathbf{e}\rangle}\circ(\mathbf{B}f\circ[\mathbf{CB}\circ\nabla]),\\
(3) & \vdash \Delta_{\langle n+1,\mathbf{e}\rangle}(\lambda x.a[\![x]\!]) = \Delta_{\langle n,\mathbf{e}\rangle}\circ(\lambda xy.a[\![x:=\lambda z.y(zx)]\!]) \quad =\\
& \qquad = \Delta_{\langle n,\mathbf{e}\rangle}\circ(\lambda xy.a[\![x:=(y)\circ(\nabla x)]\!]) \quad =\\
& \qquad = \lambda x_1.\Delta_{\langle n,\mathbf{e}\rangle}(\lambda x_2.a[\![x:=\lambda y.x_2(yx_1)]\!]) =\\
& \qquad = \lambda x_1.\Delta_{\langle n,\mathbf{e}\rangle}(\lambda x_2.a[\![x:=(x_2)\circ(\nabla x_1)]\!]).
\end{array}$$

Proof. Straightforward, from definitions. □

6.16 Lemma.

For any structure index $\mathbf{e}$, and all $\lambda!$-terms f, a,

$$\begin{array}{ll}
(1) & \vdash \Delta_{\langle o,\lceil\mathbf{e}\rceil\rangle} = \lambda f.!u.\Delta_{\mathbf{e}}(\lambda y.f(\lambda z.y(z[u]))),\\
(2) & \vdash \Delta_{\langle o,\lceil\mathbf{e}\rceil\rangle}(f) = !u.\Delta_{\mathbf{e}}(\lambda y.f(\lambda z.y(z[u]))),\\
(3) & \vdash \Delta_{\langle o,\lceil\mathbf{e}\rceil\rangle}(\lambda x.a[\![x]\!]) = !u.\Delta_{\mathbf{e}}(\lambda y.a[\![x:=\lambda z.y(z[u])]\!]).
\end{array}$$

Proof. From definitions. □

6.17 Corollary.

For any structure index $\mathbf{e}$, all $n \in \mathbb{N}$, and all $\lambda!$-terms a,

$$\begin{array}{ll}
(1) & \vdash \Delta_{\langle n,\mathbf{e}\rangle}(\lambda x.a[\![x]\!])\\
& \quad = \lambda x_1.\ldots.\lambda x_n.\Delta_{\langle o,\mathbf{e}\rangle}(\lambda y.a[\![x:=\lambda z.y(zx_1\ldots x_n)]\!]) =\\
& \quad = \lambda x_1.\ldots.\lambda x_n.\Delta_{\langle o,\mathbf{e}\rangle}(\lambda y.a[\![x:=(y)\circ(\nabla x_n)\circ\ldots\circ(\nabla x_1)]\!]),\\
(2) & \vdash \Delta_{\langle n,o\rangle}(\lambda x.a[\![x]\!]) = \lambda x_1.\ldots.\lambda x_n.\lambda y.y,\\
(3) & \vdash \Delta_{\langle n,1\rangle}(\lambda x.a[\![x]\!]) = \lambda x_1.\ldots.\lambda x_n.a[\![x:=\lambda z.zx_1\ldots x_n]\!] =\\
& \qquad = \lambda x_1.\ldots.\lambda x_n.a[\![x:=\langle x_1,\ldots,x_n\rangle]\!] =\\
& \qquad = \lambda x_1.\ldots.\lambda x_n.a[\![x:=(\nabla x_n)\circ\ldots\circ(\nabla x_1)]\!],
\end{array}$$

(4) $\vdash \Delta_{\langle n,2\rangle}(\lambda x.a[\![x]\!]) = \lambda x_1.\ldots.\lambda x_n.\lambda y.a[\![x:=\lambda z.zx_1\ldots x_n y]\!] =$
$= \lambda x_1.\ldots.\lambda x_n.\lambda y.a[\![x:=\langle x_1,\ldots,x_n,y\rangle]\!] =$
$= \lambda x_1.\ldots.\lambda x_n.\lambda y.a[\![x:=(\nabla y)\circ(\nabla x_n)\circ\ldots\circ(\nabla x_1)]\!].$

Proof. (1) By induction on n, using a previous Lemma. (2)-(4) From (1) and a preceeding Lemma. □

6.18 Corollary.

For all $n \in \mathbb{N}$, and all $\lambda!$-terms a,

$\vdash \Delta_{\langle n+1,1\rangle}(\lambda x.a[\![x]\!]) = \Delta_{\langle n,2\rangle}(\lambda x.a[\![x]\!])$.

Proof. Compare **6.17** (3) and (4). □

6.19 Lemma.

$\vdash B\Delta_e\nabla = \Delta_e \circ \nabla = I$ [for any structure index e].

Proof. By induction on structure indices. □

6.20 Definition.

$\eth_e x.a[\![x]\!] := \Delta_e(\lambda x.a[\![x]\!])$, for any structure index e.

6.21 Theorem.

For any structure index e, all $n \in \mathbb{N}$, all $\lambda!$-terms f, a, c and U-terms t,

$(\beta\eth_{\langle 0,0\rangle})$: $\vdash \eth_{\langle 0,0\rangle}x.a[\![x]\!] = \lambda x.x$,
$(\beta\eth_{\langle 0,1\rangle})$: $\vdash \eth_{\langle 0,1\rangle}x.a[\![x]\!] = a[\![x:=\lambda x.x]\!]$,
$(\beta\eth_{\langle n+1,\underline{e}\rangle})$: $\vdash (\eth_{\langle n+1,\underline{e}\rangle}x.a[\![x]\!])c = \eth_{\langle n,\underline{e}\rangle}x.a[\![x:=\lambda z.x(zc)]\!]$,
$(\beta\eth_{\langle 0,\tau\rangle})$: $\vdash (\eth_{\langle 0,\tau\rangle}x.a[\![x]\!])[t] = \eth_e x.a[\![x:=\lambda z.x(z[t])]\!]$,

$(\eta\eth_e)$: $\vdash \eth_e x.xf = f$, [x not free in f].

Proof. Using preceeding Lemmas, one has, in $^t\lambda!$, for any structure index e, and all $n \in \mathbb{N}$,

$(\beta\eth_{\langle 0,0\rangle})$:
$\vdash \eth_{\langle 0,0\rangle}x.a[\![x]\!] = \Delta_{\langle 0,0\rangle}(\lambda x.a[\![x]\!]) = \mathbf{KI}(\lambda x.a[\![x]\!]) = I$.

$(\beta\eth_{\langle 0,1\rangle})$:
$\vdash \eth_{\langle 0,1\rangle}x.a[\![x]\!] = \Delta_{\langle 0,1\rangle}(\lambda x.a[\![x]\!]) = (\lambda x.xI)(\lambda x.a[\![x]\!]) = a[\![x:=\lambda x.x]\!]$.

$(\beta\eth_{\langle n+1,\underline{e}\rangle})$:
$\vdash (\eth_{\langle n+1,\underline{e}\rangle}x.a[\![x]\!])c = \Delta_{\langle n+1,\underline{e}\rangle}(\lambda x.a[\![x]\!])c =$
$= (\lambda x_1.\Delta_{\langle n,\underline{e}\rangle}(\lambda x_2.a[\![x:=\lambda y.x_2(yx_1)]\!]))c = \Delta_{\langle n,\underline{e}\rangle}(\lambda x_2.a[\![x:=\lambda y.x_2(yc)]\!]) =$
$= \eth_{\langle n,\underline{e}\rangle}x.a[\![x:=\lambda z.x(zc)]\!]$.

$(\beta\,\mathscr{J}_{\langle o,\tau\rightarrow\sigma\rangle})$:
$\vdash (\mathscr{J}_{\langle o,\tau\rightarrow\sigma\rangle}x.a[\![x]\!])[t] = \Delta_{\langle o,\tau\rightarrow\sigma\rangle}(\lambda x.a[\![x]\!])[t] =$
$= (!u.\Delta_{\sigma}(\lambda y.a[\![x:=\lambda z.y(z[u]\,]\!])\,)[t] = \Delta_{\sigma}(\lambda y.a[\![x:=\lambda z.y(z[t]\,]\!]) =$
$= \mathscr{J}_{\sigma}x.b[\![x:=\lambda z.x(z[t]\,)\!]$.

$(\eta\,\mathscr{J}_{\sigma})$: if x is not free in f then
$\vdash \mathscr{J}_{\sigma}x.xf = \Delta_{\sigma}(\lambda x.xf) = \Delta_{\sigma}(\nabla f) = (\Delta_{\sigma}\circ\nabla)(f) = If = f.$ □

6.22 Remark.

In the end, the "type-free" calculi $^{r}\lambda!$ have also a certain *heuristic* value, since they evidentiate in a perspicuous way the "free structure" of first-order (classical) proofs. Indeed, in practice, as, e. g., in the design of a "soft" proof-assistant for first-order classical logic, it is only this structure that should be taken into account for implementation purposes. [Proper "type-checking" can be always implemented as a distinct "module", so to speak.]

Classical proof-conversion is not trivial. In view of **6.7** and **6.21**, we have immediately the following consistency result.

6.23 Theorem (*Consistency of classical first-order proof-calculi*).

Cons$[\lambda\mathscr{J}!]$.

Proof. Any type/proposition A in $\mathcal{L}[\supset,\forall]$ is of the form:

$$A \equiv [A_1 \supset .A_2 \supset .A_3 \supset \ldots .A_n \supset B],$$

where $n \geq 0$ and either $B \equiv \top$ or $B \equiv \bot$ or $B \equiv P$, for some atomic type/proposition P, or $B \equiv \forall u.C[\![u]\!]$. Define then **idx**(A), *the index of* A, inductively, by

(1°) if $B \equiv \top$ then **idx**(A) = **(n,0)**,
(2°) if $B \equiv \bot$ then **idx**(A) = **(n,1)**,
(3°) if $B \equiv P$ then **idx**(A) = **(n,2)**, for any atomic type P,
(4°) if $B \equiv \forall u.C$ and **idx**(C) = **e** then **idx**(A) = **(n,[e])**.

The indices of types/propositions in $\mathcal{L}[\supset,\forall]$ are, isomorphically, the structure indices of **Definition 6.1**. Choose now $^{r}\lambda!$ to be such that

- $^{r}\lambda!$ has the same set of $\mathcal{U}$-variables as $\lambda\mathscr{J}!$ and
- the $\mathcal{U}$-terms of $^{r}\lambda!$ are the $\mathcal{U}$-terms of $\lambda\mathscr{J}!$.

Let Term$[\lambda\mathscr{J}!]$ be the set of p-terms such that $\Phi \vdash a$, for some Φ and define a map $(...)^{F}$: Term$[\lambda\mathscr{J}!]$ --> Term$[^{r}\lambda!]$, by setting first $(t)^{F} \equiv t$ [whence also $(v)^{F} \equiv v$, for $v \in \mathrm{Var}_{\mathcal{U}}$], and by:

$$
\begin{aligned}
(x)^{\mathsf{F}} &= x,\\
(ab)^{\mathsf{F}} &= (a)^{\mathsf{F}}(b)^{\mathsf{F}},\\
(\lambda x{:}A.a)^{\mathsf{F}} &= \lambda x.(a)^{\mathsf{F}},\\
(\gamma x{:}A^{-}.a)^{\mathsf{F}} &= \gamma_{\mathbf{idx}(A)}x.(a)^{\mathsf{F}},\\
(!u.a)^{\mathsf{F}} &= !u.(a)^{\mathsf{F}},\\
(a[t])^{\mathsf{F}} &= (a)^{\mathsf{F}}[t],
\end{aligned}
$$

where $\gamma_{-}x.a$, $\mathbf{e} \equiv \mathbf{idx}(A)$, is as earlier.

The map $(\ldots)^{\mathsf{F}}$ erases the type-structure of $\lambda\gamma!$, preserving - where applicable - the indices of types/propositions as structure-indices on γ-abstractors in type-free terms of the form $\gamma_{-}x.a$.

Then we have $\lambda\gamma! \vdash a = b \Longrightarrow {}^{\mathsf{f}}\lambda! \vdash (a)^{\mathsf{F}} = (b)^{\mathsf{F}}$, by induction on the length of derivation of $\lambda\gamma! \vdash a = b$, using **6.21**, and we had $\mathbf{Cons}[{}^{\mathsf{f}}\lambda!]$ in **6.7**. So $\mathbf{Cons}[\lambda\gamma!]$. □

6.24 Remark.

In particular, $\mathbf{Cons}[\lambda\gamma!]$ yields $\mathbf{Cons}[\lambda\gamma]$, *proof-consistency for classical propositional logic* **C**, as formulated in the "minimal" provability language $L[\supset]$.

6.25 Exercises ("*ϑ-diagonalization*", "***abort***" *and Clavian functionals*).

(1) Show that the "*reductio*-functionals" ϑ_σ satisfy the following "diagonalization" property, for any structure index e:

$(\phi\vartheta_\sigma)$: $\vdash \vartheta_\sigma x.f[\![x]\!](x(\vartheta_\sigma y.c[\![x,y]\!])) = \vartheta_\sigma z.f[\![z]\!](c[\![x:=z]\!][\![y:=z]\!])$,

whence also a "weak diagonalization":

$(\phi_o\vartheta_\sigma)$: $\vdash \vartheta_\sigma x.x(\vartheta_\sigma y.c[\![x,y]\!]) = \vartheta_\sigma z.c[\![x:=z]\!][\![y:=z]\!]$, and

$(\kappa\vartheta_\sigma)$: $\vdash \vartheta_\sigma x.x(\vartheta_\sigma y.c[\![x]\!]) = \vartheta_\sigma x.c[\![x]\!]$, (y not free in $c[\![x]\!]$),
$(\kappa\varepsilon_\sigma)$: $\vdash \vartheta_\sigma x.x(\vartheta_\sigma y.x(a[\![x]\!])) = \vartheta_\sigma x.x(a[\![x]\!])$ (y not free in $a[\![x]\!]$),
$(\kappa\omega_\sigma)$: $\vdash \vartheta_\sigma x.x(\vartheta_\sigma y.f) = \vartheta_\sigma x.f$, (x, y not free in f).

(2) Define for any structure index e [where x is not free in f],

$\omega_\sigma(f) := \vartheta_\sigma x.f$, $\varepsilon_\sigma x.a[\![x]\!] := \vartheta_\sigma x.x(a[\![x]\!])$ and

$\vartheta^\sigma x.c[\![x]\!] := \varepsilon_\sigma x.\omega_\sigma(c[\![x]\!])$,
$\omega^\sigma(f) := \vartheta^\sigma x.f$, $\varepsilon^\sigma x.a[\![x]\!] := \vartheta^\sigma x.x(a[\![x]\!])$, etc.

[NB. The ε-families consist of "Clavian functionals", called so after the Jesuit mathematician Christoph Clavius (1538-1612). The ω-families contain "**abort**-cancellators" that should be familiar from the proof-calculus of Heyting's [1930] logic.]

Show that one has, successively, if x is not free in f,

$(\kappa\vartheta_\sigma)$: $\vdash \vartheta^\sigma x.c[\![x]\!] = \varepsilon_\sigma x.\omega_\sigma(c[\![x]\!]) = \vartheta_\sigma x.c[\![x]\!]$,
$(\kappa\varepsilon_\sigma)$: $\vdash \varepsilon^\sigma x.a[\![x]\!] = \vartheta^\sigma x.x(a[\![x]\!]) = \varepsilon_\sigma x.\omega_\sigma(x(a[\![x]\!])) = \varepsilon_\sigma x.a[\![x]\!]$,
$(\kappa\omega_\sigma)$: $\vdash \omega^\sigma(f) = \vartheta^\sigma x.f = \varepsilon_\sigma x.\omega_\sigma(f) = \vartheta_\sigma x.f = \omega_\sigma(f)$.

(3) Note that $(\phi_o\vartheta_\sigma)$ yields

$(\phi_o\varepsilon_\sigma)$: $\vdash \varepsilon_\sigma x.\varepsilon_\sigma y.a[\![x,y]\!]) = \varepsilon_\sigma z.a[\![x:=z]\!][\![y:=z]\!]$, etc.

Check $(\phi\varepsilon_\sigma)$ directly (in "type-free" λ-calculus). Conclude that the (ω-ε)-family (-pair) is tantamount the ϑ-family.

(4) Express the Δ-family in terms of (ω-ε). What happens if we leave out the ω's? Establish "Δ-diagonalization" properties analogous to $(\phi_o\vartheta)$ and $(\phi_o\varepsilon)$ above.

(5) Derive properties of the ε-family corresponding to those of the ϑ-family from Theorem **6.21**. What about the ω-analogues?

(6) Show that the "Clavian" ε-family alone does *not* suffice in order to express the basic properties of the ϑ- and/or the Δ-families (as listed in **6.21** and in the above).

(7) Axiomatize the equational properties of the ***reductio***-family ($\mathfrak{g}$). Formulate an appropriate notion of *completeness* and establish *completeness results* for such axiomatizations.

(8) Same problem for the ε-family and for the ε-ω-family (-pair).

(9) Formulate [stratified] $\lambda\mathfrak{g}(!)$-*theories* [= Post consistent extensions of $\lambda\mathfrak{g}(!)$] satisfying typed analogues of $(\kappa\mathfrak{g})$ and the "$\mathfrak{g}$-diagonalization" properties $(\phi\mathfrak{g})$, $(\phi_o\mathfrak{g})$. [*Hint*. Define first an extension $\lambda\mathfrak{g}\phi!$ of $\lambda\mathfrak{g}!$ by a (reduction) rule $[\phi\mathfrak{g}]$:

$$\frac{\Phi[x{:}A^-] \vdash f[\![x]\!] : t\ [\equiv \bot \supset \bot],\quad \Phi[x{:}A^-][y{:}A^-] \vdash c[\![x,y]\!] : \bot}{\Phi \vdash \mathfrak{g}x{:}A^-.f[\![x]\!](x(\mathfrak{g}y{:}A^-.c[\![x,y]\!])) \Rightarrow \mathfrak{g}z{:}A^-.f[\![z]\!](c[\![z,z]\!]) : A},$$

where $c[\![z,z]\!] \equiv c[\![x:=z]\!][\![y:=z]\!]$, using, *mutatis mutandis*, the above in order to show **Cons**$[\lambda\mathfrak{g}\phi!]$. The stratified analogues of $(\kappa\mathfrak{g})$, $(\phi_o\mathfrak{g})$ are then derivable in $\lambda\mathfrak{g}\phi!$.]

Show that there are stratified analogues of ω and ε above allowing alternative [equationally equivalent] formulations for $\lambda\mathfrak{g}(!)$ and the $\lambda\mathfrak{g}(!)$-theories obtained previously.

(10) Isolate the provability system ("logic") corresponding to proof-languages $\lambda\varepsilon(!)$, say, based only on stratified λ ["abstraction"/"deduction"], **@** ["detachment"/*modus ponens*] and ε [*consequentia mirabilis*/"**escape**"] and, possibly, the $\forall$-primitives. [*Hint*. The stratified "abstraction" ε must go into a rule. For each type A, its corresponding "closure" $\lambda f{:}(A^-\supset A).\varepsilon x{:}A^-.fx$ is a proof-combinator **E**$[\![A]\!]$, that "proves" all instances of *mirabilis consequentia* (the so-called "Law of Clavius") $A^- \supset A \supset A$.] Discuss the resulting "proof-theories", formulated with and/or without an analogue of the "ε-diagonalization" property.

(11) In connection with (10), see also Curry's "logic of complete refutability" **LD** (in, e.g., **JSL 17**, 1952, pp. 35-42 or [Curry 1963]). Compare $\lambda\varepsilon!$ (based on a provability language with primitives $\top$, $\bot$, $\supset$ and $\forall$ alone), as obtained *sub* (10), with the *intended* concept of a **LD**-*proof*. Formalize the *full* first-order **LD**-system in $\lambda\varepsilon!$-style. [*Hints*. Use the $\lambda\mathfrak{g}(\phi)!$-pattern, replacing the ***reductio*** abstractor $\mathfrak{g}$ by the stratified/typed "Clavian" ε-abstractor from (10). One should also realize that Curry's **LD** has $[\vee,\exists]$-proof operations that are more general than the corresponding *Minimalkalkül*/Heyting analogues: the former admits also of "Clavian" hypotheses $\Phi[x{:}A^-] \vdash a[\![x]\!] : A$. Cf. **13.12** below. **NB** The *full* **LD** requires also "commuting" $\vee$- and $\exists$-rules that are slightly more general than those of the Heyting proof-calculus. For provability matters alone, see also J. P. Seldin in **Studia Logica XLVIII** (2), 1989, pp. 193-217.]

7. Equational proof-behaviors in λ⅁!. We review several specific equational properties of the first-order classical proof-operations.

*The **duplex negatio** functionals and the **next** proof-combinators.* In classical logic, the proofs of $A^{--} \supset A$ [*duplex negatio affirmat*], appear as *specific functionals*. In particular, such functionals can be shown to be "recursively stratified". In fact, the "complex" applications of *duplex negatio* operators $\Delta[\![A]\!]$, for A non-atomic are recursively eliminable, *modulo* proof-conversion, in favor of "atomic" ones ($\Delta[\![B]\!]$, where B is $\top$, $\bot$ or a "prime" type/proposition $P[\![u_1,\ldots,u_n]\!]$) and a few well-behaved ("stricty linear") ordinary functionals (viz. the *stratified* **next** *functionals*, whose type-free structure has been already evidentiated earlier).

> The Δ's are "oracle functionals", so to speak. Intuitively, an "oracle functional" is to be contrasted with functionals usually/oft referred to as "(effective) constructions" in intuitionism (Brouwer and Heyting) and/or in constructive mathematics (Bishop, Bridges, etc.). In its full generality [= for **all** types/propositions A], $\Delta[\![A]\!]$ [$\equiv \lambda x{:}A^{--}.\,⅁y{:}A^{-}.\,xy$], is a case in point.

It is useful to note first that the (primitive) ⅁-abstractors can be eliminated in favor of the family of $\Delta[\![A]\!]$-combinators. That is:

7.1 Theorem (*⅁-elimination*).

> For all types/propositions A, in $L[\supset,\forall]$, any context Φ and all proof-terms $a[\![x]\!]$, such that $\Phi\,[x : A^{-}] \vdash a[\![x]\!] : \bot$,
>
> $$\Phi \vdash \Delta[\![A]\!](\lambda x{:}A^{-}.a[\![x]\!]) \Rightarrow ⅁x{:}A^{-}.a[\![x]\!] : A.$$

Proof. If $\Phi\,[x : A^{-}] \vdash a[\![x]\!] : \bot$ then $\Phi \vdash \lambda x{:}A^{-}.a[\![x]\!] : A^{--}$, whence, by the definition of $\Delta[\![A]\!]$, and $(\beta\supset\lambda)$, one has $\Phi \vdash \Delta[\![A]\!](\lambda x{:}A^{-}.a[\![x]\!]) \equiv_\alpha (\lambda z{:}A^{--}.⅁y{:}A^{-}.zy)(\lambda x{:}A.a[\![x]\!]) \Rightarrow ⅁y{:}A^{-}.(\lambda x{:}A^{-}.a[\![x]\!])y \Rightarrow ⅁y{:}A^{-}.a[\![y]\!] \equiv_\alpha ⅁x{:}A^{-}.a[\![x]\!] : A$. □

Introduced next is a family of "recursors" on the Δ-"oracles" - called ***next** proof-combinators*. These are *stratified* variants of the previous **next** type-free combinators.

In order to ease readability, we extract, as earlier, internal type-declarations of variables "bound" by λ, ⅁ under a local **with**-declaration. For instance, the proof-term $\mathbf{1}[\![A,B]\!] \equiv \lambda x.\lambda y.xy$ **with** $[x : A \supset B][y : A]$ is such that

$$[\ ] \vdash \mathbf{1}[\![A,B]\!]\ [\ \equiv \lambda x{:}(A\supset B).\lambda y{:}A.xy\ ] : A \supset B \supset .A \supset B.$$

7.5 Definition.

For all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$,

$\delta[\![A]\!] := \Delta[\![A]\!]\ [\ \equiv_\alpha \lambda f{:}A^{-}.\,\partial x{:}A^{-}.fx\ ]$, if A is atomic,
$\delta[\![A \supset B]\!] := \$_{\langle\rangle}[\![B,\bot,\bot,B,A]\!](\delta[\![B]\!])$,
$\delta[\![\forall u.A[u]]\!] := \$_{\Gamma\rceil}[\![A[u],\bot,\bot,A[u]]\!](\delta[\![A[u]]\!])$.

7.6 Remark.

A direct inductive argument shows that $\delta[\![A]\!]$ is correctly defined [*exercise*: only type-assignment rules are required].

7.7 Lemma.

For all types/propositions C, in $\mathcal{L}[\supset,\forall]$, $[\ ] \vdash \delta[\![C]\!] : C^{-} \supset C$.

Proof. By induction on the structure of C. □

"Lifting" $\beta\partial$-conversions. In order to show the fact that the "complex" $\Delta[\![A]\!]$'s are recursively eliminable, *modulo* proof-conversion, it is sufficient to show that, for all types/propositions C, one has: $[\ ] \vdash \delta[\![C]\!] = \Delta[\![C]\!] : C^{-} \supset C$. We note first that the $\beta\partial$-rules can be "lifted" by extensionality assumptions, *modulo* proof-conversion.

7.8 Theorem (*"Lifting" $\beta\partial$-conversions*).

For all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$,

1° Basic hyper-$\beta\partial$-conversions.

$$(\vdash\beta\partial\supset):\ \frac{\Phi[\ x : (A\supset B)^{-}\ ] \vdash c[\![x]\!] : \bot}{\Phi \vdash \partial x{:}(A\supset B)^{-}.c[\![x]\!] = \lambda \mathbf{x_0}{:}A.\,\partial \mathbf{x_1}{:}B^{-}.c'[\![\mathbf{x_0},\mathbf{x_1}]\!] : A \supset B},$$

where $c'[\![\mathbf{x_0},\mathbf{x_1}]\!] \equiv c[\![x:=\lambda z{:}(A\supset B).\mathbf{x_1}(z\mathbf{x_0})]\!]$.

2° First-order hyper-$\beta\partial$-conversion. If u is not free in c[x],

$$(\vdash\beta\partial\forall):\ \frac{\Phi[\ x : (\forall u.A[\![u]\!])^{-}\ ] \vdash c[\![x]\!] : \bot}{\Phi \vdash \partial x{:}(\forall u.A[\![u]\!])^{-}.c[\![x]\!] = !u.\,\partial \mathbf{x_1}{:}A^{-}[\![u]\!].c'[\![u,\mathbf{x_1}]\!] : \forall u.A[\![u]\!]},$$

where $c'[\![u,\mathbf{x_1}]\!] \equiv c[\![x:=\lambda z{:}(\forall v.A[\![v]\!]).\mathbf{x_1}(z[u])]\!]$.

Proof. For $(\vdash\beta\partial\supset)$, $(\vdash\beta\partial\forall)$: from the corresponding $\beta\partial$-rules, by extensionality (η-rules) and the appropriate compatibility rules, (here: ξ-rules) [*exercise*]. □

7.9 Corollary $(\vdash\beta\Delta)$.

For all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$,

$(\vdash\beta\Delta\supset)$: $[\ ] \vdash \Delta[\![A \supset B]\!] = \lambda f{:}(A\supset B)^{-}.\lambda x{:}A.\,\partial y{:}B^{-}.f(\lambda z{:}(A\supset B).y(zx)) : (A \supset B)^{-} \supset (A \supset B)$,

7.2 Definition (*The* ***next*** *proof-combinators*).

For all types/propositions A, B, C, D, E, in $L[\supset,\forall]$,

(1) $\$_{<>}[\![A,B,C,D,E]\!] := \lambda h.\lambda f.\lambda x.h(\lambda y.f(\lambda z.y(zx)))$, **with** $[h : A\supset B\supset C\supset D][f : E\supset A\supset B\supset C][x : E][y : A\supset B][z : E\supset A]$.

(2) $\$_{[]}[\![A[\![u]\!],B,C,D[\![u]\!]]\!] := \lambda h.\lambda f.!u.h(\lambda y.f(\lambda z.y(z[u])))$, **with** $[h : A[\![u]\!] \supset B \supset C \supset D[\![u]\!]][f : (\forall u.A \supset B) \supset C][u :: \mathcal{U}]$ $[y : A[\![u]\!] \supset B][z : \forall u.A[\![u]\!]]$.

7.3 Notation (*Most general* **NEXT**-*types*).

For all types/propositions A, B, C, D, E, in $L[\supset,\forall]$,

(1) $\mathbf{NEXT}[\supset][\![A,B,C,D,E]\!] \equiv [A \supset B \supset C \supset D \supset .E \supset A \supset B \supset C \supset (E \supset D)]$.

(2) $\mathbf{NEXT}[\forall][\![A[\![u]\!],B,C,D[\![u]\!]]\!] \equiv$
$\equiv [A[\![u]\!] \supset B \supset C \supset D[\![u]\!] \supset .(\forall u.A[\![u]\!] \supset B) \supset C \supset \forall u.D[\![u]\!]]$.

7.4 Lemma (***Next***- "*correctness*").

For all types/propositions A, B, C, D, E, in $L[\supset,\forall]$,

(1) $[\ ] \vdash \$_{<>}[\![A,B,C,D,E]\!] : \mathbf{NEXT}[\supset][\![A,B,C,D,E]\!]$,
(2) $[\ ] \vdash \$_{[]}[\![A[\![u]\!],B,C,D[\![u]\!]]\!] : \mathbf{NEXT}[\forall][\![A[\![u]\!],B,C,D[\![u]\!]]\!]$,

Proof. Trivial [*exercise*]. (*Hint*: Fitch/de Bruijn-style type-checking). □

Consider the following type-instances of the p-terms $\$_{<>}[\![A,B,C,D,E]\!]$, $\$_{[]}[\![A[\![u]\!],B,C,D[\![u]\!]]\!]$, as well as the corresponding type-instances of $\mathbf{NEXT}[\supset][\![A,B,C,D,E]\!]$, $\mathbf{NEXT}[\forall][\![A[\![u]\!],B,C,D[\![u]\!]]\!]$, resp:

(1) $\$_{<>}[\![B,\bot,\bot,B,A]\!] := \lambda h.\lambda f.\lambda x.h(\lambda y.f(\lambda z.y(zx)))$, **with** $[h : B^{-} \supset B][f : (A \supset B)^{-}][x : A][y : B^{-}][z : A \supset B]$,

$\mathbf{NEXT}[\supset][\![B,\bot,\bot,B,A]\!] \equiv$
$\equiv [B \supset \bot \supset \bot \supset B \supset .A \supset B \supset \bot \supset \bot \supset (A \supset B)] \equiv$
$\equiv [B^{-} \supset B \supset .(A \supset B)^{-} \supset (A \supset B)]$.

(2) $\$_{[]}[\![A[\![u]\!],\bot,\bot,A[\![u]\!]]\!] := \lambda h.\lambda f.!u.h(\lambda y.f(\lambda z.y(z[u])))$, **with** $[h : (A[\![u]\!])^{-} \supset A[\![u]\!]][f : (\forall u.A)^{-}][u :: \mathcal{U}][y : (A[\![u]\!])^{-}][z : \forall u.A]$,

$\mathbf{NEXT}[\forall][\![A[\![u]\!],\bot,\bot,A[\![u]\!]]\!] \equiv$
$\equiv [(A[\![u]\!] \supset \bot \supset \bot \supset A[\![u]\!] \supset .\forall u.A[\![u]\!] \supset \bot \supset \bot \supset \forall u.A[\![u]\!]] \equiv$
$\equiv [(A[\![u]\!])^{-} \supset A[\![u]\!] \supset .(\forall u.A[\![u]\!])^{-} \supset \forall u.A[\![u]\!]]$.

$(\vdash\beta\Delta\forall)$: $[\ ] \vdash \Delta[\![\forall u.A]\!] = \lambda f{:}(\forall u.A)^{=}.!u.\not\!\!\partial y{:}A^{-}.f(\lambda z{:}(\forall u.A).y(z[u])) :$
$: (\forall u.A)^{=} \supset (\forall u.A).$

Proof. From the definition of $\Delta[\![C]\!]$ [$\equiv \lambda x{:}C^{=}.\not\!\!\partial x{:}C^{-}.fx$], by **7.8**, using **7.1**. □

Another way of writing this is, of course:

7.10 Corollary ($\vdash\beta\Delta$: "*complex*" Δ-*eliminations*).

(1) For all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$,

$(\vdash\beta\Delta\supset)$: $[\ ] \vdash \Delta[\![A \supset B]\!] = \lambda f{:}(A\supset B)^{=}.\lambda x{:}A.\Delta[\![B]\!](\lambda y{:}B^{-}.f(\lambda z{:}(A\supset B).y(zx))) :$
$: (A \supset B)^{=} \supset (A \supset B),$

$(\vdash\beta\Delta\forall)$: $[\ ] \vdash \Delta[\![\forall u.A]\!] = \lambda f{:}(\forall u.A)^{=}.!u.\Delta[\![A]\!](\lambda y{:}A^{-}.f(\lambda z{:}(\forall u.A).y(z[u])))$
$: (\forall u.A)^{=} \supset (\forall u.A).$

(2) Analogously, for all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$, any context Φ and all proof-terms f,

$$(\vdash\beta\Delta\supset):\ \frac{\Phi \vdash f : (A\supset B)^{=}}{\Phi \vdash \Delta[\![A\supset B]\!](f) = \lambda x{:}A.\Delta[\![B]\!](\lambda y{:}B^{-}.f(\lambda z{:}(A\supset B).y(zx))) : A\supset B},$$

$$(\vdash\beta\Delta\forall):\ \frac{\Phi \vdash f : (\forall u.A)^{=}}{\Phi \vdash \Delta[\![\forall u.A]\!](f) = !u.\Delta[\![A]\!](\lambda y{:}A^{-}.f(\lambda z{:}(\forall u.A).y(z[u]))) : \forall u.A},$$

Proof. (1) From the definition of $\Delta[\![C]\!]$, by **7.9**. (2) From (1), by β-reduction/conversion. □

7.11 Theorem.

For all types/propositions C, in $\mathcal{L}[\supset,\forall]$,

$$[\ ] \vdash \delta[\![C]\!] = \Delta[\![C]\!] : C^{=} \supset C.$$

Proof. By induction on the structure of C, using $(\beta\not\!\!\partial)$-rules and extensionality; in fact, the $(\vdash\beta\Delta)$-"hyper-rules" above [**7.10**].

(1) If C is atomic, there is nothing to prove.

(2) Else, we must show that, for all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$, (viz., for $C \equiv [A \supset B]$, $\forall u.A$),

(21) $[\ ] \vdash \delta[\![A \supset B]\!] = \Delta[\![A \supset B]\!] : (A \supset B)^{=} \supset (A \supset B)$,
(22) $[\ ] \vdash \delta[\![\forall u.A]\!] = \Delta[\![\forall u.A]\!] : (\forall u.A[\![u]\!])^{=} \supset (\forall u.A[\![u]\!])$.

The details are easy and can be left to the reader. □

7.12 Remark.

The procedure used to eliminate "complex" Δ's" in the above is implicit in the [Prawitz 1965,1971] *N*-style presentation of first-order classical logic (where it is also obliquely referred to as "constructivization").

Extensionality principles and ground proof-combinators. The proof-calculus $\lambda\mathfrak{d}!$ is *extensional* in the expected sense. In other words, all proof-operations involved in $\lambda\mathfrak{d}!$ can be shown to be "extensional", relative to the formalized concept of *proof-equality/conversion*.

7.13 Theorem (*Extensionality principles*).

For all types/propositions A, B in $L[\supset,\forall]$, and proof-terms a, b,

$$\text{(ext }\lambda\text{):}\quad \frac{\Phi[x : A] \vdash ax = bx : B \qquad \Phi \vdash a,\ b : A \supset B}{\Phi \vdash a = b\ [: A \supset B]},\ [x \text{ not free in } a,\ b],$$

$$\text{(ext }\mathfrak{d}\text{):}\quad \frac{\Phi[\ x : A^{-}\ ] \vdash xa = xb : \bot \qquad \Phi \vdash a,\ b : A}{\Phi \vdash a = b\ [: A]},\ [x \text{ not free in } a,\ b],$$

$$\text{(ext }!\text{):}\quad \frac{\Phi[u :: \mathcal{U}] \vdash a[u] = b[u] : A[\![u]\!] \qquad \Phi \vdash a,\ b : \forall u.A[\![u]\!]}{\Phi \vdash a = b\ [: \forall u.A[\![u]\!]\ ]},\ [u \text{ not free in } a,\ b],$$

Proof. (ext λ): Assume $\Phi \vdash a,\ b : A \supset B$, $\Phi[\ x : A\ ] \vdash ax = bx : B$, such that x is not free in a, b. Then, by ($\xi\supset\lambda$), ($\eta\supset\lambda$), etc.,

$$\Phi \vdash a \ll= \lambda x{:}A.ax = \lambda x{:}A.bx \gg b : A \supset B,$$

whence $\Phi \vdash a = b : A \supset B$.

(ext $\mathfrak{d}$): Assume $\Phi \vdash a,\ b : A$, $\Phi[\ x : A^{-}\ ] \vdash xa = xb : \bot$, (x not free in a, b). So, by ($\xi\supset\mathfrak{d}$), ($\eta\supset\mathfrak{d}$), $\Phi \vdash a \ll= \mathfrak{d}x{:}A^{-}.xa = \mathfrak{d}x{:}A^{-}xb \gg b : A$, whence $\Phi \vdash a = b : A$.

(ext !): Assume $\Phi \vdash a,\ b : \forall u.A[\![u]\!]$, $\Phi[\ u :: \mathcal{U}\ ] \vdash a[u] = b[u] : A[\![u]\!]$, such that u is not free in a, b. Then, by ($\xi\forall$), ($\eta\forall$), etc., one has $\Phi \vdash a \ll= !u.a[u] = !u.b[u] \gg b : \forall u.A[\![u]\!]$. So, $\Phi \vdash a = b : \forall u.A[\![u]\!]$. □

In $\lambda\mathfrak{d}!$, the "extensionality" assumptions can be also expressed in terms of Boolean proof-combinators.

7.14 Theorem (*Extensionality principles: combinatory version*).

For all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$,

[**BI**]: $[\ ] \vdash \mathbf{B}[\![B,B,A]\!](\mathbf{I}[\![B]\!]) = \mathbf{I}[\![A\supset B]\!] : A \supset B \supset .A \supset B$,
[$\Delta\nabla$]: $[\ ] \vdash \Delta[\![A]\!] \circ \nabla[\![A]\!] [= \mathbf{B}[\![A^-,A,A]\!](\Delta[\![A]\!])(\nabla[\![A]\!])] = \mathbf{I}[\![A]\!] : A \supset A$,
[$\mathbf{B}_\cup\mathbf{I}$]: $[\ ] \vdash \mathbf{B}_\cup[\![A,A]\!](\mathbf{I}[\![A]\!]) = \mathbf{I}[\![\forall u.A]\!] : \forall u.A \supset \forall u.A$ [u not in $FV_\cup(A)$].

Proof. For [**BI**] one has, by unfolding, $[\ ] \vdash \mathbf{B}[\![B,B,A]\!](\mathbf{I}[\![A]\!]) \equiv_\alpha$ $\equiv_\alpha (\lambda x:(B\supset B).\lambda y:(A\supset B).\lambda z:B.x(yz))(\mathbf{I}[\![B]\!]) \Rightarrow\!\!\!\!\rightarrow \lambda y:(A\supset B).\lambda z:B.\mathbf{I}[\![B]\!](yz) \equiv$ $\equiv \lambda y:(A\supset B).\lambda z:B.(\lambda z:B.z)(yz) \Rightarrow\!\!\!\!\rightarrow \lambda y:(A\supset B).\lambda z:B.yz \Rightarrow\!\!\!\!\rightarrow \lambda y:(A\supset B).y \equiv_\alpha$ $\equiv_\alpha \mathbf{I}[\![A\supset B]\!]$, using $(\eta\supset\lambda)$, as a last reduction step.

[$\Delta\nabla$]: Note first that: $[x : A][y : A^-] \vdash \nabla[\![A]\!]xy \Rightarrow\!\!\!\!\rightarrow yx : A^-$. Then $[\ ] \vdash \Delta[\![A]\!] \circ \nabla[\![A]\!] \equiv \mathbf{B}[\![A^-,A,A]\!](\Delta[\![A]\!])(\nabla[\![A]\!]) \Rightarrow\!\!\!\!\rightarrow \lambda z:A.\Delta[\![A]\!](\nabla[\![A]\!](z)) \equiv$ $\equiv \lambda z:A.(\lambda x:A^-.\eth y:A^-.xy)(\nabla[\![A]\!](z)) \Rightarrow\!\!\!\!\rightarrow \lambda z:A.\eth y:A^-.\nabla[\![A]\!](z)(y) \Rightarrow\!\!\!\!\rightarrow$ $\Rightarrow\!\!\!\!\rightarrow \lambda z:A.\eth y:A^-.yx \Rightarrow\!\!\!\!\rightarrow \lambda z:A.z \equiv_\alpha \mathbf{I}[\![A]\!]$, by $(\eta\supset\eth)$ [last reduction step].

Finally, for [$\mathbf{B}_\cup\mathbf{I}$], if u is not in $FV_\cup(A)$ then $[\ ] \vdash \mathbf{B}_\cup[\![A,A]\!](\mathbf{I}[\![A]\!]) \equiv$ $\equiv (\lambda x:(A\supset A).\lambda y:(\forall u.A).!u.x(y[u]))(\mathbf{I}[\![A]\!]) \Rightarrow\!\!\!\!\rightarrow \lambda y:(\forall u.A).!u.(\mathbf{I}[\![A]\!])(y[u]) \equiv$ $\equiv \lambda y:(\forall u.A).!u.y[u] \Rightarrow\!\!\!\!\rightarrow \lambda y:(\forall u.A).y \equiv_\alpha \mathbf{I}[\![\forall u.A]\!]$, by $(\eta\forall)$ [last step]. □

Δ-rules and ground proof-combinators. In the end, the conversion rules involving $\eth$-abstractors can be also expressed in "closed" form, by means of (ground) proof-combinators only.

7.15 Definition.

For all types/propositions A, B, C, in $\mathcal{L}[\supset,\forall]$,

(1) $\mu[\![A,B,C]\!] := \lambda x:A.\lambda y:(B\supset C).\lambda z:(A\supset B).y(zx)$,
(2) $\mu_\cup[\![A,B]\!] := !u.\lambda y:(A\supset B).\lambda z:(\forall v.A[\![v]\!]).y(z[u])$.

7.16 Lemma.

For all types/propositions A, B, C in $\mathcal{L}[\supset,\forall]$, one has,

(1) $[\ ] \vdash \mu[\![A,B,C]\!] = \mathbf{B}'[\![A\supset B,B,C]\!] \circ \mathbf{C}_*[\![A,B]\!] : A \supset .B\supset C \supset (A\supset B\supset C)$,
[where $\circ$ stands for $\mathbf{B}[\![A\supset B\supset B,(B\supset C)\supset(A\supset B\supset C),A]\!]$],

(2) $[\ ] \vdash \mu_\cup[\![A,B]\!] = \mathbf{B}'[\![\forall u.A,A,B]\!] \circ_\cup \mathbf{C}_{*\cup}[\![A]\!] : \forall u.(A\supset B \supset .(\forall v.A)\supset B)$,
[where $\circ_\cup$ stands for $\mathbf{B}_\cup[\![(\forall u.A)\supset A,(A\supset B)\supset((\forall u.A)\supset B))]\!]$].

7.17 Theorem (*Δ-rules*).

For all types/propositions A, B, in $\mathcal{L}[\supset,\forall]$,

[$\Delta\mathbf{T}$]: $\Phi \vdash f : \mathbf{T}^- \Longrightarrow \Phi \vdash \Delta[\![\mathbf{T}]\!](f) = \Omega : \mathbf{T}$,

[$\Delta\mathbf{\bot}$]: $\Phi \vdash f : \mathbf{\bot}^- \Longrightarrow \Phi \vdash \Delta[\![\mathbf{\bot}]\!](f) = f(\mathbf{I}[\![\mathbf{\bot}]\!]) : \mathbf{\bot}$,

$$[\Delta\supset]:\quad \frac{\Phi \vdash f : (A\supset B)^{-},\ \Phi \vdash a : A}{\Phi \vdash \Delta[\![A\supset B]\!](f)(a) = \Delta[\![B]\!](f \circ \mu[\![A,B,\bot]\!](a)) : B \quad [\ \circ \equiv B[\![(A\supset B)^{-},\bot,B^{-}]\!]\ ]},$$

$$[\Delta\forall]:\quad \frac{\Phi \vdash f : (\forall u.A)^{-},\ \Phi \Vdash t :: U}{\Phi \vdash \Delta[\![\forall u.A]\!](f)[t] = \Delta[\![A]\!](f \circ \mu_{\cup}[\![A,\bot]\!][t]) : A[\![u:=t]\!] \quad [\ \circ \equiv B[\![(\forall u.A[\![u]\!])^{-},\bot,(A[\![u]\!])^{-}]\!]\ ]}.$$

Proof. [$\Delta\mathbf{T}$]: If $\Phi \vdash f : \mathbf{T}^{-}$ then $\Phi \vdash \Delta[\![\mathbf{T}]\!](f) \equiv (\lambda x{:}\mathbf{T}^{-}.\,\gamma y{:}\mathbf{T}^{-}.xy)f = \gamma y{:}\mathbf{T}^{-}.fy = \Omega$, by $(\beta\gamma\mathbf{T})$.

[$\Delta\bot$]: If $\Phi \vdash f : \bot^{-}$ then $\Phi \vdash \Delta[\![\bot]\!](f) = (\lambda x{:}\bot^{-}.\,\gamma y{:}\bot^{-}.xy)f = \gamma y{:}\bot^{-}.fy = (fy)[\![y:=I[\![\bot]\!]]\!] \equiv f(I[\![\bot]\!]) : \bot$, by $(\beta\gamma\bot)$.

[$\Delta\supset$] and [$\Delta\forall$]: From the $\beta\gamma$-rules and the definitions of μ, $\mu_{\cup}$ above. [Alternatively, one could have had $\mu[\![A,B,\bot]\!] \equiv B'[\![A\supset B,B,\bot]\!] \circ C_{*}[\![A,B]\!]$ and $\mu_{\cup}[\![A,\bot]\!][t] \equiv B'[\![\forall u.A[\![u]\!],A[\![u]\!],\bot]\!] \circ_{\cup} C_{*\cup}[\![A[\![u]\!]]\!][t]$, for instance.] □

[59]

8. Boolean Combinatory Logic: the theory of Boolean combinators. In this section we introduce a *proof-combinator theory* **C[CQ]** for *first-order classical logic*.

C[CQ] is an *equational theory* in the stratified language π_{ω}[CQ]. As a ***combinatory logic***, it *codifies the equational behavior of first-order classical proofs*. In fact, it is tantamount $\lambda\beta!$, as we show in this section.

As earlier, **a proposition is a type**. The *type syntax* of **C[CQ]** is thus the *provability syntax* of **CQ**. On the other hand, its *proof-syntax* is, by definition, the same as that of π_{ω}[CQ]

Intuitively, the *statements* of C[CQ] are of the following forms:

- A is a type/proposition [A :: **type**],
- X has type A, [X : A]
- **t** is a $\mathcal{U}$-term [**t** :: $\mathcal{U}$],

as in π_{ω}[CQ], with, moreover, statements of the form

- X, Y are equal combinatory proof-terms of type A [X = Y : A].

The latter statement-forms are considered as abbreviations of the conjunction "X : A **and** X : A **and** X = Y", in the metalanguage.

The above statements are supposed to be relativized to contexts, in the obvious way.

So, in a [proof-] context Φ, the auxiliary syntax is as follows:

- ***epi-theoretic statements***: $\Phi \Vdash$ **t** :: $\mathcal{U}$, $\Phi \Vdash$ A :: **type**,
- ***classifications*** ["***assignments***"]: $\Phi \vdash_c$ X : A,
- ***equations*** ["***proof-equivalences***"]: $\Phi \vdash_c$ X = Y : A.

The epi-theoretic statements "$\Phi \Vdash$ A :: **type**" are usually phrased in a colloquial style. Formally, they are supposed to be such that the $\mathcal{U}$-parameters of A are among the $\mathcal{U}$-parameters of Φ.

8.1 Definition.

For all types/propositions A, B, C, in $\mathcal{L}[\supset,\forall]$,

$\underline{\mu}[\![A,B,C]\!] := \lambda_*x{:}A.\lambda_*y{:}(B\supset C).\lambda_*z{:}(A\supset B).y(zx)$,
$\underline{\mu}_{\mathcal{U}}[\![A,B]\!] := !_*u.\lambda_*y{:}(A\supset B).\lambda_*z{:}(\forall v.A[\![v]\!]).y(z[u])$.

8.2 Remark.

(1) For all types/propositions A, B, C in $\mathcal{L}[\supset,\forall]$, one has,

($\underline{\mu}$): [ ] $\vdash_c$ $\underline{\mu}[\![A,B,C]\!]$: A $\supset$ .B $\supset$ C $\supset$ (A $\supset$ B $\supset$ C),
($\underline{\mu}_{\mathcal{U}}$): [ ] $\vdash_c$ $\underline{\mu}_{\mathcal{U}}[\![A,B]\!]$: $\forall$u.(A $\supset$ B $\supset$.($\forall$v.A) $\supset$ B).

(3) One could have defined alternatively, in C[CQ] and $\pi_{\Leftarrow}$[CQ], for all types/propositions A, B, C, in $L[\supset,\forall]$,

$$\underline{\underline{B}}[\![A,B,C]\!] := \underline{\underline{B}}'[\![A\supset B,B,C]\!] \circ \underline{\underline{C}}_*[\![A,B]\!],$$
$$\underline{\underline{B}}_U[\![A,B]\!] := \underline{\underline{B}}'[\![\forall u.A,A,B]\!] \circ_U \underline{\underline{C}}_{*U}[\![A]\!],$$

with, e. g., if $\underline{\underline{B}}[\![A,B,C]\!]$ were primitive proof-combinators,

$\circ$ for $\underline{\underline{B}}[\![A\supset B\supset B,(B\supset C)\supset(A\supset B\supset C),A]\!]$ and
$\circ_U$ for $\underline{\underline{B}}_U[\![(\forall u.A)\supset A,(A\supset B)\supset((\forall u.A)\supset B))]\!]$.

As noted before, *the combinatory proof-theory* C[CQ] presupposes the syntax and the rules of $\pi_{\Leftarrow}$[CQ], i. e.,

- ***context rules***: < >, <I>, <K>, <K$\forall$>,
- ***type-specifications***: (I), (K), (S), (Ω), (Δ), (K_U), (S_U), ($@_U$), (Φ_U), [possibly: (B), (C), (B_U), (C_U)],
- ***type-derivation rules***: [$\supset$e], [$\forall$e], [$\forall$i]$_{oo}$.

The specific ***equational proof-behavior in*** C[CQ] is as follows:

8.3 Definition (*The combinatory proof theory* C[CQ]).

Note. In order to spare on typography, we agree on the fact that assumed uniformly, before $\vdash \varphi$, or $[x : D] \vdash \varphi$, or $[u :: \mathcal{U}] \Vdash \varphi$, in the statement of the rules below, is an arbitrary context Φ.

1. ***Ground equations.***

[I]: $\vdash X : A \Longrightarrow \vdash \underline{\underline{I}}[\![A]\!]\ X = X : A$,

[K]: $\vdash X : A,\ \vdash Y : B \Longrightarrow \vdash \underline{\underline{K}}[\![A,B]\!]\ X\ Y = X : A$,

[S]: $$\frac{\vdash F : A\supset.B\supset C,\ \vdash G : A\supset B,\ \vdash X : A}{\vdash \underline{\underline{S}}[\![A,B,C]\!]\ F\ G\ X = FX(GX) : C},$$

[K_U]: $\vdash X : A,\ \Vdash t :: \mathcal{U} \Longrightarrow \vdash \underline{\underline{K}}_U[\![A]\!]\ X\ [t] = X : A$ [u not in $FV_U(A)$],

[S_U]: $$\frac{\vdash F : \forall u.(A[\![u]\!]\supset B[\![u]\!]),\ \vdash G : \forall u.A[\![u]\!],\ \Vdash t :: \mathcal{U}}{\vdash \underline{\underline{S}}_U[\![A,B]\!]\ F\ G\ [t] = F[t](G[t]) : B[\![u:=t]\!]},$$

[$@_U$]: $$\frac{\vdash F : A\supset\forall u.B[\![u]\!],\ \Vdash t :: \mathcal{U},\ [u:\mathcal{U}]\vdash X : A}{\vdash \underline{\underline{@}}_U[\![A,B]\!]\ F\ [t]\ X = (FX)[t] : B[\![u:=t]\!]}\ [u \text{ not in } FV_U(A)],$$

[Φ_U]: $$\frac{\vdash F : \forall u.\forall v.A[\![u,v]\!],\ \Vdash s :: \mathcal{U},\ \Vdash t :: \mathcal{U}}{\vdash \underline{\underline{\Phi}}_U[\![A]\!]\ F\ [s]\ [t] = F[t][s] : A[\![u:=t,v:=s]\!]},$$

$[\underline{\Delta}\mathbf{T}]$: $\vdash F : \mathbf{T}^{-} \Longrightarrow \vdash \underline{\Delta}[\![\mathbf{T}]\!]\, F = \underline{\Omega} : \mathbf{T}$,

$[\underline{\Delta}\bot]$: $\vdash F : \bot^{-} \Longrightarrow \vdash \underline{\Delta}[\![\bot]\!]\, F = F(\underline{I}[\![\bot]\!]) : \bot$,

$$[\underline{\Delta}\supset]: \frac{\vdash F : (A\supset B)^{-},\ \vdash X : A}{\vdash \underline{\Delta}[\![A\supset B]\!]\, F = \underline{\Delta}[\![B]\!](F \circ \underline{\mu}[\![A,B,\bot]\!](X)) : B},$$

$$[\underline{\Delta}\forall]: \frac{\vdash F : (\forall u.A)^{-},\ \Phi \Vdash \mathbf{t} :: U}{\vdash \underline{\Delta}[\![\forall u.A]\!]\, F\,[\mathbf{t}] = \underline{\Delta}[\![A]\!](F \circ \underline{\mu}_{U}[\![A,\bot]\!][\mathbf{t}]) : A[\![u:=\mathbf{t}]\!]},$$

$[\underline{\Delta}\underline{\nabla}]$: $\vdash \underline{\Delta}[\![A]\!] \circ \underline{\nabla}[\![A]\!] = \underline{I}[\![A]\!] : A \supset A$.

2. *Congruence.*

$[\rho]$: $\vdash X : A \Longrightarrow \vdash X = X : A$,
$[\sigma]$: $\vdash X = Y : A \Longrightarrow \vdash Y = X : A$,
$[\tau]$: $\vdash X = Y : A,\ \vdash Y = Z : A \Longrightarrow \vdash X = Z : A$,

$[\mu\supset]$: $\vdash F = G : A \supset B,\ \vdash X : A \Longrightarrow \vdash FX = GX : B$,
$[\nu\supset]$: $\vdash F : A \supset B,\ \vdash X = Y : A \Longrightarrow \vdash FX = FY : B$,
$[\mu\forall]$: $\vdash F = G : \forall u.A,\ \Vdash \mathbf{t} :: U \Longrightarrow \vdash F[\mathbf{t}] = G[\mathbf{t}] : A[\![u:=\mathbf{t}]\!]$.

3. *Extensionality.*

$$[\mathrm{EXT}\ \lambda]: \frac{\vdash F : A \supset B,\ \vdash G : A \supset B,\ [x : A]\ \vdash Fx = Gx : B}{\vdash F = G : A \supset B},\ [x \text{ not in } F, G],$$

$$*[\mathrm{EXT}\ \partial]: \frac{\vdash X : A,\ \vdash Y : A,\ [x : A^{-}]\ \vdash xX = xY : \bot}{\vdash X = Y : A},\ [x \text{ not in } X, Y],$$

$$[\mathrm{EXT}\ !]: \frac{\vdash F : \forall u.A,\ \vdash G : \forall u.A,\ [u :: U]\ \vdash F[u] = G[u] : A[\![u]\!]}{\vdash F = G : \forall u.A}, [u \text{ not in } F, G].$$

8.4 Remark (*Some alternatives*).

(1) In $[\underline{\Delta}\underline{\nabla}]$ one can restrict A to *atomic* types/propositions.

(2) If $\underline{\mu}[\![A,B,C]\!]$, $\underline{\mu}_{U}[\![A,B]\!]$ are defined as in **8.2** (3), then one has, alternatively, for all types/propositions A, B, C, in $\mathcal{L}[\supset,\forall]$,

- in $[\underline{\Delta}\supset]$: $\underline{\mu}[\![A,B,\bot]\!] \equiv \underline{B}'[\![A\supset B,B,\bot]\!] \circ \underline{C}_{*}[\![A,B]\!]$ and
- in $[\underline{\Delta}\forall]$: $\underline{\mu}_{U}[\![A,\bot]\!][\mathbf{t}] \equiv \underline{B}'[\![\forall u.A[\![u]\!],A[\![u]\!],\bot]\!](\underline{C}_{*U}[\![A[\![u]\!]]\!][\mathbf{t}])$.

(3) If, for all A, B, C, in $\mathcal{L}[\supset,\forall]$, $\underline{B}[\![A,B,C]\!]$, $\underline{C}[\![A,B,C]\!]$, $\underline{B}_\upsilon[\![A,B]\!]$, $\underline{C}_\upsilon[\![A,B]\!]$ are among the ground combinators, then one needs also,

$$[\underline{B}]:\ \frac{\vdash F : A \supset B,\ \vdash G : C \supset A,\ \vdash X : C}{\vdash (F \circ G)X\ [\equiv_{df} \underline{B}[\![A,B,C]\!]\ F\ G\ X\ ] = F(GX) : B},$$

$$[\underline{C}]:\ \frac{\vdash F : A \supset .B \supset C,\ \vdash X : B,\ \vdash Y : A}{\vdash \underline{C}[\![A,B,C]\!]\ F\ X\ Y = FYX : C},$$

$$[\underline{B}_\upsilon]:\ \frac{\vdash F : A \supset B,\ \vdash G : \forall u.A,\ \Vdash t :: U}{\vdash \underline{B}_\upsilon[\![A,B]\!]\ F\ G\ [t] = F(G[t]) : B},\ [u \text{ not in } FV_\upsilon(A,B)],$$

$$[\underline{C}_\upsilon]:\ \frac{\vdash F : \forall u.(A \supset B),\ \vdash G : A,\ \Vdash t :: U}{\vdash \underline{C}_\upsilon[\![A,B]\!]\ F\ G\ [t] = F[t]G : B[\![u:=t]\!]},\ [u \text{ not in } FV_\upsilon(A)].$$

In these conditions, it is appropriate to assume that used are the *extended* "algorithms" for λ_* and $!_*$ [2.12 (2)].

(4) If $\underline{B}[\![A,B,C]\!]$ is a ground combinator then $\circ$ is supposed to be defined *via* $X \circ Y \equiv_{df} \underline{B}[\![A,B,C]\!](X)(Y)$ and, obviously,

- in $[\underline{\Delta}\supset]$: $\circ$ stands for $\underline{B}[\![(A\supset B)^-, \bot, B^-]\!]$,
- in $[\underline{\Delta}\forall]$: $\circ$ stands for $\underline{B}[\![(\forall u.A[\![u]\!])^-, \bot, (A[\![u]\!])^-]\!]$,
- in $[\underline{\Delta}v]$: $\circ$ stands for $\underline{B}[\![A^-, A, A]\!]$.

One should, however, realize, that λ_* is always assumed to be defined such as to make [$\circ$] hold [cf. 2.21 (1)], i. e.,

[$\circ$]: $X \circ Y \equiv \lambda_* x{:}C.X(Yx) : C \supset B$, for $\vdash X : A \supset B$, $\vdash Y : C \supset A$.

(5) The "$\supset$-monotony" rules [$\mu\supset$] and [$\nu\supset$] can be replaced by:

[$\mu\nu\supset$]: $\Phi \vdash F = G : A \supset B,\ \Phi \vdash X = Y : A \Longrightarrow \Phi \vdash FX = GY : B$,

(6) We can define, for all types/propositions A, in $\mathcal{L}[\supset,\forall]$,

$$\begin{aligned}
\underline{@}_{*\upsilon}[\![A]\!] &\equiv (\underline{@}_\upsilon[\![A,A]\!])(\underline{I}[\![\forall u.A]\!]) : \forall u.(\forall v.A[\![u:=v]\!] \supset A[\![u]\!]),\\
&\quad [\text{provided } v \text{ is not in } FV_\upsilon(A[\![u]\!])],\\
\underline{@}_t[\![A]\!] &\equiv (\underline{@}_{*\upsilon}[\![A]\!])[t] : \forall v.A[\![v]\!] \supset A[\![u:=t]\!],\ [\ \Vdash t :: U\ ],\\
\underline{@}_t[\![A]\!]\ F &\equiv (\underline{@}_{*\upsilon}[\![A]\!])[t]F\ [\equiv (\underline{@}_\upsilon[\![A,A]\!])(\underline{I}[\![\forall u.A]\!])[t]F],\\
&\quad [\text{in context } \Phi, \text{ for } F \text{ such that } \Phi \vdash F : \forall v.A[\![v]\!]],
\end{aligned}$$

whence the *instantiation rule* could have been expressed by:

$$(\forall e):\quad \Phi \Vdash t :: U,\ \Phi \vdash F : \forall v.A[\![v]\!] \Longrightarrow \Phi \vdash \underline{@}_t[\![A]\!]\ F : A[\![u:=t]\!],\ [\text{provided } v \text{ is not in } FV_\upsilon(A[\![u]\!])].$$

We can easily show that, from an equational point of view, C[CQ] is not "more" than $\lambda\mathcal{J}!$.

8.5 Theorem (*$\lambda\eta!$-Embedding for* **C[CQ]**).

For all types/propositions A, in $L[\sqsupset,\forall]$, and all proof-combinators X, Y in **C[CQ]**,

$$\Phi \vdash_c X = Y : A \Longrightarrow \Phi \vdash (X)^{\vdash} = (Y)^{\vdash} : A$$

Proof. By induction on the [length of] derivation of the premiss: the equational behavior of $\underline{I}[\![A]\!]$, $\underline{K}[\![A,B]\!]$, $(\underline{B}[\![A,B,C]\!]$, $\underline{C}[\![A,B,C]\!])$, $\underline{S}[\![A,B,C]\!]$, etc. is straightforward. For $[\underline{\Delta}\top]$, $[\underline{\Delta}\bot]$, $[\underline{\Delta}\sqsupset]$, $[\underline{\Delta}\forall]$, $[\underline{\Delta}\nabla]$, one has the analogous "closed" derived rules of section 6. The congruence conditions are immediate. Finally, the images of [EXT λ], [EXT η], [EXT !] have been obtained in section 6. □

This yields immediately the following [Post-] consistency result.

8.6 Theorem (*Post-consistency*).

Cons[C[CQ]] (i. e., the proof-equality of **C[CQ]** is non-trivial).

Proof. From **Cons[$\lambda\eta!$]** and the *$\lambda\eta!$-Embedding Theorem* above. □

The converse embedding is meant to show that **C[CQ]** can "simulate" the equational behavior of $\lambda\eta!$ and requires several preliminary lemmas.

8.7 Lemma (*Positive β-rules*).

For all types/propositions A, in $L[\sqsupset,\forall]$, and all combinatory proof-terms X, Y in **C[CQ]**,

$$(\beta\sqsupset\lambda_*):\quad \frac{\Phi \vdash X : A \qquad \Phi[\, x : A\,] \vdash Y[\![x]\!] : B}{\Phi \vdash (\lambda_* x{:}A.\, Y[\![x]\!])X = Y[\![x{:=}X]\!] : B}$$

$$(\beta\forall!_*):\quad \frac{\Phi \Vdash t :: U \qquad \Phi[\, u :: U\,] \vdash X[\![u]\!] : A[\![u]\!]}{\Phi \vdash (!_* u.\, X[\![x]\!])[t] = X[\![u{:=}t]\!] : A[\![u{:=}t]\!].}\ [\text{u not free in t}],$$

Proof. $(\beta\sqsupset\lambda_*)$: By induction on the structure of $Y[\![x]\!]$. $(\beta\forall!_*)$: By induction on the structure of $X[\![u]\!]$. □

8.8 Corollary (*Positive η-rules*).

For all types/propositions A, in $L[\sqsupset,\forall]$, and all combinatory proof-terms F, in **C[CQ]**,

$(\eta\sqsupset\lambda_*)$: If x is not in F, $\Phi \vdash F : A \sqsupset B \Longrightarrow \Phi \vdash \lambda_* x{:}A.\, Fx = F$.

$(\eta\forall!_*)$: If u is not in F, $\Phi \vdash F : \forall u.A \Longrightarrow \Phi \vdash !_* u.\, F[u] = F$.

Proof. From $(\eta\sqsupset\lambda_*)_=$ and $(\eta\forall!_*)_=$, in **2.15**, where we had the same rules with $\equiv$ in place of $=$. □

In presence of $(\eta\sqsupset\lambda_*)$ and $(\eta\forall!_*)$, the "extensionality" of the λ_*- and $!_*$-proof-operations can be expressed as follows.

8.9 Lemma (*Positive* ξ*-rules*).

For all types/propositions A, in $L[\sqsupset,\forall]$, and all combinatory proof-terms X, Y in $C[CQ]$,

$$(\xi\sqsupset\lambda_*):\quad \frac{\Phi[\ x : A\ ] \vdash X[\![x]\!] = Y[\![x]\!] : B}{\Phi \vdash \lambda_*x{:}A.X[\![x]\!] = \lambda_*x{:}A.Y[\![x]\!] : A \sqsupset B},$$

$$(\xi\forall!_*):\quad \frac{\Phi[\ u :: \mathcal{U}\ ] \vdash X[\![u]\!] = Y[\![u]\!] : A[\![u]\!]}{\Phi \vdash !_*u.X[\![u]\!] = !_*u.Y[\![u]\!] : \forall u.A[\![u]\!]}.$$

Proof. $(\xi\sqsupset\lambda_*)$: Assume $\Phi[\ x : A\ ] \vdash X[\![x]\!] = Y[\![x]\!] : B$. Then, one has also $\Phi[x{:}A] \vdash X[\![x]\!] = (\lambda_*x{:}A.X[\![x]\!])x = (\lambda_*x{:}A.Y[\![x]\!])x : B$, by $(\beta\sqsupset\lambda_*)$, whence $\Phi \vdash \lambda_*x{:}A.X[\![x]\!] = \lambda_*x{:}A.Y[\![x]\!] : A \sqsupset B$, by [EXT λ].

$(\xi\forall!_*)$: Analogously. Assume $\Phi[\ u :: \mathcal{U}\ ] \vdash X[\![u]\!] = Y[\![u]\!] : A[\![u]\!]$. Then, $\Phi[u :: \mathcal{U}] \vdash X[\![u]\!] = (!_*u.X[\![u]\!])[u] = (!_*u.Y[\![u]\!])[u] = Y[\![u]\!] : A[\![u]\!]$, by $(\beta\forall!_*)$, whence $\Phi \vdash !_*u.X[\![u]\!] = !_*u.Y[\![u]\!] : \forall u.A[\![u]\!]$, by [EXT !]. □

8.10 Lemma (*Negative* [$\eth_*$-] *extensionality*).

For all types/propositions A, in $L[\sqsupset,\forall]$, and all combinatory proof-terms F, in $C[CQ]$,

$(\eta\sqsupset\eth_*)$: If x is not in F, $\Phi \vdash F : A \Longrightarrow \Phi \vdash \eth_*x{:}A^-.xF = F$.

$$(\xi\sqsupset\eth_*):\quad \frac{\Phi[\ x : A^-\ ] \vdash X[\![x]\!] = Y[\![x]\!] : \bot}{\Phi \vdash \eth_*x{:}A^-.X[\![x]\!] = \eth_*x{:}A^-.Y[\![x]\!] : A}.$$

Proof. $(\eta\sqsupset\eth_*)$: Assume that x is not in F and $\Phi \vdash F : A$. In **2.16** we had, with $\underline{\forall}_=[\![A]\!](F) := \underline{S}[\![A^-,A,\bot]\!](\underline{I}[\![A^-]\!])(\underline{K}[\![A,A^-]\!](F))$,

$(\eta\sqsupset\eth_*)_=$: $\Phi \vdash F : A \Longrightarrow \Phi \vdash \eth_*x{:}A^-.xF \equiv \underline{\Delta}[\![A]\!](\underline{\forall}_=[\![A]\!](F)) : A$,

in the same conditions. The rules of $C[CQ]$ yield then

$$\Phi[\ x : A^-\ ] \vdash \underline{\forall}_=[\![A]\!](F)(x) = \underline{S}[\![A^-,A,\bot]\!](\underline{I}[\![A^-]\!])(\underline{K}[\![A,A^-]\!](F))(x) = \underline{I}[\![A^-]\!](x)(\underline{K}[\![A,A^-]\!](F)(x)) = xF,$$

whence $\Phi \vdash \underline{\forall}_=[\![A]\!](F) = \lambda_*x{:}A^-.\underline{\forall}_=[\![A]\!](F)(x) = \lambda_*x{:}A^-.xF : A^=$, by $(\eta\sqsupset\lambda_*)$, $(\xi\sqsupset\lambda_*)$, etc. But, $\underline{\forall}[\![A]\!] \equiv_{df} C_*[\![A,\bot]\!] \equiv \lambda_*x{:}A.\lambda_*y{:}A^-.yx$. So, by $(\beta\sqsupset\lambda_*)$, one obtains $\Phi \vdash \underline{\forall}[\![A]\!](F) \equiv (\lambda_*x{:}A.\lambda_*y{:}A^-.yx)(F) = \lambda_*y{:}A^-.yF : A^= \equiv_\alpha \lambda_*x{:}A^-.xF = \underline{\forall}_=[\![A]\!](F) : A^=$ and this yields, by $(\vartheta\sqsupset)$, using $(\eta\sqsupset\eth_*)_=$,

$$\Phi \vdash \underline{\Delta}[\![A]\!](\underline{\nabla}[\![A]\!](F)) = \underline{\Delta}[\![A]\!](\underline{\nabla}_{=}[\![A]\!](F)) \equiv \gamma_{*}x{:}A^{-}.xF : A.$$

From this, by [$\underline{I}$], [$\underline{\Delta\nabla}$], [$\circ$] (cf. 2.21), ($\beta\supset\lambda_{*}$), ($\mu\supset$) and ($\nu\supset$),

$$\Phi \vdash F = \underline{I}[\![A]\!](F) = (\underline{\Delta}[\![A]\!] \circ \underline{\nabla}[\![A]\!])(F) \equiv (\lambda_{*}x{:}C.\,\underline{\Delta}[\![A]\!](\underline{\nabla}[\![A]\!](x))(F) = \underline{\Delta}[\![A]\!](\underline{\nabla}[\![A]\!](F)) = \gamma_{*}x{:}A^{-}.xF : A.$$

($\xi\supset\gamma_{*}$): By ($\xi\supset\lambda_{*}$) and ($\nu\supset$). □

8.11 Remark ($\beta\gamma_{*}$-*rules*).

It is easy to see that the $*$-analogues of the $\beta\gamma$-rules of $\lambda\gamma!$ are also available in **C[CQ]** [*exercise*]. (*Hint*: one must use the $\underline{\Delta}$-rules of **C[CQ]**, viz. [$\underline{\Delta}\top$], [$\underline{\Delta}\bot$], [$\underline{\Delta}\supset$] and [$\underline{\Delta}\forall$].)

8.12 Remark.

Given [$\underline{\Delta\nabla}$] and the monotony rules ($\mu\supset$), ($\nu\supset$), one can show that the rule [EXT γ] is, in fact, redundant. Indeed, assume $\Phi \vdash X : A$, $\Phi \vdash X, Y : A$ and $\Phi\,[x : A^{-}] \vdash xX = xY : \bot$. Then, by ($\xi\supset\gamma_{*}$), one has $\Phi \vdash \gamma_{*}x : A^{-}.xX = \gamma_{*}x : A^{-}.xY$, whence $\Phi \vdash X = Y : A$, by ($\eta\supset\lambda_{*}$). From the above, it is also clear that the proof of ($\xi\supset\gamma_{*}$) does not use [EXT γ].

8.13 Theorem (*Combinatory embedding of* $\lambda\gamma!$).

For all types/propositions A in $\mathcal{L}[\supset,\forall]$, all proof-terms a, b in $\lambda\gamma!$, and all contexts Φ,

$$\Phi \vdash a = b : A \Longrightarrow \Phi \vdash (a)^{c} = (b)^{c} : A.$$

Proof. By induction on the [length of] derivation of the premiss: ($\beta\supset\lambda$), ($\beta\forall$), ($\eta\supset\lambda$), ($\eta\forall$), resp. translate into ($\beta\supset\lambda_{*}$), ($\beta\forall!_{*}$), ($\eta\supset\lambda_{*}$), ($\eta\forall!_{*}$), resp. and ($\eta\supset\gamma$) follows from ($\eta\supset\gamma_{*}$), whereas **8.11** yields the ($\beta\gamma_{*}$-rules). The ξ_{*}-rules ($\xi\supset\lambda_{*}$), ($\xi\supset\gamma_{*}$), ($\xi\forall!_{*}$) have been obtained above; the remaining congruence rules are already in the primitive combinatory syntax. □

So, ultimately, the combinatory proof-theory **C[C(Q)]** is equivalent to $\lambda\gamma(!)$. One can also see easily that the result obtains for the corresponding propositional fragments (as parenthesized).

In the end, from **4.3** one has, *a fortiori*, the following,

8.14 Corollary

For all types/propositions A, in $\mathcal{L}[\supset,\forall]$, any combinatory proof-term X in **C[CQ]**, and any context Φ,

$$\Phi \vdash_{c} X : A \Longrightarrow \Phi \vdash_{c} (X^{\llcorner})^{c} = X : A.$$

Proof. In **8.3** we had, for π_{c}**[CQ]**, $\Phi \vdash_{c} X : A \Longrightarrow \Phi \vdash_{c} (X^{\llcorner})^{c} \equiv X$, in the same conditions. □

In order to complete the circle, we can note also the following:

8.15 Lemma.

For all types/propositions A, B in $\mathcal{L}[\supset,\forall]$, all combinatory proof-terms X, in $\mathbf{C}[\mathbf{CQ}]$, and any context Φ,

(1) $\Phi\,[\,x : A\,] \vdash_c X : B \Longrightarrow \Phi \vdash (\lambda_* x{:}A.X)^{\llcorner} = \lambda x{:}A.(X)^{\llcorner} : A \supset B$,

(2) $\Phi\,[\,x : A^-\,] \vdash_c X : \bot \Longrightarrow \Phi \vdash (\gamma_* x{:}A^-.X)^{\llcorner} = \gamma x{:}A^-.(X)^{\llcorner} : A$,

(3) $\Phi\,[\,u :: \boldsymbol{U}\,] \vdash_c X : A \Longrightarrow \Phi \vdash (!_* u.X)^{\llcorner} = !u.(X)^{\llcorner} : \forall u.A$,

Proof. (1), (3): By induction on the structure of X. (2): From (1) and the definition of γ_*. Indeed, if $\Phi\,[\,x : A^-\,] \vdash_c X : \bot$ then applying the (**IH**) tacitly, we have

$$\begin{aligned}\Phi \vdash (\gamma_* x{:}A^-.X)^{\llcorner} &\equiv_{df} ((\underline{\Delta}[\![A]\!](\lambda_* x{:}A^-.X))^{\llcorner} = \\ &= (\underline{\Delta}[\![A]\!])^{\llcorner}(\lambda_* x{:}A^-.X)^{\llcorner} = [\text{using (1)}] \\ &= (\Delta[\![A]\!])(\lambda x{:}A^-.(X)^{\llcorner}) = (\lambda x{:}A^-.\gamma y{:}A^-.xy)(\lambda x{:}A^-.(X)^{\llcorner}) \Longrightarrow \\ &\Longrightarrow \gamma y{:}A^-.(\lambda x{:}A^-.(X)^{\llcorner})y \Longrightarrow \gamma y{:}A^-.(X)^{\llcorner}[\![x:=y]\!] \equiv_{\alpha} \gamma x{:}A^-.(X)^{\llcorner}. \quad \square\end{aligned}$$

8.16 Theorem.

For all types/propositions A in $\mathcal{L}[\supset,\forall]$, all p-terms a in $\lambda\gamma!$, and any context Φ, $\Phi \vdash a : A \Longrightarrow \Phi \vdash ((a)^{c})^{\llcorner} = a : A$.

Proof. By induction on the structure of a. If a is a p-variable or Ω, the result is immediate. If a is of the form bc or b[t], the (**IH**) yields the result directly. Else, if a is $\lambda x{:}A.b$, $\gamma x{:}A^-.b$ or $!u.b$ the (**IH**) yields the result from **8.15**. $\square$

9. Détour elimination. Intuitively, applying a reduction rule to a proof-term in $\lambda\mathcal{G}!$ amounts to the elimination of a *proof-détour* (*Beweisumweg*, in Gentzen's terms). This way of speaking has an *operational* reading and can be made technically precise. *Mutatis mutandis*, the following terminology is as in [Barendregt 1984].

9.1 Definition.

(1) A *notion of reduction* for $\lambda\mathcal{G}!$ is just a binary relation $\mathbf{r}$ on $\mathrm{Term}[\lambda\mathcal{G}!]$.

(2) The p-terms falling within the domain of a notion of reduction $\mathbf{r}$ are identified as $\mathbf{r}$-*détours*, or $\mathbf{r}$-*redexes*, whereas the range [co-domain] of $\mathbf{r}$ gives the associated *contracta*.

(3) A p-term a is said to be $\mathbf{r}$-*normal* or *in* $\mathbf{r}$-*normal form* if no subterm of a is a $\mathbf{r}$-détour.

9.2 Notation.

If $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n$ are notions of reduction then $\mathbf{r}_1\mathbf{r}_2\ldots\mathbf{r}_n$ stands for the (set-) union $\mathbf{r} = \bigcup_{i\leq n} \mathbf{r}_i$.

9.3 Example.

$[\eta\supset\mathcal{G}] = \{ (\mathcal{G}x{:}A^{-}.xf,\ f) : f \in \mathrm{Term}[\lambda\mathcal{G}!],\ x \text{ not in } FV(f) \}$ is a notion of reduction in $\lambda\mathcal{G}!$. Then the $[\eta\supset\mathcal{G}]$-détours, determined by $[\eta\supset\mathcal{G}]$, are p-terms of the form $\mathcal{G}x{:}A^{-}.xf$, with $f \in \mathrm{Term}[\lambda\mathcal{G}!]$, and x not in $FV_{\lambda}(f)$.

Equivalently, a notion of (proof-) reduction can be defined by specifying exhaustively the typology of $\mathbf{r}$-détours and $\mathbf{r}$-contracta (by stating "$\mathbf{r}$-contraction rules", say).

9.4 Remark.

In fact, $\lambda\mathcal{G}!$ has a *standard notion of reduction*: $\mathbf{r}[\lambda\mathcal{G}!]$ or, simplifying the notation,

$$\lambda\mathcal{G}! = [\beta\supset\lambda][\eta\supset\lambda][\beta\mathcal{G}][\eta\supset\mathcal{G}][\beta\forall][\eta\forall],$$

where $[\beta\mathcal{G}]$ is short for $[\beta\mathcal{G}\top][\beta\mathcal{G}\bot][\beta\mathcal{G}\supset][\beta\mathcal{G}\forall]$. This notion is a *complex* one and can be further analyzed into $\lambda = [\beta\supset\lambda][\eta\supset\lambda]$, $\mathcal{G} = [\beta\mathcal{G}][\eta\supset\mathcal{G}]$ and $! = [\beta\forall][\eta\forall]$, resp. and, in more detail, by distinguishing (as in work of H. B. Curry *et al.*) among β- and η-type notions: $[\beta\supset\lambda]$ *vs* $[\eta\supset\lambda]$, $[\beta\mathcal{G}]$ *vs* $[\eta\supset\mathcal{G}]$, etc. Each such a notion can be given by an appropriate "$\mathbf{r}$-contraction rule", as already suggested in the presentation of $\lambda\mathcal{G}!$.

9.5 Notation. Terminology.

If r is a particular notion of reduction then

(1) $_r\!\!\rightarrow\!\!\gg_1$ (*one-step r-reduction*) is the least relation on Term[λ𝒥!], such that $_r\!\!\rightarrow\!\!\gg_1$ contains r and is compatible with the syntatic operations in Term[λ𝒥!],
(2) $_r\!\!\rightarrow\!\!\gg$ (*r-contraction*) is the reflexive closure of $_r\!\!\rightarrow\!\!\gg_1$,
(3) $_r\!\!=\!\!\gg$ (*r-reduction* or *r-reducibility*) is the transitive closure of $_r\!\!\rightarrow\!\!\gg$,
(4) $_r=$ (*r-conversion* or *r-equality* or *r-convertibility*) is the least congruence on Term[λ𝒥!] relative to the syntactic operations in Term[λ𝒥!] or, alternatively, the equivalence generated by $_r\!\!\rightarrow\!\!\gg$.

9.6 Remark.

One has, $_r\!\!\rightarrow\!\!\gg_1 \subset {}_r\!\!\rightarrow\!\!\gg \subset =\!\!\gg \subset {}_r=$, for any notion of reduction r on Term[λ𝒥!].

9.7 Definition.

In particular, if $\Phi \vdash a\ {}_r\!\!=\!\!\gg b$, for some r-normal p-term b, then b is said to be *a r-normal form of* a.

Reduction sequences, normalization. From an *operational* point of view, the r-normal p-terms can be thought of as *limits* of certain sequences of p-terms. One can elaborate on this, with some profit.

9.8 Definition.

(1) If r is a notion of reduction, a *r-reduction sequence from* a is a (possibly infinite) sequence of p-terms in Term[λ𝒥!] $\rho := a, a_1, a_2, a_3, \ldots$ such that

$$\Phi \vdash a\ {}_r\!\!\rightarrow\!\!\gg_1 a_1\ {}_r\!\!\rightarrow\!\!\gg_1 a_2\ {}_r\!\!\rightarrow\!\!\gg_1 a_3\ {}_r\!\!\rightarrow\!\!\gg_1 \ldots,$$

i. e., the members of ρ are linearly ordered ["connected"] by one-step r-reductions.

(2) If r is as above and a is a p-term in Term[λ𝒥!], then

$$\Sigma[r,a] = \{\ \rho : \rho \text{ is a r-reduction sequence from a }\}$$

is the *r-reduction spectrum of* a.

NB: Σ[r,a] can be viewed as a *tree* of p-terms, with bottom a.

9.9 Notation.

(1) We write **lh**(ρ) for the number of terms of ρ, if ρ is finite, else, for convenience, **lh**(ρ) = ω (where ω denotes the first transfinite ordinal).

(2) If **r** is fixed, we write $\Sigma(a)$ for the r-reduction spectrum of a.

9.10 Definition.

Let **r** be a notion of reduction on Term[$\lambda\mathcal{G}!$]. A proof-term $a \in$ Term[$\lambda\mathcal{G}!$] is **r**-*bounded* (*bounded relative to* **r** or even "strongly normalizable relative to **r**"), if there is a (non-negative) integer n such that no r-reduction sequence ρ from a has **lh**(ρ) > n. In this case, n = **bnd**(ρ,a) is the least such n.

Therefore, $\Phi \vdash \rho \equiv a \ {}_r\!\!\twoheadrightarrow_1 a_1 \ {}_r\!\!\twoheadrightarrow_1 a_2 \ {}_r\!\!\twoheadrightarrow_1 a_3 \ {}_r\!\!\twoheadrightarrow_1 \ldots$ implies that **bnd**(ρ,a) > **bnd**(ρ_1,a_1) > **bnd**(ρ_2,a_2) > **bnd**(ρ_3,a_3) > ..., where ρ_i is the r-reduction-sequence from a_i.

9.11 Remark.

(1) So, if a is **r**-bounded, the **r**-reduction spectrum of a contains only "bounded paths/branches"; that is: every **r**-reduction sequence $\rho \in \Sigma$[**r**,a] is finite ["any $\rho \in \Sigma$[**r**,a] terminates in a finite number of steps" or still, negatively, "there is no infinite **r**-reduction sequence from a"].

(2) In terms of trees, this amounts to the fact that the **r**-reduction spectrum Σ[**r**,a] of a has no infinite branch. Conversely, the existence of such a tree implies the fact that a is r-bounded (this requires the use of König's Lemma [**KL**], however). Indeed, Σ[**r**,a] is a finitely branching tree, by definition (since a p-term has a only a finite number of subterms). If it has no infinite branch, it is [by **KL**] finite, whence the existence of an integer **bnd**(a) > **bnd**(ρ,a), for every branch $\rho_{r,a}$ of Σ[**r**,a].

9.12 Notation.

If $\underline{\lambda}$ is a typed λ-calculus and **r** is, *mutatis mutandis*, a notion of reduction (for $\underline{\lambda}$, defined on the set Term[$\underline{\lambda}$] of $\underline{\lambda}$-terms), then "**SN**[$\underline{\lambda}$,**r**]" abbreviates the statement "if $\Phi \vdash a : A$ in $\underline{\lambda}$, then a is bounded, for all $a \in$ Term[$\underline{\lambda}$]". We write also **SN**[$\lambda\mathcal{G}!$] or **SN**[**r**], for **SN**[$\lambda\mathcal{G}!$,**r**], if **r** is *the standard notion of reduction* of $\lambda\mathcal{G}!$.

SN[**r**] reads "**r** is strongly normalizable" and states *the existence* of a bound **bnd**(a) > **bnd**(ρ,a), for every r-reduction sequence ρ from a. A compact equivalent statement [by **KL**] amounts to the fact that **r**-*reduction spectra* [of p-terms] *are finite*.

In particular, if a is r-bounded and **SN**[$\lambda\mathcal{G}!$,**r**] holds then any ρ in the **r**-reduction spectrum of a will eventually terminate in a **r**-*normal* p-term b such that b is *a* **r**-*normal form* of a.

9.13 Remark.

Note that **SN**[$\lambda\mathcal{G}!$,**r**] does *only* establish the *existence* of r-normal forms (in $\lambda\mathcal{G}!$), it does not state anything about their *unicity*, however. Usually, this is shown separately.

9.14 Remark.

A weaker statement **WN**[$\lambda\gamma!$, r] ("weak normalization" or "weak normalizability") states the existence of a finite path in Σ[r, a], for every p-term a.

Confluence statements and confluence proofs. Let $\underline{\lambda}$ be a λ-calculus, (e. g., $\underline{\lambda}$ restricts $\lambda\gamma!$ to a subset Term[$\underline{\lambda}$] of Term[$\lambda\gamma!$] or $\underline{\lambda}$ extends $\lambda\gamma!$) and r be an arbitrary notion of reduction [a binary relation] on Term[$\underline{\lambda}$]. With $\underline{\lambda}$ fixed arbitrarily, the remaining notions above can be defined relative to Term[$\underline{\lambda}$] and r in a similar way.

9.15 Notation. Terminology (*Confluence*).

Then **CR**[$\underline{\lambda}$, r] is a statement about the associated relation $\twoheadrightarrow_r$ of r-*reduction* [the transitive closure of $\rightarrow_r$], viz., "for all a, b_i (i := 1,2),

$$\Phi \vdash a \twoheadrightarrow_r b_i \Longrightarrow \Phi \vdash b_i \twoheadrightarrow_r c,$$

for some c". This is the so-called "Church-Rosser property" for $\twoheadrightarrow_r$, or yet "the diamond property". If **CR**[$\underline{\lambda}$, r] holds then $\twoheadrightarrow_r$ (or $\underline{\lambda}$, if r is the standard notion of reduction in $\underline{\lambda}$) is said to be *confluent*.

The "residuation" method. Usually, a confluence property must be shown separately, for each $\underline{\lambda}$ and each notion of reduction r on Term[$\underline{\lambda}$]. There is, however, a nearly *standard method* of proving **CR**-statements, applying uniformly to a large class of pairs [$\underline{\lambda}$, r], where $\underline{\lambda}$ is a λ-calculus, and r is a notion of reduction on Term[$\underline{\lambda}$]. The method is due to J. B. Rosser, W. W. Tait and P. Martin-Löf ([Rezus 1981], [Barendregt 1984], [Takahashi 1989]).

For a given "**CR**-pair" [$\underline{\lambda}$, r], the technique [the *residuation method*] consists of endowing the given $\underline{\lambda}$ with a *residual* (or *parallel*) *extension* $^{\sim}$r (of r), "*the residuation of* r".

Residuations can be described *schematically*. If r is an arbitrary binary relation, let r° be the transitive closure of r and $r^{=\circ}$ be the reflexive and transitive closure of r.

Now, for a given pair [$\underline{\lambda}$, r], the *residuation* $^{\sim}$r *of* r (*on* $\underline{\lambda}$) is a relation on Term[$\underline{\lambda}$], the set of terms of $\underline{\lambda}$, such that:

(0°) $^{\sim}$r can be viewed as a simultaneous operation on (sequences of "non-interferring") [$\underline{\lambda}$-r]-détours, attempting to eliminate [$\underline{\lambda}$-r]-détours in "parallel", so to speak, whereas, technically,
(1°) $^{\sim}$r is a reflexive relation on Term[$\underline{\lambda}$],
(2°) $^{\sim}$r can be shown to be compatible with the syntactic operations of $\underline{\lambda}$,
(3°) $(^{\sim}r)^{\circ} \subseteq (\rightarrow_r)^{=\circ} = (\twoheadrightarrow_r)^{\circ}$,

[That is: $(\tilde{r})^\circ$, the transitive closure of $\tilde{r}$ is contained in the reflexive and transitive closure of $\to_{r}\!\!\gg_1$ (the "one-step" r-reduction relation on Term[$\underline{\lambda}$]), or - equivalently - the transitive closure of $\to_{r}\!\!\gg$ (the r-contraction on Term[$\underline{\lambda}$]) contains the transitive closure of $\tilde{r}$.]

(4°) one can prove **CR**[$\underline{\lambda}$,$\tilde{r}$] easily.

From (1°)-(2°) one has immediately

(5°) $\to_{r}\!\!\gg_1 \subseteq \to_{r}\!\!\gg \subseteq \tilde{r}$, whence
(6°) $(\to_{r}\!\!\gg_1)^{=\circ} \subseteq (\to_{r}\!\!\gg)^\circ \subseteq (\tilde{r})^\circ$, and, finally, by (3°),
(7°) $(\to_{r}\!\!\gg_1)^{=\circ} = (\to_{r}\!\!\gg)^\circ = (\tilde{r})^\circ$.

The intended result, **CR**[$\underline{\lambda}$,r], comes out by a technical lemma.

9.16 Lemma (*Transitive closures respect confluence*).

If r and $\tilde{r}$ have the same transitive closure, as relations on Term[λ], then **CR**[$\underline{\lambda}$,$\tilde{r}$] => **CR**[$\underline{\lambda}$,r].

Proof. Easy. [*Exercise*]. □

9.17 Remark ("*Residuals*").

The proof of a **CR**[$\underline{\lambda}$,$\tilde{r}$]-statement for a residuation $\tilde{r}$ on $\underline{\lambda}$ can be reduced, in a standard way, to a proof of a statement about/on "residuals".

Schematically again, where $\to_{\tilde{r}}\!\!\gg$ stands for the $\tilde{r}$-reduction relation on Term[$\underline{\lambda}$], the "r-residuals" are terms of the form $(a)^{r\bullet\bullet}$ in Term[$\underline{\lambda}$], generated inductively from terms a in Term[$\underline{\lambda}$], such that one has the following "residual-inversion" property, **res**[$\underline{\lambda}$,$\tilde{r}$], for short:

for all a, b ∈ Term[$\underline{\lambda}$], $\Phi \vdash a \to_{\tilde{r}}\!\!\gg b$ ==> $\Phi \vdash b \to_{\tilde{r}}\!\!\gg (a)^{r\bullet\bullet}$.

Obviously, **res**[$\underline{\lambda}$,$\tilde{r}$] ==> **CR**[$\underline{\lambda}$,$\tilde{r}$]. So, finally, the main task in the proof of a **CR**[$\underline{\lambda}$,r]-statement, by the "residuation" technique, would mainly consists of finding an appropriate inductive way of generating "r-residuals" of terms in Term[$\underline{\lambda}$].

9.18 Remark ("*Tracing a residual-history*").

There are analogous techniques meant to exploit induction on "residuals", consisting mainly of a systematic "tracing" of a "residual history" (as, e. g., by "underlining" certain types of détours - the ones that should be eliminable - and by "freezing" the remaining ones, in order to "block" their elimination). Such techniques amount, essentially, to a similar attempt of *making explicit recursion on "residuals"*. (See, e. g., [Klop 1980], [Barendregt 1984] for examples.)

Weak confluence. There is also a weaker statement, **WCR**[$\underline{\lambda}$,r] say, connecting r-contractions and proper r-reductions, in a similar "diamond-like" way.

9.19 Notation. Terminology (*Weak confluence*).

If r is a notation of reduction on Term[$\underline{\lambda}$] then **WCR**[$\underline{\lambda}$,r] is the statement "for all a, b_i (i := 1,2),

$$\Phi \vdash a \ {}_r\!\!\twoheadrightarrow b_i \Rightarrow \Phi \vdash b_i \ {}_r\!\!\twoheadrightarrow c,$$

for some c". Alternative way of speaking: "$_r\!\!\twoheadrightarrow$ is weakly confluent" or even "$_r\!\!\twoheadrightarrow$ is weakly Church-Rosser".

9.20 Remark.

If r is a notation of reduction r on Term[$\underline{\lambda}$], then one has **CR**[$\underline{\lambda}$,r] ==> **WCR**[$\underline{\lambda}$,r], but *not* conversely.

For a large variety of notions of reduction r on Term[$\underline{\lambda}$], **WCR**[$\underline{\lambda}$,r] is, usually, easier to prove than **CR**[$\underline{\lambda}$,r]. **WCR**[$\underline{\lambda}$,r]-properties are useful, in presence of stronger [$\underline{\lambda}$-r]-properties. For most typed λ-calculi $\underline{\lambda}$, there is a a well-known strategy of obtaining **CR**[$\underline{\lambda}$,r]-proofs, in view of the following:

9.21 Fact (M. H. A. Newman 1942).

If $\underline{\lambda}$ is a λ-calculus and r a notion of reduction r on Term[$\underline{\lambda}$], then **WCR**[$\underline{\lambda}$,r] **&** **SN**[$\underline{\lambda}$,r] ==> **CR**[$\underline{\lambda}$,r].

Proof. By *reductio ad absurdum*. □

Recall that λ is the ordinary typed λ-calculus with types taken from $\mathcal{L}[\supset]$ (this language has also $\top$ and $\bot$ as "atomic types").

9.22 Fact.

SN[λ].

Proof. Well-known. Cf., e. g., [Tait 1967] or [Hindley & Seldin 1986], **Appendix 2**. □

Usually, **SN**[λ] is obtained as a by-product, from stronger results. In fact, as we show next, we hardly need more for present purposes.

Inasfar the reduction behavior of proof-terms is concerned, $\lambda!$, the $\eth$-free part of $\lambda\eth!$, is an inessential extension of λ. One can check easily the following

9.23 Fact.

SN[$\lambda!$] <==> **SN**[λ].

Proof. (As expected, **λ!** has types/propositions from $L[\supset,\forall]$.) Fix a valuation/interpretation # of the U-variables/U-terms of **λ!** in Term[**λ**], as ever. Define then a mapping

$$(...)^\circ : \mathrm{Term}[\boldsymbol{\lambda!}] \longrightarrow \mathrm{Term}[\boldsymbol{\lambda}]$$

with $(a)^\circ \equiv a$, for all $a \in \mathrm{Term}[\boldsymbol{\lambda}]$ and also $(!u.a)^\circ \equiv \lambda x{:}A.(a)^\circ$, $(f[t])^\circ \equiv (f)^\circ t^*$, for an appropriate type/proposition A [set, e. g., $A \equiv \bot \supset \bot$, for instance, or whatever else, compatible with the choice of #], such that

$$\Phi \vdash_{\boldsymbol{\lambda}!} a \Rightarrow b \Longrightarrow \Phi_* \vdash_{\boldsymbol{\lambda}} a^\circ \Rightarrow b^\circ,$$

where Φ_* is obtained from Φ by valuating the U-variables *via* #. (There are such °'s [*exercise*].) This shows **SN[λ!]** <== **SN[λ]**. The converse is immediate: indeed, **λ** is a sub-system of **λ!**, w. r. t. reduction behaviors. [**NB**: Of course, a *direct* proof of **SN[λ!]** is also possible. Adapt, e. g., any one of the **SN**-proofs appearing in [van Daalen 1980], to the case in point.]. □

9.24 Remark.

> So, in a sense, **λ!** is *not more* than the ordinary typed λ-calculus, in its "extensional" version, i. e., the Curry "functionality theory". Actually, **λ!** would essentially correspond, after formalization, to the so-called "theory of generalized functionality" of H. B. Curry and J. P. Seldin ([Seldin 1979], [Hindley & Seldin 1986], **16**, [Seldin 1987].)

The situation described in the above does not change too much for ξ-extensions of **λ(!)**. In other words, one retrieves the same "level of **SN**-proof-complexity".

We show this next, by exploiting a well-known "provability-only" result for **HQ** (due, in part, to Kolmogorov [1925], Glivenko [1928] for **H** and to Gödel [1933], Gentzen [1933], [Kuroda 1951] for **HQ**; cf. also [Löb 1976]).

> *Note.* Specifically, this is a consequence of the familiar "double-negation" translation(s) $\neg\neg$: **CQ** ⟶ **HQ** and says that **CQ** ⊩ A <==> **HQ** ⊩ A, for all "negative" formulas/propositions A in $L[\supset,\forall]$. (A is "negative" iff all propositional atoms P of A occur only "negated" in A, i. e., in sub-formulas of the form $P \supset \bot$.) For "provability-only" details, see, e. g., the recent survey of [Troelstra & van Dalen 1988], **I: Chapter 2, 3.6** and **I: Chapter 2, 8.5**, for a historical comment.

10. "Lifting" and hyper-reductions. In proofs of **SN**[$\lambda\gamma(!)$], it is - technically - more convenient an approach to manipulate slightly stronger notions of reduction than the standard ones.

Such notions are obtained by "lifting" $\beta\gamma$-détours *via* extensionality assumptions. (Cf. section **7** for an analogous strategy, as applied to $\beta\gamma$-conversions.)

Hyper-$\beta\gamma$-reductions, type-normality. In the sequel, we isolate the "complex" $\beta\gamma$-notions [$\beta\gamma\supset$], [$\beta\gamma\forall$] from the remaining ones and "lift them up" to appropriate *hyper-notions of $\beta\gamma$-reduction* [$\vdash\beta\gamma\supset$], [$\vdash\beta\gamma\forall$], resp.

The "lifted" notions are intended to eliminate "complex" γ-détours from p-terms no matter whether they occur or not in particular subterm-environments which would make them "ready-for-evaluation".

For the class of $\lambda\gamma(!)$-calculi of concern here, the hyper-notions are as follows:

10.1 Definition.

Basic hyper-$\beta\gamma$-reductions.

$$(\vdash\beta\gamma\supset):\quad \frac{\Phi[\ x\ :\ (A\supset B)^-\ ]\ \vdash\ c[\![x]\!]\ :\ \bot}{\Phi\ \vdash\ \gamma x{:}(A\supset B)^-.c[\![x]\!]\ \Rightarrow\!\!>\ \lambda\mathbf{x_0}{:}A.\,\gamma\mathbf{x_1}{:}B^-.c'[\![\mathbf{x_0},\mathbf{x_1}]\!]\ [:\ A\supset B]},$$

where $c'[\![\mathbf{x_0},\mathbf{x_1}]\!] \equiv c[\![x:=\lambda z{:}(A\supset B).\mathbf{x_1}(z\mathbf{x_0})]\!]$.

First-order hyper-$\beta\gamma$-reduction. If u is not free in c[[x]],

$$(\vdash\beta\gamma\forall):\quad \frac{\Phi[\ x\ :\ (\forall u.A[\![u]\!])^-\ ]\ \vdash\ c[\![x]\!]\ :\ \bot}{\Phi\ \vdash\ \gamma x{:}(\forall u.A[\![u]\!])^-.c[\![x]\!]\ \Rightarrow\!\!>\ !u.\,\gamma\mathbf{x_1}{:}A^-[\![u]\!].c'[\![u,\mathbf{x_1}]\!]\ [:\ \forall u.A[\![u]\!]\ ]},$$

where $c'[\![u,\mathbf{x_1}]\!] \equiv c[\![x:=\lambda z{:}(\forall v.A[\![v]\!]).\mathbf{x_1}(z[u])]\!]$.

10.2 Definition.

$\lambda^{\vdash}\gamma(!)$ is defined as $\lambda\gamma(!)$, except for the fact that the primitive (contraction, reduction, etc.) rules ($\beta\gamma\supset$), ($\beta\gamma\forall$) are replaced by the "lifted" variants ($\vdash\beta\gamma\supset$), ($\vdash\beta\gamma\forall$) resp.

10.3 Remark.

Note that the *standard notion of reduction of* $\lambda^{\vdash}\gamma(!)$ is $\lambda^{\vdash}\gamma(!) = [\beta\supset\lambda][\eta\supset\lambda][^{\vdash}\beta\gamma][\eta\supset\gamma]([\beta\forall][\eta\forall])$, where $[^{\vdash}\beta\gamma]$ stands for $[\beta\gamma\mathbf{T}][\beta\gamma\mathbf{\bot}][\vdash\beta\gamma\supset][\vdash\beta\gamma\forall]$. This should not be confused with the "pure" hyper-notion $[\vdash\beta\gamma\supset][\vdash\beta\gamma\forall]$.

10.4 Notation.

$[\vdash\beta\mathfrak{g}] := [\vdash\beta\mathfrak{g}\sqsupset][\vdash\beta\mathfrak{g}\forall]$ is *the standard "pure" hyper-notion of* $\beta\mathfrak{g}$*-reduction for (relative to)* $L[\sqsupset,\forall]$ *in* $\lambda^{\vdash}\mathfrak{g}(!)$.

10.5 Definition.

(1) A p-term a is *type-normal* if every subterm of a which is a $\mathfrak{g}$-abstract $\mathfrak{g}x{:}A^{-}.a[\![x]\!]$ is such that A is *atomic*, viz., either a propositional constant ($\top$ or $\bot$) or an atomic proposition $P[u_1,\ldots u_n]$.

(2) The *type-normal fragment* of $\lambda\mathfrak{g}(!)$ - denoted hereafter by ${}^{TN}\lambda\mathfrak{g}(!)$ - is the restriction of $\lambda\mathfrak{g}(!)$ to type-normal p-terms.

10.6 Notation.

$\mathrm{Term}[{}^{TN}\lambda\mathfrak{g}(!)]$ is *the set of type-normal p-terms in* $\lambda^{\vdash}\mathfrak{g}(!)$.

10.7 Remark.

Certainly, $\mathrm{Term}[{}^{TN}\lambda\mathfrak{g}(!)]$ is *the set of type-normal p-terms in* $\lambda\mathfrak{g}(!)$, as well, and ${}^{TN}\lambda\mathfrak{g}(!)$ is the same thing as the restriction of $\lambda^{\vdash}\mathfrak{g}(!)$ to type-normal p-terms. One should, however, note that ${}^{TN}\lambda\mathfrak{g}(!)$ has a standard notion of reduction, given by $\lambda\mathfrak{g}(!) - (\beta\mathfrak{g}\sqsupset][\beta\mathfrak{g}\forall])$ (equivalently, by ${}^{\vdash}\lambda\mathfrak{g}(!) - [\vdash\beta\mathfrak{g}]$, of course!). That is, ${}^{TN}\lambda\mathfrak{g}(!)$ can be obtained from $\lambda\mathfrak{g}(!)$ [or from $\lambda^{\vdash}\mathfrak{g}(!)$, for that matter], by restricting the set of (well-formed and "correct") p-terms to $\mathrm{Term}[{}^{TN}\lambda\mathfrak{g}(!)]$ and by leaving out the rules $(\beta\mathfrak{g}\sqsupset)$ and $(\beta\mathfrak{g}\forall)$ [resp. the "hyper-rules" $(\vdash\beta\mathfrak{g}\sqsupset)$ and $(\vdash\beta\mathfrak{g}\forall)$], that would have, anyway, no effect in a type-normal environment.

Clearly then, the "pure" hyper-notion $[\vdash\beta\mathfrak{g}]$ is meant to reduce p-terms to type-normal forms, *relative to the provability language* $L[\sqsupset,\forall]$.

10.8 Remark.

The relativization-*proviso* points out to the fact that, for a more complex primitive setting (as regards the underlying provability language; in presence of a primitive &, for instance), an analogous "lifting" should be performed on the remaining $\beta\mathfrak{g}$-notions of the associated proof-language.

10.9 Remark.

Let $\vdash$ stand ambiguously for derivability in $\lambda\mathfrak{g}(!)$ and let $\vdash_H$ stand for the analogous concept in $\lambda^{\vdash}\mathfrak{g}(!)$, whereas $\Rightarrow$ and ${}_H\Rightarrow$ are the associated standard reduction-relations. Then for all a, b $\in \mathrm{Term}[\lambda\mathfrak{g}(!)]$,

$$\Phi \vdash a \Rightarrow b \quad ==> \quad \Phi \vdash_H a \ {}_H\Rightarrow b.$$

This follows immediately, since the rules $(\beta\not\sigma\sqsupset)$ and $(\beta\not\sigma\forall)$ resp. are derivable in $\lambda^{\vdash}\not\sigma!$. Moreover, if $=$ and $=_H$ are the associated equality (conversion) relations then an analogous statement can be shown to hold for $=$ and $=_H$ and this can be even strengthened up to an equivalence

$$\Phi \vdash a = b \iff \Phi \vdash_H a =_H b,$$

for all $a, b \in \mathrm{Term}[\lambda\not\sigma(!)]$. Here, the $\Leftarrow$-part of the statement can be obtained by appropriate uses of extensionality properties of $=$ in the $\not\sigma$-free fragment of $\lambda\not\sigma(!)$.

10.10 Lemma.

$$\mathbf{SN}[\lambda^{\vdash}\not\sigma(!)] \Rightarrow \mathbf{SN}[\lambda\not\sigma(!)].$$

Proof. By **10.9**, viz., the part applying to reduction relations. □

10.11 Remark.

Actually, if we decide to translate entire reduction-sequences from $\lambda\not\sigma(!)$ into $\lambda^{\vdash}\not\sigma(!)$, then the applications of the rules $(\beta\not\sigma\sqsupset)$ and $(\beta\not\sigma\forall)$ resp. [in $\lambda\not\sigma(!)$] become instances of $(\beta\sqsupset\lambda)$ and $(\beta\forall)$ resp. [in $\lambda^{\vdash}\not\sigma(!)$]! This amounts to the fact that, for each p-term a, with $\Phi \vdash a : A$ [in $\lambda\not\sigma(!)$], and every path p in a $\Sigma[\lambda\not\sigma(!), a]$, there is a uniquely determined $p_{\vdash}$ in the tree $\Sigma[\lambda^{\vdash}\not\sigma(!), a]$, such that

(1°) every détour that is not a $(\beta\not\sigma\sqsupset)$- or a $(\beta\not\sigma\forall)$-détour is eliminated exactly in the same way in both $\lambda\not\sigma(!)$ and $\lambda^{\vdash}\not\sigma(!)$ and such that

(2°1) $(\beta\not\sigma\sqsupset)$-détour-eliminations [in $\lambda\not\sigma(!)$] are replaced by corresponding $(\beta\sqsupset\lambda)$-détour-eliminations [in $\lambda^{\vdash}\not\sigma(!)$] and

(2°2) $(\beta\not\sigma\forall)$-détour-eliminations [in $\lambda\not\sigma(!)$] are replaced by corresponding $(\beta\forall)$-détour-eliminations [in $\lambda^{\vdash}\not\sigma(!)$].

Moreover, the form of the détours involved in translation can be established explicitly [*exercise*].

We show next that the "pure" hyper-$\beta\not\sigma$-reductions are well-behaved.

Strong normalization for hyper-$\beta\not\sigma$-reductions. Intuitively, it is already clear that hyper-$\beta\not\sigma$-reductions decrease the complexity of the $\not\sigma$-détours. So, the following fact is as expected:

10.12 Lemma.

$$\mathbf{SN}[\lambda^{\vdash}\not\sigma(!), [^{\vdash}\beta\not\sigma]].$$

Proof. For $\Phi \vdash c : C$ in $\lambda^{\vdash}\mathfrak{g}(!)$, let $\mathbf{h}(c)$, *the height of* c, be defined inductively, as follows:

(0) if a contains no $\mathfrak{g}$-abstract as a subterm then $\mathbf{h}(c) = 0$;

else

(1) if $c \equiv ab$ then $\mathbf{h}(c) = \mathbf{h}(a) + \mathbf{h}(b)$,
(2) if $c \equiv \lambda x{:}A.a$ or
(3) if $c \equiv a[t]$ or
(4) if $c \equiv !u.a$ then $\mathbf{h}(c) = \mathbf{h}(a)$,
(5) if $c \equiv \mathfrak{g}x{:}C^{-}.a[\![x]\!]$ and
(51) if $C \equiv \top$, $\bot$ or $P[u_1,\ldots u_n]$) then $\mathbf{h}(c) = \mathbf{h}(a) + 1$,
(52) if $C \equiv A \sqsupset B$ then $h = 10 \times \mathbf{h}(c)$, where
$c \equiv \lambda \mathbf{x_0}{:}A.\ \mathfrak{g}\mathbf{x_1}{:}B^{-}.a[\![x:=\lambda z{:}(A \sqsupset B).\mathbf{x_1}(z\mathbf{x_0})]\!]$,
(53) if $C \equiv \forall u.A[\![u]\!]$ then $h = 2^{h<c>}$, where
$c \equiv !u.\ \mathfrak{g}\mathbf{x_1}{:}A^{-}[\![v:=u]\!].a[\![x:=\lambda z{:}(\forall v.A[\![v]\!]).\mathbf{x_1}(z[u])]\!]$,

Let $\mathbf{H} = [\vdash\beta\mathfrak{g}]$, for short. Then, one can check immediately that the corresponding one-step H-reductions are such that

$$\Phi \vdash a \ {}_{H}\!\!-\!\!\gg_1 b \Longrightarrow \mathbf{h}(a) > \mathbf{h}(b),$$

i. e., one-step H-reductions reduce strictly the complexity of p-terms, inasfar **h** is concerned.

Eventually, on every hyper-$\beta\mathfrak{g}$-reduction path ρ from a, in the tree $\Sigma[[\vdash\beta\mathfrak{g}],a]$, all subterms c of a that are also hyper-$\beta\mathfrak{g}$-redexes, i. e., of the form $c \equiv \mathfrak{g}x{:}C^{-}.a[\![x]\!]$, with $C \equiv A \sqsupset B$ and/or $C \equiv \forall u.A$ are replaced by p-terms containing as subterms only $\mathfrak{g}$-abstracts $c' \equiv \mathfrak{g}x{:}D^{-}.a'[\![x]\!]$, where D is less complex than C (e. g., C has either at least an extra $\sqsupset$ or at least an extra $\forall$). Clearly then, on each branch $\rho_{H,\infty} \in \Sigma[[\vdash\beta\mathfrak{g}],a]$ the process stops with a type-normal form. □

This shows that *type-normal forms do always exist* (as limit points of $\vdash\beta\mathfrak{g}$-reduction paths in $\vdash\beta\mathfrak{g}$-reduction spectra of p-terms)!

Confluence for hyper-$\beta\mathfrak{g}$-reductions. In order to insure *unicity of type-normal forms*, we prove first a stronger result, viz. that *hyper-$\beta\mathfrak{g}$-reductions are confluent.*

This is shown by the Rosser/Tait/Martin-Löf "residuation" method. Using the general proof-pattern, we need a "parallel" version, $\mathbf{\pi H} := {}^{\pi}\vdash\beta\mathfrak{g}$, say, of $\mathbf{H} := \vdash\beta\mathfrak{g}$, viz. a (formal) relation ${}_{H}\!\!\rightarrowtail$ on $\mathrm{Term}[\lambda\mathfrak{g}(!)]$ such that ${}_{H}\!\!\rightarrowtail$ and the standard notion ${}_{H}\!\!-\!\!\gg$ of hyper-$\beta\mathfrak{g}$-contraction have the same transitive closure (here: ${}_{H}\!\!=\!\!\gg$).

The required result follows then by the technical **Lemma 9.16**, on transitive closures, as relativized to $\lambda\mathfrak{g}(!)$, viz.:

> If **r** and ${}^{\pi}\mathbf{r}$ have the same transitive closure as relations on $\mathrm{Term}[\lambda\mathfrak{g}(!)]$, then $\mathbf{CR}[\lambda\mathfrak{g}(!),{}^{\pi}\mathbf{r}] \Rightarrow \mathbf{CR}[\lambda\mathfrak{g}(!),\mathbf{r}]$.

Formally, we equip $\lambda^{\langle h \rangle}\partial(!)$ with a relation $\twoheadrightarrow_H = {}^{=h}\beta\partial$ on the set Term[$\lambda\partial(!)$] and use the resulting formal system, $\lambda^{=h}\beta\partial(!) := (\lambda\partial(!), {}^{=h}\beta\partial)$ as a technical tool to prove CR[$\lambda^{\langle h \rangle}\partial(!), {}^{=h}\beta\partial$].

The formal system $\lambda^{=h}\beta\partial(!)$ is given by the following rules:

10.13 Definition.

($^{=h}\rho$): *Reflexivity*: $\Phi \vdash a : A \Longrightarrow \Phi \vdash a \twoheadrightarrow_H a : A$.

($^{=h}\beta\partial\sqsupset$): *Basic parallel hyper-$\beta\partial$-reduction.*

$$\frac{\Phi[\, x : (A \sqsupset B)^- \,] \vdash c_1[\![x]\!] \twoheadrightarrow_H c_2[\![x]\!] : \bot}{\Phi \vdash \partial x{:}(A \sqsupset B)^-.c_1[\![x]\!] \twoheadrightarrow_H \lambda x_0{:}A.\, \partial x_1{:}B^-.c'[\![x_0, x_1]\!]\ [:\ A \sqsupset B]},$$

where $c'[\![x_0, x_1]\!] \equiv c_2[\![x := \lambda z{:}(A \sqsupset B).x_1(zx_0)]\!]$.

($^{=h}\beta\partial\forall$): *First-order hyper-$\beta\partial$-reduction.* If u is not in $FV_U(c_i[\![x]\!])$, $(i := 1,2)$,

$$\frac{\Phi[\, x : (\forall u.A[\![u]\!])^- \,] \vdash c_1[\![x]\!] \twoheadrightarrow_H c_2[\![x]\!] : \bot}{\Phi \vdash \partial x{:}(\forall u.A[\![u]\!])^-.c_1[\![x]\!] \twoheadrightarrow_H !u.\, \partial x_1{:}A^-[\![u]\!].c'[\![u, x_1]\!]\ [:\ \forall u.A[\![u]\!]]},$$

where $c'[\![u, x_1]\!] \equiv c_2[\![x := \lambda z{:}(\forall v.A[\![v]\!]).x_1(z[u])]\!]$.

Compatibility.

$$(^{=h}\mu\nu\sqsupset):\ \frac{\Phi_1 \vdash f \twoheadrightarrow_H g : A \sqsupset B \qquad \Phi_2 \vdash a_1 \twoheadrightarrow_H a_2 : A}{\Phi_1\Phi_2 \vdash fa_1 \twoheadrightarrow_H ga_2\ [:\ B]},$$

$$(^{=h}\xi\sqsupset\lambda):\ \frac{\Phi[\, x : A \,] \vdash a[\![x]\!] \twoheadrightarrow_H b[\![x]\!] : B}{\Phi \vdash \lambda x{:}A.a[\![x]\!] \twoheadrightarrow_H \lambda x{:}A.b[\![x]\!] : [A \sqsupset B]},$$

$$(^{=h}\xi\sqsupset\partial):\ \frac{\Phi[\, x : A^- \,] \vdash a[\![x]\!] \twoheadrightarrow_H b[\![x]\!] : \bot}{\Phi \vdash \partial x{:}A^-.a[\![x]\!] \twoheadrightarrow_H \partial x{:}A^-.b[\![x]\!]\ [:\ A]},$$

$$(^{=h}\mu\forall):\ \frac{\Phi_1 \Vdash t :: U \qquad \Phi_2 \vdash f \twoheadrightarrow_H g : \forall u.A[\![u]\!]}{\Phi_1\Phi_2 \vdash f[t] \twoheadrightarrow_H g[t]\ [:\ A[\![u := t]\!]]},\ [u \text{ not free in } f, g]$$

$$(^{=h}\xi\forall):\ \frac{\Phi[\, u :: U \,] \vdash a[\![u]\!] \twoheadrightarrow_H b[\![u]\!] : A[\![u]\!]}{\Phi \vdash !u.a[\![u]\!] \twoheadrightarrow_H !u.b[\![u]\!]\ [:\ \forall u.A]}.$$

10.14 Definition.

For each p-term $c \in \mathrm{Term}[\lambda\eth(!)]$, the p-term c^{hr} is defined by:

(0) if $c \equiv x$ then $c^{hr} \equiv x$,
(1) if $c \equiv ab$ then $c^{hr} \equiv (a^{hr})(b^{hr})$,
(2) if $c \equiv \lambda x{:}A.a$ then $c^{hr} \equiv \lambda x{:}A.a^{hr}$,
(3) if $c \equiv a[t]$ then $c^{hr} \equiv (a^{hr})[t]$,
(4) if $c \equiv !u.a$ then $c^{hr} \equiv !u.a^{hr}$,
(5) if $c \equiv \eth x{:}A^{-}.a[\![x]\!]$ and
(51) $A \equiv \top$, $\bot$ or $P[u_1,\ldots,u_n]$ then $c^{hr} \equiv \eth x{:}A^{-}.a^{hr}$,
(52) if $C \equiv A \supset B$ then $c^{hr} \equiv \lambda \mathbf{x}_0{:}A.\,\eth \mathbf{x}_1{:}B^{-}.a'$
where $a' \equiv a^{hr}[\![x:=\lambda z{:}(A\supset B).\mathbf{x}_1(z\mathbf{x}_0)]\!]$,
(53) if $C \equiv \forall u.A[\![u]\!]$ then $c^{hr} \equiv !u.\,\eth \mathbf{x}_1{:}A^{-}[\![v:=u]\!].a'$,
where $a' \equiv a^{hr}[\![x:=\lambda z{:}(\forall v.A[\![v]\!]).\mathbf{x}_1(z[u])]\!]$.

10.15 Lemma (*Hyper-$\beta\eth$-residual inversion lemma*).

For all p-terms a, b $\in \mathrm{Term}[\lambda\eth(!)]$, $a \;{}_H\!\rightarrowtail b \Longrightarrow b \;{}_H\!\rightarrowtail a^{hr}$.

Proof. By induction on the [subterm-] structure of a. □

10.16 Theorem (*Confluence for parallel hyper-$\beta\eth$-reductions*).

CR$[\lambda^{h}\eth(!), {}^{=h}\beta\eth]$.

Proof. From **10.15**. □

10.17 Remark.

The relations ${}_H\!\rightarrowtail$ and ${}_H\!\twoheadrightarrow$ have the same transitive closure (on $\mathrm{Term}[\lambda\eth(!)]$), viz. ${}_H\!=\!\twoheadrightarrow$.

10.18 Theorem (*Confluence for hyper-$\beta\eth$-reductions*).

CR$[\lambda^{h}\eth(!), {}^{h}\beta\eth]$.

Proof. From **CR**$[\lambda^{h}\eth(!), {}^{=h}\beta\eth]$ [**10.16**], by **10.17** and **9.16**. □

The main consequences of this are, as expected:

10.19 Corollary (*The "Church-Rosser" theorem for ${}^{h}\beta\eth$-equality*).

Where ${}_H\!=$ and ${}_H\!\twoheadrightarrow$ are ${}^{h}\beta\eth$-*equality* and hyper-$\beta\eth$-reduction on $\mathrm{Term}[\lambda\eth(!)]$, for all p-terms a_1, $a_2 \in \mathrm{Term}[\lambda\eth(!)]$,

$$\lambda^{h}\eth(!) \vdash a_1 \;{}_H\!= a_2 \Longrightarrow \lambda^{h}\eth(!) \vdash a_1 \;{}_H\!\twoheadrightarrow c,$$

for some $c \in \mathrm{Term}[\lambda\eth(!)]$.

Proof. By tiling the plane, from **CR**$[\lambda^{h}\eth(!), {}^{h}\beta\eth]$, i. e., **10.18**. □

10.20 Theorem (*Unicity of type-normal forms*).

UN[$\lambda^{\vdash}\beta(!)$,$^{\vdash}\beta$], i. e., type-normal forms are unique up to $\equiv_{\alpha}$.

Proof. From the "Church-Rosser" theorem for $^{\vdash}\beta$-equality **10.19**. □

10.21 Corollary (*Non-triviality of* $^{\vdash}\beta$*-equality*).

Cons[$\lambda^{\vdash}\beta(!)$]; i. e., hyper-β-conversion is not trivial.

Proof. From **10.20**. □

10.22 Notation.

In view of **10.20**, we can write $^{tn}(a) \equiv$ "**the** type-normal form of a", for every $a \in$ Term[$\lambda\beta(!)$].

10.23 Remark.

In the end, the intuitive meaning of the (joint) property

SN[$\lambda^{\vdash}\beta(!)$,$^{\vdash}\beta$] **&** **CR**[$\lambda^{\vdash}\beta(!)$,$^{\vdash}\beta$] **&** **UN**[$\lambda^{\vdash}\beta(!)$,$^{\vdash}\beta$]

is that hyper-β-reductions/expansions amount to a *systematic abbreviation technique*, applicable to the type-normal fragment $^{TN}\lambda\beta(!)$ of $\lambda^{\langle\vdash\rangle}\beta(!)$, where the $^{\vdash}$-superscripting looses its proper meaning. In particular, translations from $\lambda^{\vdash}\beta(!)$ to $^{TN}\lambda\beta(!)$ respecting hyper-β-reductions should be also "rigid" (or deterministic, in nature) and define a "faithful" surjection $^{tn} : \lambda^{\vdash}\beta(!) \dashrightarrow {}^{TN}\lambda\beta(!)$, say, preserving $^{TN}\lambda\beta(!)$, in the sense that $^{tn}(\lambda^{\vdash}\beta(!)) = {}^{TN}\lambda\beta(!)$. This fact will be implicitly exploited next.

11. The negative translation $^N : \lambda^{\vdash}\mathcal{G}(!) \longrightarrow \lambda(!)$. The official "typing" statements (formally: the *t-statements*) of $\lambda^{\langle\vdash\rangle}\mathcal{G}(!)$ are triples of the form $\varphi \equiv (\Phi, a, A)$.

The fact that φ "holds" or that φ "is derivable" in $\lambda^{\langle\vdash\rangle}\mathcal{G}(!)$, i. e., that $\lambda^{\langle\vdash\rangle}\mathcal{G}(!) \vdash \varphi$ is, usually, written as $\Phi \vdash a : A$, understanding implicitly that the turnstile "$\vdash$" bears a (hidden) "relativization"-subscript "$\lambda^{\langle\vdash\rangle}\mathcal{G}(!)$".

So, alternatively, the calculus $\lambda\mathcal{G}!$ can be viewed as *a set of t-statements* of the form $\varphi \equiv (\Phi, a, A)$ such that $\lambda\mathcal{G}! \vdash \varphi$.

Conservativity of t-statements over $\mathcal{G}$-free fragments. We note first a conservativity property of the proof-syntax of $\lambda\mathcal{G}(!)$-calculus.

11.1 Notation.

Let "$\vdash$", "$=\!\gg$" and "$=$" resp. stand for derivability, standard reduction and standard conversion (or equality) resp., in $\lambda^{\langle\vdash\rangle}\mathcal{G}(!)$, whereas "$\vdash_{\lambda^{\langle!\rangle}}$", $\lambda^{\langle!\rangle}\!=\!\gg$, $\lambda^{\langle!\rangle}\!=$ resp. denote the analogous notions in the corresponding $\mathcal{G}$-free fragment(s) $\lambda(!)$. Note also that "$\vdash$" has the same meaning in $\lambda^{\vdash}\mathcal{G}(!)$ and $\lambda\mathcal{G}(!)$. But, if necessary to distinguish betweeen "$=\!\gg$", "$=$" resp. in $\lambda\mathcal{G}(!)$ and the their "hyper"-counterparts in $\lambda^{\vdash}\mathcal{G}(!)$, the latter will bear a subscript "**H**", say.

11.2 Theorem (*Conservativity of $\vdash$ over $\vdash_{\lambda^{\langle!\rangle}}$*).

For all p-terms a, b $\in$ Term[$\lambda\mathcal{G}(!)$] and all contexts Φ,

$$(\Phi \vdash a : A) \;\&\; (a \text{ is } \mathcal{G}\text{-free}) \Longrightarrow \Phi \vdash_{\lambda^{\langle!\rangle}} a : A.$$

Proof. By induction on the structure of a, using the appropriate "subterm correctness" properties [3.15]. □

The negative translation: heuristics. We define next a translation $^N : \lambda^{\vdash}\mathcal{G}(!) \dashrightarrow \lambda(!)$, called *negative translation*, from t-statements of $\lambda^{\vdash}\mathcal{G}(!)$ to ($\mathcal{G}$-free) t-statements of $\lambda(!)$, such that ^{N}

(1°) preserves proper $\lambda(!)$-détours ("faithfully") and

(2°) translates proper $\mathcal{G}$-détours into appropriate $\lambda(!)$-détours.

The negative translation won't, however, preserve $\lambda^{\langle\vdash\rangle}\mathcal{G}(!)$-typings.

Actually, we shall see that $\lambda^{\vdash}\mathcal{G}(!) \vdash \varphi \Longrightarrow \lambda(!) \vdash {}^N(\varphi)$, for any "typing" statement φ of $\lambda^{\vdash}\mathcal{G}(!)$, i. e., where $\varphi \equiv (\Phi, a, A)$, the "negative" image $\varphi^N \equiv (\Phi^N, a^N, A^N)$ of φ, for $\Phi \vdash a : A$, is such that $\Phi^N \vdash a^N : A^N$ [in $\lambda^{\vdash}\mathcal{G}(!)$ and, therefore, in $\lambda(!)$ by the Conservativity Theorem 11.2], although A and A^N are, in general, (syntactically) distinct propositions/types.

The relevant information is, in this case, extracted from the fact that, with notation as above, one has

$$\Phi \vdash a \Rightarrow b \Longrightarrow \Phi^{N} \vdash_{\lambda^{<!>}} a^{N}\ {}_{\lambda^{<!>}}\!\Rightarrow b^{N},$$

for all p-terms a, b ∈ Term[λ𝒥(!)].

11.3 Remark.

This is the essence of the "double-negation interpretation" of Kolmogorov-Glivenko-Gödel-Gentzen-Kuroda, as expressed for a proper proof-setting and abstracting from any irrelevant information. λ! is the *same thing* as the proof-part of he [⊃,∀]-fragment of Heyting's first-order logic **HQ**, but this is completely irrelevant for proof-theoretic purposes.

11.4 Convention (local).

For the rest of this section we assume that the symbol ⊤ is eliminated uniformly from propositions/types of $\mathcal{L}[\supset,\forall]$ and the λ(𝒥)(!)-syntax, by setting $\top \equiv [\bot \supset \bot]$.

[**NB:** The present convention is part of the definition of $^{N}(\ldots)$ below. Otherwise, we keep referring to $\mathcal{L}[\supset,\forall]$ as the *underlying provability language* of λ𝒥(!).]

11.5 Notation.

Let $\mathbf{u} := u_{m,1},\ldots,u_{m,n}$ be (possibly empty) sequences of **U**-parameters and $P_j \equiv P_j[\mathbf{u}]$ $(1 \leq j \leq m)$ denote the *proper* atomic propositions/types in $\mathcal{L}[\supset,\forall]$. Here, in the limit case, the propositional variables are the P_j's with **u** empty. Further, in order to avoid ambiguities, the propositional atomic constants ⊤ and ⊥ will not count as (proper) atomic propositions.

11.6 Remark.

One can define, somewhat redundantly, three *simultaneous substitution operators* (on contexts, propositions/types and p-terms), all denoted by "$\ldots[\![\ldots_j:=\ldots_j\ldots]\!]_{j<m}$" in an ambiguous way, in what follows.

(1) If $\Phi = [x_i:A_i]_{i<n}$ is a context with $\Phi \vdash \varphi$, for some φ, in $\lambda^{<!>}(!)$, and if P_j $(1 \leq j \leq m)$ are the atomic propositions/types occurrig in the A_i's, then $\Phi^{\sim}$ is the context $[x_i:B_i]_{i<n}$, where $B_i \equiv A_i[\![P_j := (P_j \supset \bot)]\!]_{j<m}$, for all $1 \leq i \leq n$.

(2) If A is a proposition/type in $\mathcal{L}[\supset,\forall]$ such that the atoms P_j $(1 \leq j \leq m)$ are all the atomic propositions/types occurrig in A, then $A^{\sim}$ is the proposition/type $A[\![P_j := (P_j \supset \bot)]\!]_{j<m}$.

(3) If a is a p-term in Term[$\lambda^{(h)}\forall(!)$] and the atoms P_j ($1 \leq j \leq m$) are all atomic propositions/types occurrig in a, then $a^{\sim}$ is the p-term $a[\![P_j := (P_j \supset \bot)]\!]_{j<m}$.

One can make this slightly more general, using a *single type of substitution operators*. Before doing this, note that

11.7 Lemma.

If $\Phi \vdash a : A$ (where Φ is possibly empty), and if **P**(a), **P**(A) resp. are the sequences of atomic propositions/types occurrig in a and A resp. then $\mathbf{P}(a) \subseteq \mathbf{P}(A)$.

Proof. By induction on the derivation of $\Phi \vdash a : A$. Obviously, only the rules ($\supset$iλ) and ($\supset$i$\forall$) do actually matter here. □

11.8 Notation.

Let $\Phi \vdash a : A$, with $\Phi \equiv [x_1{:}A_1, \ldots, x_n{:}A_n]$ possibly empty, $\mathbf{P} = \langle P_k \rangle_{k<m}$ be the sequence of atomic propositions/types occurrig in Φ and $A_1, \ldots, A_n$. Let also, for convenience, **sub**(...) stand for the (simultaneous) substitution

$$\ldots[\![P_j := (P_j \supset \bot)]\!]_{j<m}.$$

Then, clearly, $\mathbf{sub}(\Phi) \vdash \mathbf{sub}(a) : \mathbf{sub}(A)$, by the appropriate substitution rule (**\$**).

11.9 Remark.

Assume $\Phi \vdash a : A$, with $\Phi \equiv [x_1{:}A_1, \ldots, x_n{:}A_n]$ possibly empty.

(1) If $\mathbf{P}(\Phi)$ is the sequence $(P_i)_{i<p}$ of atomic types/propositions occurrig actually in $A_1, \ldots, A_n$, and $\Phi' \equiv [x_1{:}B_1, \ldots, x{:}B_n]$, where $B_i \equiv A_i[\![P_j := (P_j \supset \bot)]\!]_{j<p}$, for all $1 \leq i \leq n$, then $\Phi^{\sim} \equiv \mathbf{sub}(\Phi)$, as well.

(2) Analogously, if A is a type/proposition, with **P**(A) the sequence $(P_j)_{j<q}$ of atomic types/propositions occurrig actually in A, and $A' \equiv A[\![P_j := (P_j \supset \bot)]\!]_{j<q}$, then also $A' \equiv A^{\sim} \equiv \mathbf{sub}(A)$.

(3) In general, this notational policy allows to have, for all p-terms a, all contexts Φ and all types/propositions A,

$$\Phi \vdash a : A \Longrightarrow \mathbf{sub}(\Phi) \equiv \Phi^{\sim} \vdash \mathbf{sub}(a) \equiv a^{\sim} : \mathbf{sub}(A) \equiv A^{\sim},$$

since, if **P**(a) is the sequence of atomic types/propositions occurrig actually in a, one has always $\mathbf{P}(a) \subseteq \mathbf{P}(A)$, by **11.7**.

Negative atomic propositions. The types/propositions A of $\mathcal{L}[\supset,\forall]$ can occur actually in p-terms $a \in Term[\lambda\!\!\not\!(!)]$ either

(1°) within the scope of λ-abstractions, i. e., in sub-terms $c \equiv \lambda x{:}A.a[\![x]\!]$ or

(2°) within the scope of ∌-abstractions, in sub-terms of the form $c \equiv \not\!x{:}A^{-}.a[\![x]\!]$.

11.10 Definition.

(1) For any type/proposition $A \equiv [C \supset \bot]$, an *actual* occurrence P of an atomic proposition is said to be *positive/negative in* A, according to the following inductive clauses:

(1°) P is negative in $P \supset \bot$ [**NB**: P is atomic!],
(2°) P is negative in $A[\![P]\!] \supset B[\![P]\!]$ iff it is negative in $B[\![P]\!]$,
(3°) P is negative in $\forall u.A[\![P]\!][\![u]\!]$ iff it is negative in $A[\![P]\!][\![u]\!]$,
(4°) else, P is positive in A.

(2) An occurrence of an atomic type/proposition P in a p-term a is *negative in* a if it negative in some type/proposition $A^{-} \equiv A \supset \bot$, occurring within the scope of a ∌-abstraction, in a subterm c of a, of the form $c \equiv \not\!x{:}A^{-}.a[\![x]\!]$, else P is said to be *positive in* a.

11.11 Remark.

(1) Let P be an atomic type/proposition. If P has a negative occurrence $\underline{P}$ in A and $A' \equiv A[\![P{:=}(P\supset\bot)]\!]$ such that $\underline{P}$ goes into $[\underline{P} \supset \bot]$ in A', then the corresponding occurrence of P in A' is positive in A'. (Note that $[\![\ldots{:=}\ldots]\!]$ can be understood here either as a substitution operator or as a replacement operation, applying to the indicated token-occurence $\underline{P}$ of P. The statement holds in both cases.)

(2) Let A and P be as above. If P has a negative occurrence $\underline{P}$ in A then the matching occurrence $\underline{P}$ of P in $A^{\sim}$ is positive in $A^{\sim}$.

Proof. (1) By induction on the structure of A. (2) From (1) and the definition of $(\ldots)^{\sim}$. □

11.12 Lemma.

For all p-terms $a \in Term[\lambda\!\!\not\!(!)]$ and all contexts Φ,

$$\Phi \vdash a \Longrightarrow a^{\sim} \text{ contains no negative atoms.}$$

Proof. The hypothesis $\Phi \vdash a$ (i. e., "$\Phi \vdash a : A$, for some A") is meant to insure that a is "correctly typed". Note first that the substitution $(...)^\sim$ does not introduce new negative atoms.

Let $\underline{P}$ be a given negative occurrence of an atomic type/proposition P in a. So, there is a subterm $d \equiv \mathfrak{d}x{:}A^-.b[\![x]\!]$ of a, such that P occurs negatively in A^-. By induction on the structure of A, one has:

(1°) A is atomic, so $A \equiv \underline{P}$ (the cases $A \equiv \top$, $A \equiv \bot$ are excluded by assumption). Then $d' \equiv (d)^\sim \equiv \mathfrak{d}x{:}\underline{P}^-.(b[\![x]\!])^\sim$ and this very occurrence of P in d' (within the scope of $\mathfrak{d}$) is now positive in d'.

(2°) $A \equiv [B \supset C]$, so P occurs negatively in C. Then $d' \equiv (d)^\sim \equiv \mathfrak{d}x{:}(B \supset C)^\sim.(b[\![x]\!])^\sim \equiv \mathfrak{d}x{:}((B)^\sim \supset (C)^\sim).(b[\![x]\!])^\sim$, whence the chosen occurrence of P should be positive in $(C)^\sim$, i. e., also positive in d'.

(3°) $A \equiv \forall u.B[\![u]\!]$, analogously. □

11.13 Definition.

A $\beta\mathfrak{d}$-détour of the form $d \equiv \mathfrak{d}x{:}A^-.a[\![x]\!]$ is

(1°) *atomic*, if $A \equiv \top$, $\bot$ or an atomic type/proposition P, else it is *non-atomic* or *complex*,

(2°) *critical*, if A contains negative occurences of atomic types/propositions, else it is *non-critical*.

11.14 Remark.

The critical atomic $\beta\mathfrak{d}$-détours are therefore of the form $d \equiv \mathfrak{d}x{:}A^-.a[\![x]\!]$, with $A \equiv P$ [$\equiv P[\mathbf{u}]$], for some (proper) atomic type/proposition P [here, P is negative in d], whereas the non-critical atomic détours are either of the form $d \equiv \mathfrak{d}x{:}\top^-.a[\![x]\!]$, or of the form $d \equiv \mathfrak{d}x{:}\bot^-.a[\![x]\!]$. Note also that the complex détours $d \equiv \mathfrak{d}x{:}A^-.a[\![x]\!]$, with $A \equiv P^-$ [$\equiv [P \supset \bot]$], (i. e., of the form $d \equiv \mathfrak{d}x{:}P^-.a[\![x]\!]$, where $P^- \equiv [[P \supset \bot] \supset \bot]$, as usual) are non-critical.

So, if $\Phi \vdash a : A$, the p-term a contains critical $\beta\mathfrak{d}$-détours iff a contains negative occurrences of atomic types/propositions.

11.15 Corollary.

If $\Phi \vdash a$ then $a^\sim$ contains no critical $\beta\mathfrak{d}$-détours.

Proof. By **11.12**, $(...)^\sim$ eliminates the negative (occurrences of) propositional atoms from a. □

The standard negative translation. We can now introduce one of the possible versions of the *negative translation*.

11.16 Definition.

For p-terms c in Term[λ⅁(!)], the p-term (c)* is defined inductively by:

(0°) if c ≡ x then (c)* ≡ x,
(1°) if c ≡ ab then (c)* ≡ (a*)(b*),
(2°) if c ≡ λx:A.a[[x]] then (c)* ≡ λx:A.(a[[x]])*,
(3°) if c ≡ a[t] then (c)* ≡ (a*)[t],
(4°) if c ≡ !u.a then (c)* ≡ !u.(a)*,
(5°) if c ≡ ⅁x:A⁻.a[[x]]
and
(5°1) if A ≡ ⊤ [≡ ⊥ ⊃ ⊥] then (c)* ≡ λz:⊥.z,
(5°2) if A ≡ ⊥ then (c)* ≡ (d)*
where d ≡ a[[x:=λz:⊥.z]],
(5°3) if A ≡ P [≡ P[u]] then (c)* ≡ λx₀:P.(d[[x₀]])*,
where d[[x₀]] ≡ a[[x:=λz:P⁻.zx₀]],
(5°4) if A ≡ (B⊃C) then (c)* ≡ λx₀:B.(d[[x₀]])*,
where d[[x₀]] ≡ ⅁x₁:C⁻.a[[x:=λz:(B⊃C).x₁(zx₀)]].
(5°5) if A ≡ (∀v.B[[v]]) then (c)* ≡ !u.(d[[u]])*,
where
d[[u]] ≡ ⅁x₁:B[[u]]⁻.a[[x:=λz:(∀v.B[[v]]).x₁(z[u])]].

11.17 Remark.

So, for ⅁-free p-terms c in Term[λ(!)], (c)* ≡ c. Similarly, for *type-normal* p-terms c in Term[λ⅁(!)], (c)* ≡ c, if c does not contain atomic β⅁-détours (i. e., of the form ⅁x:A⁻.a[[x]], with A ≡ ⊤ [≡ ⊥ ⊃ ⊥, here] or A ≡ ⊥ or A ≡ P.).

11.18 Remark.

For all practical purposes, one could have used a *restriction* of (...)* to a mapping (...)** given as follows: for p-terms c in Term[λ⅁(!)], (c)** is a p-term defined by

(0°) c ≡ x then (c)** ≡ x,
(1°) c ≡ ab then (c)** ≡ (a**)(b**),
(2°) c ≡ λx:A.a[[x]] then (c)** ≡ λx:A.(a[[x]])*,
(3°) c ≡ a[t] then (c)** ≡ (a**)[t],
(4°) c ≡ !u.a then (c)** ≡ !u.(a)**,
(5°) c ≡ ⅁x:A⁻.a[[x]] and
(5°1) A ≡ ⊤ [≡ ⊥ ⊃ ⊥] then (c)** ≡ λz:⊥.z,
(5°2) A ≡ ⊥ then (c)** ≡ (a[[x:=λz:⊥.z]])**,
(5°3$_P$) A ≡ P then (c)** ≡ λx₀:P.(d[[x₀]])**,
where d[[x₀]] ≡ a[[x:=λz:P⁻.zx₀]].

11.19 Definition (*The negative translation*).

For any statement $\varphi \equiv (\Phi, a, A)$ of $\lambda^{\vdash}\beta(!)$, φ^N is defined by $\varphi^N \equiv (\Phi^N, a^N, A^N)$, where

(1°) $\Phi^N \equiv \Phi^- \; [\equiv \mathbf{sub}(\Phi)]$,
(2°) $A^N \equiv A^- \; [\equiv \mathbf{sub}(A)]$ and
(3°) $a^N \equiv ((^{tn}a)^-)^*$.

11.20 Remark.

It is easy to see that $^N : \lambda^{\vdash}\beta(!) \dashrightarrow \lambda(!)$ is a well defined mapping from t-statements to t-statements.

Indeed, the terms of the form $c \equiv {}^{tn}(a)$ *exist*, by $\mathbf{SN}[{}^{\vdash}\beta\eta]$, and are *uniquely determined*, by confluence for hyper-$\beta\eta$-reductions. Moreover, $\mathbf{CR}[{}^{\vdash}\beta\eta]$ implies that the actual strategy/path we might have chosen in order to reach a type-normal form is completely *immaterial* (the result must be, anyway, the same). This situation is - clearly - not affected by arbitrary applications of the (simultaneous) substitution operation $(\ldots)^- \equiv (\ldots[\![P_j := (P_j \supset \bot)]\!]_{<j<m>})$ [for all atomic P_j in c].

The p-terms of the form c^-, with $c \equiv {}^{tn}a$, can possibly contain

(1°) proper $\lambda(!)$-détours and/or
(2°) non-critical $\beta\eta$-détours $d \equiv \eta x{:}A^-.a[\![x]\!]$.

In the second case, one can have only the following sub-cases:

(2°1) atomic $\beta\eta$-détours such that
(2°11) $A \equiv \top \; [\equiv \bot \supset \bot]$ with $(d)^* \equiv \lambda z{:}\bot.z$, or
(2°12) $A \equiv \bot$ with $(d)^* \equiv (a[\![x := \lambda z{:}\bot.z]\!])^*$,

(2°2) non-atomic $\beta\eta$-détours such that
(2°21) $A \equiv P^- \; [\equiv P \supset \bot]$ with $(d)^* \equiv \lambda x_0{:}P.(e[\![x_0]\!])^*$,
where $e[\![x_0]\!] \equiv \eta x_1{:}\bot^-.a[\![x := \lambda z{:}(P \supset \bot).x_1(zx_0)]\!]$.

So, clause (5°4) and (5°5) in the definition of $(\ldots)^*$ would never apply, whereas clause (5°3) applies only for $A \equiv P$. This suggests the following simplification of the above.

11.21 Definition.

For all p-terms $a \in \mathrm{Term}[\lambda\eta(!)]$, $a^n \equiv ((^{tn}a)^-)^*$.

11.22 Corollary.

For all p-terms $a \in \mathrm{Term}[\lambda\eta(!)]$, all contexts Φ, and all propositions/types A,

$$\Phi \vdash a : A \Longrightarrow \Phi^N \vdash a^N \equiv a^n : A^N.$$

Proof. By induction on the length of derivation of the premiss. □

11.23 Remark.

For all p-terms $a \in Term[\lambda\not{g}(!)]$, $\Phi \vdash a \Longrightarrow a^N$ is $\not{g}$-free.

11.24 Lemma.

The negative translation ^{N} preserves proper $\lambda(!)$-détours.

Proof. Let a be a p-term. Then $a^N \equiv ((^{tn}a)^\sim)^*$ and (1) the operation $^{tn}(...)$ preserves proper $\lambda(!)$-détours, since it contracts only hyper-$\beta\not{g}$-détours. (2) Arbitrary substitutions and thus $(...)^\sim$ leave the result unchanged as regards the subterm-structure of the détours, whereas (3) the operation $(...)^*$ preserves proper $\lambda(!)$-détours, by the definition of *. Finally, the composition of proper $\lambda(!)$-détour-preserving operations on p-terms does also preseve proper $\lambda(!)$-détours. □

11.25 Corollary.

For all p-terms $a \in Term[\lambda\not{g}(!)]$, all contexts Φ and all proposition/types A,

$$\Phi \vdash a : A \Longrightarrow \Phi^N \vdash_{\lambda<!>} a^N : A^N.$$

Proof. By **11.23**, for all $a \in Term[\lambda\not{g}(!)]$, a^N is $\not{g}$-free. The result follows then directly, by conservativity [11.2], from the fact that $\Phi \vdash a : A \Longrightarrow \Phi^N \vdash a^N : A^N$ [11.22]. □

11.26 Theorem.

For all p-terms $a, b \in Term[\lambda\not{g}(!)]$ and all contexts Φ,

$$\Phi \vdash a \twoheadrightarrow b \Longrightarrow \Phi^N \vdash_{\lambda<!>} a^N \twoheadrightarrow_{\lambda<!>} b^N.$$

Proof. By induction on the derivation of the premiss in $\lambda\not{g}(!)$, using **11.25**, in order to insure the correctness of typing in $\lambda(!)$. □

This yields, finally, the main result.

11.27 Theorem.

$$\mathbf{SN}[\lambda(!)] \Longleftrightarrow \mathbf{SN}[\lambda^{\supset}\not{g}(!)]$$

Proof. The $\Longrightarrow$-part follows from **11.26**. The converse ($\Longleftarrow$) is straightforward, since $\lambda(!)$ is a sub-system of $\lambda^{\supset}\not{g}(!)$ with

$$\Phi \vdash_{\lambda<!>} a \twoheadrightarrow_{\lambda<!>} b \Longrightarrow \Phi \vdash a \twoheadrightarrow b,$$

for all p-terms $a, b \in Term[\lambda(!)]$ and all contexts Φ. □

11.28 Remark.

We have, of course, in the end, **SN**[$\lambda^{\vdash}\not\gamma(!)$], and, finally, also **SN**[$\lambda\not\gamma(!)$], *tout court*, by **10.10**, since **SN**[$\lambda(!)$] can be proved by standard methods (**9.22**, **9.23**). However, the main interest here is to show that, from a proof-theoretic point of view, $\lambda^{\langle\vdash\rangle}\not\gamma(!)$ is neither more "complex", nor less "constructive" than $\lambda(!)$ could be. In other words: *the elementary proof-theory of first-order classical logic is at most as "complex" as* λ *and at least as "constructive" as* λ, the ordinary typed λ-calculus!

11.29 Remark.

The intuition behind the proof of **SN**[$\lambda(!)$] ==> **SN**[$\lambda^{\vdash}\not\gamma(!)$] above is that we translate, by ^{N}, *entire* standard reduction sequences of $\lambda^{\vdash}\not\gamma(!)$ into standard reduction sequences of $\lambda(!)$. Thereby, proper $\lambda(!)$-détours are translated *verbatim*, whereas the proper $\not\gamma$-détours are replaced by appropriate sequences of proper $\lambda(!)$-détours. So, finally, ^{N} "prolongates" an original reduction sequence ρ into a longer ρ', but entirely within $\lambda(!)$. The fact that ^{N} "disturbs" the original typings is immaterial for the proof-mechanism behind détour-eliminations and, in particular, for termination properties in détour-elimination processes!

12. Confluence in first-order classical logic. We show, finally, **CR**[$^{\mathrm{TN}}\lambda\mathcal{G}!$], *confluence for the type-normal fragment of* $\lambda\mathcal{G}!$, by the "residuation" [Rosser/Tait/Martin-Löf] technique, using a structural case-analysis of parallel $\lambda\mathcal{G}!$-reductions, as restricted to the set Term[$^{\mathrm{TN}}\lambda\mathcal{G}!$]. This implies, by **10.20**, the property we are looking for, viz. **UN**[$\lambda\mathcal{G}!$], *unicity of normal forms in* $\lambda\mathcal{G}!$.

Isolating parallel $^{\mathrm{TN}}\lambda\mathcal{G}(!)$*-reductions.* For technical purposes, in the proof of confluence for **CQ**, one needs a "parallel" version of the $\lambda\mathcal{G}(!)$-reduction rules.

12.1 Theorem (*Parallel reductions*).

The following rules, matching the initial classification of the reduction rules of $\lambda\mathcal{G}!$ in the obvious way, are *derivable* in $\lambda\mathcal{G}!$ (with the usual provisoes on contexts):

p-evaluation.

Basic p-evaluation.

$$(\beta\sqsupset\lambda)_{\mathrm{p}}:\ \frac{\Phi_1 \vdash a_1 \Rightarrow\!\!> a_2 : A \qquad \Phi_2[\,x : A\,] \vdash b_1[\![x]\!] \Rightarrow\!\!> b_2[\![x]\!] : B}{\Phi_1\Phi_2 \vdash (\lambda x{:}A.b_1[\![x]\!])a_1 \Rightarrow\!\!> b_2[\![x:=a_2]\!]\ [:\ B]},$$

$$(\eta\sqsupset\lambda)_{\mathrm{p}}:\ \frac{\Phi \vdash f_1 \Rightarrow\!\!> f_2 : A \sqsupset B}{\Phi \vdash \lambda x{:}A.f_1x \Rightarrow\!\!> f_2},\ [x \text{ not free in } f_i,\ i := 1,2],$$

$$(\eta\sqsupset\mathcal{G})_{\mathrm{p}}:\ \frac{\Phi \vdash f_1 \Rightarrow\!\!> f_2 : A}{\Phi \vdash \mathcal{G}x{:}A^{-}.xf_1 \Rightarrow\!\!> f_2},\ [x \text{ not free in } f_i,\ i := 1,2].$$

First-order p-evaluation.

$$(\beta\forall)_{\mathrm{p}}:\ \frac{\Phi_1 \Vdash t :: U \qquad \Phi_2[\,u :: U\,] \vdash a_1[\![u]\!] \Rightarrow\!\!> a_2[\![u]\!] : A[\![u]\!]}{\Phi_1\Phi_2 \vdash (!u.a_1[\![u]\!])[t] \Rightarrow\!\!> a_2[\![u:=t]\!]\ [:\ A[\![u:=t]\!]]},$$

$$(\eta\forall)_{\mathrm{p}}:\ \frac{\Phi \vdash f_1 \Rightarrow\!\!> f_2 : \forall u.A[\![u]\!]}{\Phi \vdash !u.f_1[u] \Rightarrow\!\!> f_2},\ [u \text{ not free in } f_i,\ i := 1,2],$$

p-type-reduction.

Basic p-type-reduction.

$$(\beta\mathcal{G}\mathsf{T})_{\mathrm{p}}:\ \frac{\Phi[\,x : \mathsf{T}^{-}\,] \vdash a_1[\![u]\!] \Rightarrow\!\!> a_2[\![u]\!] : \bot}{\Phi \vdash \mathcal{G}x{:}\mathsf{T}^{-}.a_1[\![x]\!] \Rightarrow\!\!> \Omega\ [:\ \mathsf{T}]},$$

$$(\beta\not\partial\bot)_p:\ \frac{\Phi[\ x\ :\ \bot^-\ ]\ \vdash\ a_1[\![u]\!]\ \twoheadrightarrow\ a_2[\![u]\!]\ :\ \bot}{\Phi\ \vdash\ \not\partial x{:}\bot^-.a_1[\![x]\!]\ \twoheadrightarrow\ a_2[\![x:=\lambda z{:}\bot.z]\!]\ [:\ \bot]},$$

$$(\beta\not\partial\sqsupset)_p:\ \frac{\Phi_1\ \vdash\ a_1\ \twoheadrightarrow\ a_2\ :\ A \qquad \Phi_2[\ x\ :\ (A\sqsupset B)^-\ ]\ \vdash\ c_1[\![x]\!]\ \twoheadrightarrow\ c_2[\![x]\!]:\ \bot}{\Phi_1\Phi_2\ \vdash\ (\not\partial x{:}(A\sqsupset B)^-.c_1[\![x]\!])a_1\ \twoheadrightarrow\ \not\partial x{:}B^-.c'[\![x]\!]\ [:\ B]},$$

where $c'[\![x]\!] \equiv c_2[\![x:=\lambda z{:}(A\sqsupset B).x(za_2)]\!]$.

First-order p-type-reduction. If u is not free in $c_i[\![x]\!]$ (i := 1,2),

$$(\beta\not\partial\forall)_p:\ \frac{\Phi_1\ \Vdash\ t\ ::\ U \qquad \Phi_2[\ x\ :\ (\forall u.A[\![u]\!])^-\ ]\ \vdash\ c_1[\![x]\!]\ \twoheadrightarrow\ c_2[\![x]\!]:\ \bot}{\Phi_1\Phi_2\ \vdash\ (\not\partial x{:}(\forall u.A[\![u]\!])^-.c_1[\![x]\!])[t]\ \twoheadrightarrow\ \not\partial x{:}A^-[\![u:=t]\!].c'[\![x]\!]\ [:\ A']},$$

where $c'[\![x]\!] \equiv c_2[\![x:=\lambda z{:}(\forall u.A[\![u]\!]).x(z[t])]\!]$
[**and** $A' \equiv A[\![u:=t]\!]$].

p-compatibility.

$$(\mu\vee\sqsupset):\ \frac{\Phi_1\ \vdash\ f\ \twoheadrightarrow\ g\ :\ A\sqsupset B \qquad \Phi_2\ \vdash\ a_1\ \twoheadrightarrow\ a_2\ :\ A}{\Phi_1\Phi_2\ \vdash\ fa_1\ \twoheadrightarrow\ ga_2\ [:\ B]}.$$

Proof. Easy, but making hard use of $(\tau{=}\twoheadrightarrow)$. [*Exercise*: For instance, $(\mu\vee\sqsupset)$, follows from $(\mu\sqsupset)$, $(\vee\sqsupset)$ and $(\tau{=}\twoheadrightarrow)$.] □

12.2 Remark.

Note that, with the exception of $(\mu\sqsupset)$, $(\vee\sqsupset)$, the remaining $-\!\!\twoheadrightarrow$-compatibility rules of $\lambda\not\partial!$, viz.,

$(\xi\sqsupset\lambda)$, $(\xi\sqsupset\not\partial)$, [*basic rules*],
$(\mu\forall)$, $(\xi\forall)$, [*first-order rules*],

do already fit the "parallel" pattern of the derived rule $(\mu\vee\sqsupset)$ [***p-compatibility***] above.

12.3 Definition.

Let $\rightarrowtail$ be the least binary relation on the set of $\lambda\not\partial!$-terms, satisfying, *mutatis mutandis*, the following conditions:

0 ***reflexivity:***
(ρ) [*basic rule*].

1 ***p-evaluation:***
$(\beta\sqsupset\lambda)_p$, $(\eta\sqsupset\lambda)_p$, $(\eta\sqsupset\not\partial)_p$, [*basic rules*],
$(\beta\forall)_p$, $(\eta\forall)_p$, [*first-order rules*].

2 *p-type-reduction:*
$(\beta\eth\mathbf{T})_p$, $(\beta\eth\mathbf{\bot})_p$, [*basic rules*],
$(\beta\eth\sqsupset)_p$, [*basic rule*],
$(\beta\eth\forall)_p$, [*first-order rule*].

3 *p-compatibility:*
$(\mu\vee\sqsupset)$, $(\S\sqsupset\lambda)$, $(\S\sqsupset\eth)$, [*basic rules*],
$(\mu\forall)$, $(\S\forall)$, [*first-order rules*].

Next, $\rightarrowtail$ is called (*classical first-order*) *parallel reduction*. The restriction of $\rightarrowtail$ to basic rules is called *basic parallel reduction*.

Notation. In order to avoid possible confusions we write oft (**label**:$\rightarrowtail$) for $\rightarrowtail$-rules, where **label** is any one of the labels listed above. E. g., $(\S\sqsupset\lambda{:}\rightarrowtail)$ identifies the $(\S\sqsupset\lambda)$-rule for $\rightarrowtail$. [**NB**: Note that "$\vdash$" and "$\rightarrowtail$" are used ambiguously in the rest of this section.]

12.4 Corollary.

For all $\lambda\eth!$-terms a, b, one has: $\Phi \vdash a \rightarrowtail b \Longrightarrow \Phi \vdash a \twoheadrightarrow b$.

Proof. By induction on the [length of] derivation of the premiss, using **12.1**. □

12.5 Lemma.

For all $\lambda\eth!$-terms a, b, one has: $\Phi \vdash a \twoheadrightarrow_\beta b \Longrightarrow \Phi \vdash a \rightarrowtail b$.

Proof. By induction on the [length of] derivation of the premiss, using $(\rho{:}\rightarrowtail)$. □

12.6 Theorem.

Let $\rightarrowtail\!\!\!\!\twoheadrightarrow$ be the transitive closure of $\rightarrowtail$ (i.e., the least transitive relation satisfying the conditions in the **Definition** of $\rightarrowtail$). Then $\rightarrowtail\!\!\!\!\twoheadrightarrow = \twoheadrightarrow$ as relations on the set of $\lambda\eth!$-terms. In other words: for all $\lambda\eth!$-terms a, b, we have: $\Phi \vdash a \rightarrowtail\!\!\!\!\twoheadrightarrow b \Longleftrightarrow \Phi \vdash a \twoheadrightarrow b$.

Proof. The ($\Longrightarrow$)-part is stated in **12.4**, whereas, the ($\Longleftarrow$)-part follows by **12.5** and $(\tau{:}\twoheadrightarrow)$. □

12.7 Remark.

Theorem 12.6 states, in fact, that $\rightarrowtail$ and $\twoheadrightarrow$, (resp. $\twoheadrightarrow_\beta$) have the same transitive closure, viz. $\twoheadrightarrow$ itself. So, the reduction rules of $\lambda\eth!$ and/or its subsystems could have been formulated, equivalently, in terms of appropriate "parallel" notions of reduction $\rightarrowtail$. Formal proof-systems with a primitive "parallel" notion of reduction are more informative in epi-theoretic considerations on proof-reduction.

Before proving the main fact we should also note the following:

12.8 Lemma *(Substitution Lemma).*

$$(\${:}\rightarrowtail):\quad \frac{\Phi[x{:}A] \vdash a_1[\![x]\!] \rightarrowtail b_1[\![x]\!] : B \qquad \Phi \vdash a_2 \rightarrowtail b_2 : A}{\Phi \vdash a_1[\![x:=a_2]\!] \rightarrowtail b_1[\![x:=b_2]\!] : B},$$

$$(\$_U{:}\rightarrowtail):\quad \frac{\Phi[\,u :: U\,] \vdash a_1[\![u]\!] \rightarrowtail b_1[\![u]\!] : A \qquad \Phi \Vdash t :: U}{\Phi \vdash a_1[\![u:=t]\!] \rightarrowtail b_1[\![u:=t]\!] : A}.$$

Proof. In each case, by induction on the total [sum-] length of the derivation of the premisse(s). □

Type-normal "residuals" in $\lambda\partial!$. We organize the "residuals" of $^{TN}\lambda\partial!$ in a transparent way.

For a - otherwise standard - analogue of this, applying to the ordinary "extensional" type-free λ-calculus $^{f}\lambda$, see, for instance [Takahashi 1989], 3.1).

12.9 Definition *("$^{TN}\lambda\partial!$-residuals").*

For each p-term $a \in \mathrm{Term}[^{TN}\lambda\partial!]$, such that $\Phi \vdash a$, for some Φ, one defines a^R ("the residual of a") by induction on the structure of a [distinguishing among détours], as follows:

(1°)	$a \equiv x$	$a^R \equiv x$,
(2°)	$a \equiv a_1 a_2$	
(2°1)	a is not a $[\beta\supset\lambda]$-détour	$a^R \equiv (a_1)^R (a_2)^R$,
(2°2)	$a_1 \equiv \lambda x{:}A.a_0$	
(2°21)	a_1 is not an $[\eta\supset\lambda]$-détour	$a^R \equiv (a_0)^R[\![x:=(a_2)^R]\!]$,
(2°22)	$a_0 \equiv a_{00}x$, x not in $FV(a_{00})$	$a^R \equiv (a_{00})^R (a_2)^R$
(3°)	$a \equiv \lambda x{:}A.a_1$	
(3°1)	a is not an $[\eta\supset\lambda]$-détour	$a^R \equiv \lambda x{:}A.(a_1)^R$,
(3°2)	$a_1 \equiv a_0 x$, x not in $FV(a_0)$	$a^R \equiv (a_0)^R$,
(4°)	$a \equiv \partial x{:}A^-.a_1$	
(4°1)	a is not an $[\eta\supset\partial]$-détour:	
(4°11)	$A \equiv \top$	$a^R \equiv \lambda x{:}\bot.x$,
(4°12)	$A \equiv \bot$	$a^R \equiv (a_1)^R[\![x:=\lambda x{:}\bot.x]\!]$,
(4°13)	A is an atomic proposition P	$a^R \equiv \partial x{:}A^-.(a_1)^R$,
(4°2)	$a_1 \equiv xa_0$, x not in $FV(a_0)$	$a^R \equiv (a_0)^R$,
(5°)	$a \equiv a_1[t]$	
(5°1)	a is not a $[\beta\forall]$-détour	$a^R \equiv (a_1)^R[t]$,
(5°2)	$a_1 \equiv !u.a_0$	
(5°21)	a_1 is not an $[\eta\forall]$-détour	$a^R \equiv (a_0)^R[\![u:=t]\!]$,
(5°22)	$a_0 \equiv a_{00}[u]$, u not in $FV(a_{00})$	$a^R \equiv (a_{00})^R[t]$,
(6°)	$a \equiv !u.a_1$	
(6°1)	a is not an $[\eta\forall]$-détour	$a^R \equiv !u.(a_1)^R$,
(6°2)	$a_1 \equiv a_0[u]$, u not in $FV(a_0)$	$a^R \equiv (a_0)^R$.

With this notation, the main fact behind the present confluence-proof is contained in the following lemma.

12.10 Lemma (*"Residual inversion"*).

For all a, b ∈ Term[ᵀᴺλ𝒻(!)], Φ ⊢ a ↣ b ==> Φ ⊢ b ↣ (a)ᴿ.

Proof. By induction on the structure of a, distinguishing among détours, in order to be able to use a sub-induction on the length of derivation of Φ ⊢ a ↣ b in ᵀᴺλ𝒻(!) [details in the **Appendix**]. □

12.11 Theorem (*Confluence for classical proof-reduction*).

CR[ᵀᴺλ𝒻(!)]. That is: for all p-terms a, b_i ∈ Term[ᵀᴺλ𝒻(!)], (i := 1, 2), there is a p-term c ∈ Term[ᵀᴺλ𝒻(!)], such that

$$\Phi \vdash a \rightarrowtail b_i \Longrightarrow \Phi \vdash b_i \rightarrowtail c.$$

Proof. Immediate, from **12.10**. □

12.12 Theorem (*The "Church-Rosser" theorem for classical logic*).

For all p-terms a_i ∈ Term[ᵀᴺλ𝒻(!)] (i := 1, 2), there is a p-term b ∈ Term[ᵀᴺλ𝒻(!)], such that

$$\Phi \vdash a_1 = a_2 \Longrightarrow \Phi \vdash a_i \rightarrowtail b.$$

Proof. From **CR**[ᵀᴺλ𝒻(!)], i. e., **12.11**. □

As expected, this implies the following

12.13 Theorem (*Unicity of λ𝒻(!)-normal forms*).

UN[ᶜᵀᴺᵌλ𝒻(!)].

Proof. For ᵀᴺλ𝒻(!): clear. Delete "TN" by **UN**[λʰ𝒻!,ʰβ𝒻], [**10.20**]. □

Indeed, the main interest was in obtaining **UN**[λ𝒻!], *definiteness of termination for the processes involving détour eliminations in* λ𝒻! (for this, we don't need a would-be *full* **CR**[λ𝒻!]-property, for instance).

Moreover, we have also, once more,

12.14 Theorem (*Consistency for classical proof-equality*).

Cons[ᶜᵀᴺᵌλ𝒻!].

Proof. For ᵀᴺλ𝒻!: from **12.13**, as ever. The "TN"-restriction can be deleted in view of **SN**[λʰ𝒻!,ʰβ𝒻] and **CR**[λʰ𝒻!,ʰβ𝒻], i. e., [**10.12**] and [**10.18**].

12.15 Note

Jean-Yves Girard has claimed oft (in conversation) "classical proofs are *non-confluent*", using an *L*-formalism (a "sequent calculus") and a loose way of talking about "reductions"/ "confluence" in such a system, in order to support the claim. In the present setting, this becomes $\neg\mathbf{CR}[\lambda\partial!, \mathbf{r}(\lambda\partial!)]$, where $\mathbf{r}(\lambda\partial!)$ is the *standard* notion of reduction of $\lambda\partial!$, as defined on the full set $\mathrm{Term}[\lambda\partial!]$ (and "$\neg$" is negation in the meta-language). As noted before, $\mathbf{r}(\lambda\partial!)$ can be seen to arise from the corresponding type-normal part $\mathbf{r}(^{\mathrm{TN}}\lambda\partial(!))$ by *systematic abbreviations*. The related claim of Yves Lafont ([Girard *et al.* 1989]: **Appendix B**, p. 152): "classical logic is inconsistent (from an algorithmic viewpoint)" amounts to $\neg\mathbf{Cons}[\lambda\partial!]$ in the present setting and is, simply, false. **NB**: The **Cons**-proof for $\lambda\partial!$ appearing in section **6** [cf. **6.23**] does *not* depend on properties of reduction in $\lambda\partial!$.

13. λ𝒴!-definable proof-operations and the Heyting proof-calculi. In this section we simulate the reduction and equational behavior of proof-operations associated to *conjunction* (∧), *disjunction* (∨) and *classical existence* (∃). Finally, an "extensional" variant of the Heyting proof-calculus for first-order "intuitionistic" logic is shown to be available, definitionally, in λ𝒴!.

"Intensional" conjunction in λ𝒴! and "surjective" λ𝒴-calculi. In a classical proof-setting without a primitive conjunction, there are means of representing a "minimal" ("intensional") conjunction ∧, together with the associated ∧-proof operations.

Recall first that we had, in $\pi_{\Leftarrow}$[CQ], $A \wedge B := (A \supset B^{-})^{-}$ [that is, $A \wedge B \equiv (A \supset (B \supset \bot) \supset \bot$] and Boolean ∧-proof-combinators $\underline{P}[A,B]$, $\underline{U}[A,B,C]$, $\underline{P}_i[A,B]$ (i := 1, 2) [cf. **2.17** (2)] such that, for all types/propositions A, B, C,

$$
\begin{aligned}
&[\ ] \vdash_c \underline{P}[A,B] \ : A \supset .B \supset A \wedge B,\\
&[\ ] \vdash_c \underline{U}[A,B,C] : A \supset .B \supset C \supset (A \wedge B \supset C),\\
&[\ ] \vdash_c \underline{P}_1[A,B] \ : (A \wedge B \supset A),\\
&[\ ] \vdash_c \underline{P}_2[A,B] \ : (A \wedge B \supset B),
\end{aligned}
$$

where "$\underline{P}$" was an "intensional **pairing**" operator, with "left" and "right **projections**" "$\underline{P}_i$" (i := 1, 2), resp., whereas "$\underline{U}$" was an "intensional **un-currying**" (or "intensional **splitting**") operator.

Obviously, one could have had λ𝒴(!)-analogues of the above, as well. In particular, one has some use for "partially evaluated" λ𝒴-forms, corresponding to these proof-terms.

13.1 Definition (*∧-proof operations*).

(1) For all types/propositions A, B, C,

$$
\begin{aligned}
&P[A,B] \ := \lambda x{:}A.\lambda y{:}B.\lambda z{:}(A \supset B^{-}).zxy,\\
&U[A,B,C] := \lambda x{:}(A \supset .B \supset C).\lambda z{:}(A \wedge B).x(1_{A,B}[z])(2_{A,B}[z]),\\
&P_1[A,B] \ := \lambda z{:}(A \wedge B).1_{A,B}[z]),\\
&P_2[A,B] \ := \lambda z{:}(A \wedge B).2_{A,B}[z]),\\
&\text{where } 1_{A,B}[z] \equiv \mathcal{Y}z'{:}A^{-}.z(\lambda x'{:}A.\lambda y'{:}B.z'x')\\
&\text{and } \ 2_{A,B}[z] \equiv \mathcal{Y}z'{:}B^{-}.z(\lambda x'{:}A.z').
\end{aligned}
$$

(2) For all types/propositions A, B, any context Φ and all p-terms a, b such that $\Phi \vdash a : A$ and $\Phi \vdash b : B$, set, in context Φ, $\langle a, b \rangle_{A,B} := \lambda z{:}(A \supset B^{-}).zab$.

(3) If, moreover, in the same conditions, $\Phi \vdash c : A \wedge B$, set in context Φ, resp.,

$$
\begin{aligned}
&p_1[A,B](c) := 1_{A,B}[z:=c] \equiv \mathcal{Y}z'{:}A^{-}.c(\lambda x'{:}A.\lambda y'{:}B.z'x'),\\
&p_2[A,B](c) := 2_{A,B}[z:=c] \equiv \mathcal{Y}z'{:}B^{-}.c(\lambda x'{:}A.z').
\end{aligned}
$$

(4) For all types/propositions A, B, C, any context Φ and all p-terms $c[\![x,y]\!]$, f, such that $\Phi[x:A][y:B] \vdash c[\![x,y]\!] : C$ and $\Phi \vdash f : A \wedge B$, set, in context Φ,

$$\mathbf{U_c}(f,[x:A][y:B].c[\![x,y]\!]) := c[\![x:=a, y:=b]\!],$$

where (locally) $a \equiv \mathbf{1}_{A,B}[\![z:=c]\!]$ and $b \equiv \mathbf{2}_{A,B}[\![z:=c]\!]$.

13.2 Remark.

(1) For all types/propositions A, B,

$$[\ x : A\ ][\ y : B\ ] \vdash \langle x, y \rangle_{A,B} \equiv \lambda z:(A \sqsupset B^{-}).zxy : A \wedge B,$$

$$[\ z : A \wedge B\ ] \vdash \mathbf{p}_1[\![A,B]\!](z) \equiv \mathbf{1}_{A,B}[\![z]\!] \equiv \equiv \mathbb{g}z':A^{-}.z(\lambda x':A.\lambda y':B.z'x') : A,$$

$$[\ z : A \wedge B\ ] \vdash \mathbf{p}_2[\![A,B]\!](z) \equiv \mathbf{2}_{A,B}[\![z]\!] \equiv \equiv \mathbb{g}z':B^{-}.z(\lambda x':A.z') \ll= \ll= \mathbb{g}z':B^{-}.z(\lambda x':A.\lambda y':B.z'y') : B.$$

(2) For all types/propositions A, B, C, and any context Φ,

$$\frac{\Phi[x:A][y:B] \vdash c[\![x,y]\!] : C}{\Phi[z : A \wedge B] \vdash \mathbf{U_c}(f,[x:A][y:B].c[\![x,y]\!]) : C.}$$

(3) Recall that $\mathbf{K}[\![A,B]\!] \equiv_{df} \lambda x:A.\lambda y:B.x$ and $\mathbf{K'}[\![A,B]\!] \equiv_{df} \lambda x:A.\lambda y:B.y$. Then, for all types/propositions A, B,

$$[\ ] \vdash (\mathbf{U}[\![A,B,A]\!])(\mathbf{K}[\![A,B]\!]) \Rightarrow \mathbf{P}_1[\![A,B]\!],$$
$$[\ ] \vdash (\mathbf{U}[\![A,B,B]\!])(\mathbf{K'}[\![A,B]\!]) \Rightarrow \mathbf{P}_2[\![A,B]\!].$$

It is then easy to see that type-assignment ("natural deduction"-like) and "evaluation" rules, analogous to Martin-Löf's [1984] $\wedge$-rules of constructive type theory [**CST**] are derivable in $\lambda\mathbb{g}(!)$, in terms of the families $\mathbf{P}[\![A,B]\!]$ and $\mathbf{U}[\![A,B,C]\!]$ of proof-combinators. [In order to have **13.3** (2)-(3) in **CST**-notation, one must relativize Martin-Löf's generalized/dependent types $\Sigma x:A.B[\![x]\!]$ to "constant families" of types $(\ B[\![x]\!]\ )_{x:A}$ (such that x is not free in $B[\![x]\!]$) and define, as usual in this setting, $A \wedge B \equiv \Sigma x:A.B[\![x]\!]$.]

13.3 Theorem (*$\wedge$-proof operations*).

(1) *$\wedge$-axioms*. For all types/propositions A, B, C, the proof-terms $\mathbf{P}[\![A,B]\!]$, $\mathbf{U}[\![A,B,C]\!]$ and $\mathbf{P}_i[\![A,B]\!]$, resp. have, *mutatis mutandis*, the same typing as the Boolean combinators $\underline{P}[\![A,B]\!]$, $\underline{U}[\![A,B,C]\!]$ and $\underline{P}_i[\![A,B]\!]$ (i := 1, 2), resp. That is,

(**P**): $[\ ] \vdash \mathbf{P}[\![A,B]\!] : A \sqsupset .B \sqsupset (A \wedge B)$,
(**U**): $[\ ] \vdash \mathbf{U}[\![A,B,C]\!] : A \sqsupset .B \sqsupset C \sqsupset (A \wedge B \sqsupset C)$,
($\mathbf{P}_1$): $[\ ] \vdash \mathbf{P}_1[\![A,B]\!] : (A \wedge B) \sqsupset A$,
($\mathbf{P}_2$): $[\ ] \vdash \mathbf{P}_2[\![A,B]\!] : (A \wedge B) \sqsupset B$.

(2) $\wedge$-*stratification*. For all types/propositions A, B, C, and any context Φ,

$$(\wedge i):\ \frac{\Phi \vdash a : A,\ \Phi \vdash b : B}{\Phi \vdash \mathbf{P}[\![A,B]\!](a)(b) \Rightarrow \langle a,\ b \rangle_{A,B} : A \wedge B},$$

$$(\wedge e):\ \frac{\Phi[x:A][y:B] \vdash c[\![x,y]\!] : C \qquad \Phi \vdash f : A \wedge B}{\Phi \vdash \mathbf{U}[\![A,B,C]\!](\lambda x:A.\lambda y:B.c[\![x,y]\!])f \Rightarrow \mathbf{U}_C(f,[x:A][y:B].c[\![x,y]\!]) : C},$$

$$(\wedge e_1):\ \frac{\Phi \vdash f : A \wedge B}{\Phi \vdash \mathbf{P}_1[\![A,B]\!](f) \Rightarrow \mathbf{p}_1[\![A,B]\!](f) : A},$$

$$(\wedge e_2):\ \frac{\Phi \vdash f : A \wedge B}{\Phi \vdash \mathbf{P}_2[\![A,B]\!](f) \Rightarrow \mathbf{p}_2[\![A,B]\!](f) : B.}$$

(3) β-$\wedge$-*evaluation*. For all types/propositions A, B, C, and any context Φ,

$$(\beta\wedge):\ \frac{\Phi[x:A][y:B] \vdash c[\![x,y]\!] : C \qquad \Phi \vdash a : A \qquad \Phi \vdash b : B}{\Phi \vdash (\mathbf{U}[\![A,B,C]\!])(\lambda x:A.\lambda y:B.c[\![x,y]\!])\langle a,b\rangle_{A,B} \Rightarrow \Rightarrow \mathbf{U}_C(\langle a,b\rangle_{A,B},[x:A][y:B].c[\![x,y]\!]) \Rightarrow c[\![x:=a,y:=b]\!] : C},$$

$$(\beta\wedge_1):\ \frac{\Phi \vdash a : A \qquad \Phi \vdash b : B}{\Phi \vdash \mathbf{P}_1[\![A,B]\!](\mathbf{P}[\![A,B]\!](a)(b)) \Rightarrow \mathbf{p}_1[\![A,B]\!](\langle a,b\rangle_{A,B}) \Rightarrow a : A},$$

$$(\beta\wedge_2):\ \frac{\Phi \vdash a : A \qquad \Phi \vdash b : B}{\Phi \vdash \mathbf{P}_2[\![A,B]\!](\mathbf{P}[\![A,B]\!](a)(b)) \Rightarrow \mathbf{p}_2[\![A,B]\!](\langle a,b\rangle_{A,B}) \Rightarrow b : B}.$$

(4) $\wedge$-*compatibility*. For all types/propositions A_1, A_2, A, B, and any context Φ,

$$(\mu\wedge_i):\ \frac{\Phi \vdash c_1 \Rightarrow c_2 : A_1 \wedge A_2}{\Phi \vdash \mathbf{p}_i[\![A_1,A_2]\!](c_1) \Rightarrow \mathbf{p}_i[\![A_1,A_2]\!](c_2) : A_i},\ [i := 1,2],$$

$$(\xi\wedge):\ \frac{\Phi \vdash a_1 \Rightarrow a_2 : A \qquad \Phi \vdash b_1 \Rightarrow b_2 : B}{\Phi \vdash \langle a_1,\ b_1 \rangle_{A,B} \Rightarrow \langle a_2,\ b_2 \rangle_{A,B} : A \wedge B.}$$

Proof. Easy calculations [*exercise*]. □

13.4 Remark (*"Surjective" extensions of* $\lambda\partial!$).

(1) However, we can not simulate a corresponding "extensionality" principle for the pairing [$\mathbf{P}[\![A,B]\!]$, $\mathbf{P}_1[\![A,B]\!]$, $\mathbf{P}_2[\![A,B]\!]$] above, e. g., the so-called "surjectivity" property:

$(\eta\wedge)$: $\Phi \vdash c : A \wedge B \Longrightarrow \Phi \vdash \langle\, \mathbf{p}_1[\![A,B]\!](c),\ \mathbf{p}_2[\![A,B]\!](c)\, \rangle_{A,B}\ \mathbf{R}\ c$,

fails, for $\mathbf{R} \in \{ \Rightarrow, = \}$. Indeed, if $(\eta\wedge)$ could obtain in $\lambda\partial(!)$, one should also have, by the translation $(\ldots)^{\mathsf{F}}$ used in the proof of **6.23**, a "surjective pairing" in the "type-free" calculus $^{\mathsf{f}}\lambda!$ and, ultimately (cf., e. g., the proof of **6.7**), in the ordinary "type-free" calculus $^{\mathsf{f}}\lambda$. However, this contradicts a well-known negative result in λ-calculus (undefinability of "surjective pairing" in $^{\mathsf{f}}\lambda$; cf., e. g., [Barendregt 1974]).

(2) With conjunction primitive in the type-syntax - written as &, for convenience, i. e., with a provability language $\mathcal{L}[\supset,\&,\forall]$ - and a "polymorphic" *proof-term syntax* given, resp. by

- $\langle a, b \rangle$, *pairs*,
- $\mathbf{p}_i(c)$, [i := 1,2], *projections*,

resp., in place of the previous type-parametric notation $\langle a,b\rangle_{A,B}$, $\mathbf{p}_i[\![A,B]\!]$ (i := 1, 2), one could have defined thus a *proper* equational extension $\mathbf{p}\lambda\partial(!)$, say, of $\lambda\partial(!)$, by the following rules [replacing $\wedge$ by & and forgetting the type-parameters from $\langle a,b\rangle_{A,B}$, $\mathbf{p}_i[\![A,B]\!]$, i := 1, 2)]:

- &-*stratification*: (&i), ($\&e_i$),
- &-*evaluation*: ($\beta\&_i$), ($\eta\&$),
- &-*compatibility*: ($\mu\&_i$), ($\xi\&$),

[i := 1,2], together with the following additional $\beta\partial\&$-rules:

- $\mathbf{p}\lambda\partial(!)$-***reductio*** (&-*type-reduction rules*):

$$(\beta\partial\&_1):\ \frac{\Phi[\ x\ :\ (A\&B)^-\ ] \vdash b[\![x]\!] : \bot}{\Phi \vdash \mathbf{p}_1(\partial x{:}(A\&B)^-.\, b[\![x]\!]) \Rightarrow \partial \mathbf{x}{:}A^-.\, b[\![x{:=}\lambda z{:}(A\&B).\, \mathbf{x}(\mathbf{p}_1 z)]\!]\ [:\ A]},$$

$$(\beta\partial\&_2):\ \frac{\Phi[\ x\ :\ (A\&B)^-\ ] \vdash b[\![x]\!] : \bot}{\Phi \vdash \mathbf{p}_2(\partial x{:}(A\&B)^-.\, b[\![x]\!]) \Rightarrow \partial \mathbf{x}{:}B^-.\, b[\![x{:=}\lambda z{:}(A\&B).\, \mathbf{x}(\mathbf{p}_2 z)]\!]\ [:\ B]}.$$

Clearly, the *surjective* proof-calculi $\mathbf{p}\lambda\partial(!)$ are *distinct* from $\lambda\partial(!)$ only inasfar reduction/equational proof-behaviors are concerned (namely, as induced by the rules $(\eta\&)$ and $(\beta\partial\&_i)$, [i := 1,2], resp.).

(3) The main results of this book trasfer to $p\lambda\partial(!)$, as well. For instance, $\mathbf{Cons}[p\lambda\partial(!)]$ can be also shown by the methods of section 6, using $\mathbf{Cons}[{}^{r}p\lambda]$ instead, where ${}^{r}p\lambda$ is the *extensional* "type-free" λ-calculus with "surjective pairing" (for the latter type of result one must solve "Scott domain equations" of the form $D \approx [D \times D] \approx [D \to D]$, first). Also, the proof of $\mathbf{SN}[p\lambda\partial!]$ can use a "negative translation" similar to that of section 11, together with a remark of [Troelstra 1986], in order to reduce $\mathbf{SN}[p\lambda(!)]$ to $\mathbf{SN}[\lambda(!)]$. Ultimately, one has $\mathbf{SN}[p\lambda\partial!] \Leftrightarrow \mathbf{SN}[\lambda]$, too. However, with "type-normal" appropriately defined, $\mathbf{CR}[{}^{TN}p\lambda\partial!]$ cannot be, apparently, obtained directly, as in section 12; instead, one can use a Newman-argument [9.21], showing "weak confluence" first.

(4) For a set of type-assignment rules similar to those of $p\lambda\partial!$ above, see, e. g., the Gentzen *N*-system for **CQ**-*provability* appearing in [Prawitz 1965]. The Prawitz rules are stated, in fact, for a provability language with primitives $\bot$, $\supset$, $\&$ and $\forall$. For "type-reduction", see *mutatis mutandis*, Prawitz' *immediate simplifications* ([Prawitz 1971], **3.1.4**, **3.2.4.4**, **3.3.2.3** and **3.4**).

(5) Note that rules analogous to $(\beta\partial\&_i)$, $[i := 1,2]$, can be already obtained for $\wedge$, and the $\lambda\partial(!)$-definable pairing $[\, P[\![A,B]\!],\ P_1[\![A,B]\!],\ P_2[\![A,B]\!]\,]$, discussed earlier, *provided* one assumes – beyond the rules of $\lambda\partial(!)$ – an analogue of $(\eta\&)$ for the family of triples $[\, P[\![A,B]\!],\ P_1[\![A,B]\!],\ P_2[\![A,B]\!]\,]$.

(6) Martin-Löf [1984] derives an analogue of $(\eta\&)$ from specific assumptions on "identity types" $I(A)$. Such assumptions are debatable in a *classical* proof-setting. Cf. also [Rezus 1986].

Classical disjunctions and existence. In a genuinely classical proof-setting, there are several distinct ways of representing "minimal" (or "intensional") $\vee$- and $\exists$-proof operations [cf. **2.17** (3), (5) above, for combinatory variants].

We had $A \vee B \equiv_{df} (A^- \wedge B^-)^- \equiv (A^- \supset B^=)^= \equiv (A^- \supset (B^- \supset \bot)) \supset \bot)^- \equiv$ $\equiv (((A \supset \bot) \supset ((B \supset \bot) \supset \bot)) \supset \bot) \supset \bot$ and $\exists u.A[\![u]\!] := (\forall u.(A[\![u]\!])^-)^-$ $[\, \equiv (\forall u.(A[\![u]\!] \supset \bot)) \supset \bot\,]$. A convenient set of proof-operations associated to $\vee$ and $\exists$ is as follows.

13.5 Definition (*$\vee$-proof operations*).

(1) For all types/propositions A, B, C,

$$J_1[\![A,B]\!] := \lambda x{:}A.\lambda z{:}(A^- \wedge B^-).P_1[\![A^-,B^-]\!](z)(x),$$
$$J_2[\![A,B]\!] := \lambda y{:}B.\lambda z{:}(A^- \wedge B^-).P_2[\![A^-,B^-]\!](z)(y),$$
$$D[\![A,B,C]\!] := \lambda x{:}(A \supset B).\lambda y{:}(A \supset C).\lambda z'{:}(A \vee B).\partial z{:}C^-.z'c,$$

where $c \equiv c_{A,B}[\![x,y,z]\!] \equiv P[\![A^-,B^-]\!](\lambda x'{:}A.z(xx'))(\lambda y'{:}B.z(yy'))$ and, as usual, $A^= \equiv (A^-)^- \equiv (A \supset \bot) \supset \bot$.

(2) For and all types/propositions A, B, C, and any context Φ,

(1°) whenever a, b are such that $\Phi \vdash a : A$ and/or $\Phi \vdash b : B$, set, in context Φ,

$$j_1[\![A,B]\!](a) := \lambda z{:}(A^- \wedge B^-).z(\lambda x'{:}A^-.\lambda y'{:}B^-.x'a),$$
$$j_2[\![A,B]\!](b) := \lambda z{:}(A^- \wedge B^-).z(\lambda x'{:}A^-.\lambda y'{:}B^-.y'b),$$

(2°) if $\Phi \vdash h : A \vee B$, $\Phi[x{:}A] \vdash a[\![x]\!] : C$, $\Phi[y{:}B] \vdash b[\![y]\!] : C$ and $f_A[\![z]\!] \equiv \lambda x{:}A.za[\![x]\!]$, $g_B[\![z]\!] \equiv \lambda y{:}B.zb[\![y]\!]$ (where the "intensional" pair $\langle\ldots,\ldots\rangle$ is relative to A^-, B^-), set in context Φ,

$$D_C(h,[x{:}A].a[\![x]\!],[y{:}B].b[\![y]\!]) := \not\lambda z{:}C^-.h\langle f_A[\![z]\!],g_B[\![z]\!]\rangle.$$

13.6 Definition (*$\exists$-proof operations*).

(1) For all types/propositions A[u], B,

(1°) $\pi[\![A]\!] := !u.\lambda x{:}A[\![u]\!].\lambda f{:}\forall v.(A[\![u:=v]\!])^-.f[u]x$, provided v is not in $FV_U(A[\![u]\!])$,

(2°) $\tau[\![A,B]\!] := \lambda f{:}\forall u.(A[\![u]\!] \supset B).\lambda x{:}(\exists u.A[\![u]\!]).\not\lambda z{:}B^-.x(a_A[\![f,z]\!])$, where $a_A[\![f,z]\!] \equiv !v.\lambda y{:}A[\![u:=v]\!].z(f[v]y)$, provided u is not in $FV_U(B)$.

(2) For all types/propositions A[u], and any context Φ, if t and the p-term a are such that $\Phi \Vdash t :: U$ and $\Phi \vdash a : A[\![u:=t]\!]$, set, in context Φ,

$$[\ t,\ a\ ]_A := \lambda f{:}(\forall v.(A[\![u:=v]\!])^-).f[t]a,$$

where v is not in $FV_U(A[\![u]\!])$.

(3) For all types/propositions A[u], B, and any context Φ, if the p-terms c and a[u,x] are such that $\Phi \vdash c : \exists u.A[\![u]\!]$ and $\Phi[\ u :: U\ ][\ x : A\ ] \vdash a[\![u,x]\!] : B$, set, in context Φ,

$$\tau_B(c,[u::U][x{:}A[\![u]\!]].a[\![u,x]\!]) := \not\lambda z{:}B^-.c\ f[\![z]\!],$$

where $f[\![z]\!] \equiv (!v.\lambda y{:}A[\![u:=v]\!].z(a[\![u:=v,x:=y]\!]))$, such that u is not in $FV_U(B)$.

13.7 Theorem (*$\vee$-proof operations*).

(1) *$\vee$-axioms*. For all types/propositions A, B, C,

$$[\ ] \vdash J_1[\![A,B]\!] : A \supset (A \vee B),$$
$$[\ ] \vdash J_2[\![A,B]\!] : B \supset (A \vee B),$$
$$[\ ] \vdash D[\![A,B,C]\!] : A \supset C \supset .B \supset C \supset (A \vee B \supset C).$$

(2) ∨-*stratification*. For all types/propositions A, B, C, and any context Φ,

$$(\vee i_1):\quad \frac{\Phi \vdash a : A}{\Phi \vdash J_1[\![A,B]\!](a) \Rightarrow j_1[\![A,B]\!](a) : A \vee B},$$

$$(\vee i_2):\quad \frac{\Phi \vdash b : B}{\Phi \vdash J_2[\![A,B]\!](b) \Rightarrow j_2[\![A,B]\!](b) : A \vee B}$$

$$(\vee e):\quad \frac{\Phi[x:A] \vdash a[\![x]\!] : C \quad \Phi[y:B] \vdash b[\![y]\!] : C \quad \Phi \vdash h : A \vee B}{\Phi \vdash (D[\![A,B,C]\!])(\lambda x:A.a[\![x]\!])(\lambda y:B.b[\![y]\!])(h) \Rightarrow \Rightarrow D_C(h,[x:A].a[\![x]\!],[y:B].b[\![y]\!]) : C.}$$

(3) β-∨-*evaluation*. For all types/propositions A, B, C, and any context Φ,

$$(\beta\vee_1):\quad \frac{\Phi[x:A] \vdash a[\![x]\!] : C \quad \Phi[y:B] \vdash b[\![y]\!] : C \quad \Phi \vdash f : A}{\Phi \vdash (D[\![A,B,C]\!])(\lambda x:A.a[\![x]\!])(\lambda y:B.b[\![y]\!])(J_1[\![A,B]\!](f)) \Rightarrow \Rightarrow D_C(j_1[\![A,B]\!](f),[x:A].a[\![x]\!],[y:B].b[\![y]\!]) \Rightarrow a[\![x:=f]\!] : C},$$

$$(\beta\vee_2):\quad \frac{\Phi[x:A] \vdash a[\![x]\!] : C \quad \Phi[y:B] \vdash b[\![y]\!] : C \quad \Phi \vdash g : B}{\Phi \vdash (D[\![A,B,C]\!])(\lambda x:A.a[\![x]\!])(\lambda y:B.b[\![y]\!])(J_2[\![A,B]\!](g)) \Rightarrow \Rightarrow D_C(j_2[\![A,B]\!](g),[x:A].a[\![x]\!],[y:B].b[\![y]\!]) \Rightarrow b[\![y:=g]\!] : C.}$$

(4) ∨-*compatibility*. For all types/propositions A_1, A_2, A, B, C, and any context Φ,

$$(\xi\vee_i):\quad \frac{\Phi \vdash c_1 \Rightarrow c_2 : A_i}{\Phi \vdash j_i[\![A_1,A_2]\!](c_1) \Rightarrow j_i[\![A_1,A_2]\!](c_2) : A_1 \vee A_2}, \quad [i := 1,2],$$

$$(\mu\vee):\quad \frac{\Phi[x:A] \vdash a_1[\![x]\!] \Rightarrow a_2[\![x]\!] : C \quad \Phi[y:B] \vdash b_1[\![y]\!] \Rightarrow b_2[\![y]\!] : C \quad \Phi \vdash h_1 \Rightarrow h_2 : A \vee B}{\Phi \vdash D_C(h_1,[x:A].a_1[\![x]\!],[y:B].b_1[\![y]\!]) \Rightarrow \Rightarrow D_C(h_2,[x:A].a_2[\![x]\!],[y:B].b_2[\![y]\!]) : C.}$$

Proof. Easy [*exercise*]. □

13.8 Theorem (∃-*proof operations*).

(1) ∃-*axioms*. For all types/propositions A[u], B,

(π): $[\ ] \vdash \pi[\![A]\!] : \forall u.(A[\![u]\!] \supset \exists v.A[\![u:=v]\!])$,
provided v is not in $FV_U(A[\![u]\!])$,
(τ): $[\ ] \vdash \tau[\![A,B]\!] : \forall u.(A[\![u]\!] \supset B) \supset .(\exists u.A[\![u]\!]) \supset B$,
provided u is not in $FV_U(B)$.

(2) ∃-*stratification*. For all types/propositions A[u], B, and any context Φ,

$$(\exists i):\ \frac{\Phi \Vdash t :: U \qquad \Phi \vdash a : A[\![u:=t]\!]}{\Phi \vdash \pi[\![A]\!][t](a) \Rightarrow [\ t,\ a\ ]_A : \exists u.A[\![u]\!]}$$

$$(\exists e):\ \frac{\Phi[\ u :: U\ ][\ x : A[\![u]\!]\ ] \vdash a[\![u,x]\!] : B \qquad \Phi \vdash c : \exists u.A[\![u]\!]}{\Phi \vdash (\tau[\![A,B]\!])(!u.\lambda x:A[\![u]\!].a[\![u,x]\!])(c) \Rightarrow \Rightarrow \tau_B(c,[u::U[x:A[\![u]\!]].a[\![u,x]\!]) : B,}\quad [u \text{ not in } FV_U(B)],$$

(3) β-∃-*evaluation*. For and all types/propositions A[u], B, and any context Φ such that u is not in $FV_U(B)$, $FV_U(a)$ and t

$$(\beta\exists):\ \frac{\Phi[\ u :: U\ ][\ x : A[\![u]\!]\ ] \vdash c[\![u,x]\!] : B \qquad \Phi \Vdash t :: U \qquad \Phi \vdash a : A[\![u:=t]\!]}{\Phi \vdash (\tau[\![A,B]\!])(!u.\lambda x:A[\![u]\!].c[\![u,x]\!])(\pi[\![A]\!][t](a)) \Rightarrow \Rightarrow \tau_B([t,a]_A,[u::U[x:A[\![u]\!]].a[\![u,x]\!]) \Rightarrow c[\![u:=t]\!][\![x:=a]\!] : B.}$$

(4) ∃-*compatibility*. For all types/propositions A[u], B, and any context Φ,

$$(\xi\exists):\ \frac{\Phi \Vdash t :: U \qquad \Phi \vdash a_1 \Rightarrow a_2 : A[\![u:=t]\!]}{\Phi \vdash [\ t,\ a_1\ ]_A \Rightarrow [\ t,\ a_2\ ]_A : \exists u.A[\![u]\!]}$$

$$(\mu\exists):\ \frac{\Phi[\ u :: U\ ][\ x : A[\![u]\!]\ ] \vdash a_1[\![u,x]\!] \Rightarrow a_2[\![u,x]\!] : B \qquad \Phi \vdash c_1 \Rightarrow c_2 : \exists u.A[\![u]\!]}{\Phi \vdash \tau_B(c_1,[u::U[x:A[\![u]\!]].a_1[\![u,x]\!]) \Rightarrow \Rightarrow \tau_B(c_2,[u::U[x:A[\![u]\!]].a_2[\![u,x]\!]) : B.}\quad [u \text{ not in } FV_U(B)],$$

Proof. (π), (τ): Easy [*exercise*]. (∃i): Assume $\Phi \Vdash t :: U$ and $\Phi \vdash a : A[\![u:=t]\!]$. Then, one has, whenever v is not in $FV_U(A)$,

$$\Phi \vdash \pi[\![A]\!][t](a) \equiv (!u.\lambda x:A[\![u]\!].\lambda f:\forall v.(A[\![u:=v]\!])^-.f[u]x)[t](a) \Rightarrow \Rightarrow (\lambda x:A[\![u:=t]\!].\lambda f:\forall v.(A[\![v]\!])^-.f[t]x)(a) \Rightarrow \Rightarrow \lambda f:\forall v.(A[\![v]\!])^-.f[t]a : \exists u.A[\![u]\!].$$

(∃e): Assume $\Phi[\ u :: \mathcal{U}\ ][\ x : A\ ] \vdash a[\![u,x]\!] : B$ and $\Phi \vdash c : \exists u.A[\![u]\!]$, such that u is not in $FV_{\mathcal{U}}(B)$. Set $b_{\mathcal{A}}[\![f,z]\!] \equiv !v.\lambda y{:}A[\![u:=v]\!].z(f[v]y)$ and $d \equiv (!u.\lambda x{:}A[\![u]\!].a[\![u,x]\!])$, such that

$$[\ f : \forall u.(A[\![u]\!] \sqsupset B)\ ][\ z : B^{-}\ ] \vdash b_{\mathcal{A}}[\![f,z]\!] \equiv \equiv\ !v.\lambda y{:}A[\![u:=v]\!].z(f[v]y) : \ : \forall v.(A[\![v]\!])^{-}$$

$$\Phi \vdash d : \forall u.(A[\![u]\!] \sqsupset B),\quad u \text{ not in } FV_{\mathcal{U}}(B).$$

One has

$$\Phi \vdash (\tau[\![A,B]\!])(!u.\lambda x{:}A[\![u]\!].a[\![u,x]\!])(c) \equiv \equiv (\lambda f{:}\forall u.(A[\![u]\!] \sqsupset B).\lambda x{:}(\exists u.A[\![u]\!]).\not\exists z{:}B^{-}.x(b_{\mathcal{A}}[\![f,z]\!]))dc \Rightarrow\!\!\!\!\!\!\Rightarrow \not\exists z{:}B^{-}.c(b_{\mathcal{A}}[\![f:=d]\!][\![z]\!]) \equiv_{df} e,$$

where $b_{\mathcal{A}}[\![d,z]\!] \equiv !v.\lambda y{:}A[\![u:=v]\!].z(d[v]y)$, i. e.,

$$\Phi \vdash e \equiv \not\exists z{:}B^{-}.c(!v.\lambda y{:}A[\![u:=v]\!].z(d[v]y)) \equiv \equiv \not\exists z{:}B^{-}.c(!v.\lambda y{:}A[\![u:=v]\!].z((!u.\lambda x{:}A[\![u]\!].a[\![u,x]\!])[v]y) \Rightarrow\!\!\!\!\!\!\Rightarrow \not\exists z{:}B^{-}.c(!v.\lambda y{:}A[\![u:=v]\!].z((\lambda x{:}A[\![u:=v]\!].a[\![u:=v]\!][\![x]\!])y) \Rightarrow\!\!\!\!\!\!\Rightarrow \not\exists z{:}B^{-}.c(!v.\lambda y{:}A[\![u:=v]\!].za[\![u:=v]\!][\![y]\!] \equiv_{\alpha} \equiv \not\exists z{:}B^{-}.c(!u.\lambda x{:}A[\![u]\!].za[\![u,x]\!]) : B.$$

(β∃): Assume $\Phi[\ u :: \mathcal{U}\ ][\ x : A\ ] \vdash c[\![u,x]\!] : B$ and $\Phi \Vdash \mathbf{t} :: \mathcal{U}$, $\Phi \vdash a : A[\![u:=\mathbf{t}]\!]$, with u not in $FV_{\mathcal{U}}(B)$, $FV_{\mathcal{U}}(a)$ and $\mathbf{t}$. Then, one has, using (2),

$$\Phi \vdash (\tau[\![A,B]\!])(!u.\lambda x{:}A[\![u]\!].c[\![u,x]\!])(\pi[\![A]\!][\mathbf{t}](a)) \Rightarrow\!\!\!\!\!\!\Rightarrow \not\exists z{:}B^{-}.(\lambda f{:}\forall u.(A[\![u]\!])^{-}.f[\mathbf{t}]a)(!u.\lambda x{:}A[\![u]\!].zc[\![u,x]\!]) \Rightarrow\!\!\!\!\!\!\Rightarrow \not\exists z{:}B^{-}.(!u.\lambda x{:}A[\![u]\!].zc[\![u,x]\!])[\mathbf{t}]a \Rightarrow\!\!\!\!\!\!\Rightarrow \not\exists z{:}B^{-}.(\lambda x{:}A[\![u:=\mathbf{t}]\!].z(c[\![u:=\mathbf{t}]\!][\![x]\!]))a \Rightarrow\!\!\!\!\!\!\Rightarrow \not\exists z{:}B^{-}.z(c[\![u:=\mathbf{t}]\!][\![x:=a]\!]) \Rightarrow\!\!\!\!\!\!\Rightarrow c[\![u:=\mathbf{t}]\!][\![x:=a]\!] : B.$$

(ξ∃), (μ∃): From the compatibility rules for the primitive proof-operations. □

13.9 Remark.

(1) The rules ($\vee i_1$), ($\vee i_2$), ($\vee e$) are analogues of the "disjoint sum" type-assignment rules of Martin-Löf's [1984] constructive type theory [**CST**]. The "∨-evaluation" rules ($\beta\vee_i$) [i := 1,2] yield also "closed" variants of corresponding ∨-evaluation rules in **CST**. Cf. the family [$D[\![A,B,C]\!]$, $J_1[\![A,B]\!]$, $J_2[\![A,B]\!]$] above with Howard's [1980] "weak disjunction" proof-operations.

(2) The rules (∃i), (∃e), (β∃) are analogues of the "generalized sum" rules of Martin-Löf's constructive type theory, restricted to a first-order (logic) setting. Compare also the "closed" (combinator) representation [$\pi[\![A]\!]$, $\tau[\![A,B]\!]$] with Howard's [1980] "weak existence" proof-operations.

13.10 Remark (⊕-*proof-operations in* **pλ**∀!).

(1) In **pλ**∀(!), one can define disjunction alternatively, in terms of a primitive conjunction (&). Set $A \oplus B := (A^- \,\&\, B^-)^-$. Then, we have, for all types/propositions A, B, C, in **pλ**∀(!),

$$(1°1)\quad \mathbf{J}_{\langle 1\rangle}[\![A,B]\!] := \lambda x{:}A.\lambda z{:}(A^-\& B^-).\mathbf{p}_1(z)x,$$
$$(1°2)\quad \mathbf{J}_{\langle 2\rangle}[\![A,B]\!] := \lambda y{:}B.\lambda z{:}(A^-\& B^-).\mathbf{p}_2(z)y,$$
$$(2°)\quad \mathbf{D}_{\langle\rangle}[\![A,B,C]\!] := \lambda x{:}(A\supset C).\lambda y{:}(B\supset C).\lambda h{:}(A\oplus B).\forall z{:}C^-.ha',$$

where $a' \equiv a'[\![x,y,z]\!] \equiv \langle\, \lambda x'{:}A.z(xx'),\ \lambda y'{:}B.z(yy')\,\rangle$.

(2) One can check easily the fact that, for all types/propositions A, B, C, one has, in **pλ**∀(!),

$$(\mathbf{J}_{\langle 1\rangle}){:}\ [\ ]\vdash \mathbf{J}_{\langle 1\rangle}[\![A,B]\!] : A\supset (A\oplus B),$$
$$(\mathbf{J}_{\langle 2\rangle}){:}\ [\ ]\vdash \mathbf{J}_{\langle 2\rangle}[\![A,B]\!] : B\supset (A\oplus B),$$
$$(\mathbf{D}_{\langle\rangle}){:}\ [\ ]\vdash \mathbf{D}_{\langle\rangle}[\![A,B,C]\!] : A\supset C\supset .B\supset C\supset (A\oplus B\supset C).$$

and rules $(\oplus i_i)$, $(\oplus e)$, $(\beta\oplus_i)$, analogous to $(\vee i_i)$, $(\vee e)$, $(\beta\vee_i)$, resp. [i := 1,2], as well as appropriate ⊕-compatibility rules.

(3) Moreover, in view of the "surjectivity" rule (η&), one has also, in **pλ**∀(!), a kind of *extensional behavior for the* ⊕-*proof-operations*: for all types/propositions A, B, and any context Φ,

$$(\eta\oplus=){:}\ \frac{\Phi\vdash c : A\oplus B}{\Phi\vdash (\mathbf{D}_{\langle\rangle}[\![A,B,(A\oplus B)]\!])(\mathbf{J}_{\langle 1\rangle}[\![A,B]\!])(\mathbf{J}_{\langle 2\rangle}[\![A,B]\!])(c) = c.}$$

It is easy to see that (η⊕=) is derivable in **pλ**∀(!) only because applications of the form Φ[x:C] ⊢ fx make sense, in general, for some type C and p-terms f such that Φ ⊢ f : A ⊕ B, i. e., ultimately, because ⊕ is defined in terms of connectives whose associated proof-operations are extensional (here: ⊃, &). With a primitive ⊕ and primitive ⊕-proof operations like $\mathbf{D}_{\langle\rangle}$ and $\mathbf{J}_{\langle i\rangle}$ [i := 1,2], (η⊕=) does not follow from $(\beta\oplus_i)$, [i := 1,2] and rules of **λ**∀(!) only. The ⇒-analogue of (η⊕=) is also *not* available in **pλ**∀(!).

(4) We can define, in **pλ**∀(!), the analogous proof-notation for the ⊕-operations, from the family of proof-combinators [$\mathbf{D}_{\langle\rangle}[\![A,B,C]\!]$, $\mathbf{J}_{\langle 1\rangle}[\![A,B]\!]$, $\mathbf{J}_{\langle 2\rangle}[\![A,B]\!]$], as, e. g., by,

$$\mathbf{D}_{\langle c\rangle}(c,\ [x{:}A].a[\![x]\!],\ [y{:}B].b[\![y]\!]) := \forall z{:}C^-.c\langle a'[\![z]\!], b'[\![z]\!]\rangle,$$
$$\text{where } a' \equiv \lambda x'{:}A.za[\![x']\!],\ b' \equiv \lambda y'{:}B.zb[\![y']\!],$$

$$\mathbf{j}_{\langle i\rangle}[\![A,B]\!](c) := \lambda z{:}(A^-\&B^-).\mathbf{p}_i(z)c\ [i := 1,2].$$

Then it is easy to see that, *mutatis mutandis*, rules similar to those of **Theorem 13.7** (2)-(4) above are available for the latter notation, as well.

13.11 Remark (*The Heyting* ***falsum****-rule*).

Recall that, for all types/propositions A, we had, in λ𝒻(!),

(ω): [*Ex falso quodlibet*]: [ ] ⊢ ω[[A]] ≡$_{df}$ λx:⊥.𝒻y:A^{-}.x : ⊥ ⊐ A.

Ultimately, the Heyting **falsum**-rule becomes, in λ𝒻(!),

($\omega_{\mathbf{A}}$i): Φ ⊢ a : ⊥ ==> Φ ⊢ ω[[A]](a) =» $\omega_{\mathbf{A}}$(a) : A.

Heyting proof-calculi. With this, we have already the ingredients necessary for the simulation of (a variant of) the Heyting proof-calculus for **HQ**, the first-order "intuitionistic" logic.

A natural variant of the *Heyting first-order proof-calculus*, **λH!** say, can be described as follows (cf., e.g., [Martin-Löf 1984]):

13.12 Definition (*A Heyting proof-calculus* **λH!**).

(1) "Intuitionistic" *proof-syntax*:
  (11) **λH!**-*types*: the *type-syntax* of **λH!** is isomorphic to a provability language L_{HQ} := L[⊐,∧,∨,∀,∃] (or L[⊐,&,⊕,∀,∃], with & for ∧ and ⊕ for ∨). [**NB**: **T**, ⊥ are primitive types in L_{HQ}.].
  (12) **λH!**-*terms*: the *term-syntax* of **λH!** are of the form:
    (1°) λx:A.a[[x]], fa,
    (2°) <a, b>$_{\mathbf{A},\mathbf{B}}$, $\mathbf{p}_i$[[A,B]](a) [i := 1,2],
    (3°) $\mathbf{j}_i$[[A,B]](a) [i := 1,2], $\mathbf{D}_{\mathbf{C}}$(c,[x:A].a[[x]],[y:B].b[[y]]),
    (4°) !u.a[[u]], f[t],
    (5°) [t, a]$_{\mathbf{A}}$, $\dot{t}_{\mathbf{B}}$(c,[u::$\mathcal{U}$[x:A[[u]]].a[[u,x]]),
    (6°) $\omega_{\mathbf{A}}$(a),
    where f, a, b, c are **λH!**-terms, **t** is a $\mathcal{U}$-term and A, B, C are types/propositions in L_{HQ}.

(2) *Stratification*: context rules: <Ω>, <I>, <K>, <K∀> [as for λ𝒻!].

(3) *Stratification*: type-assignment rules:
  (31) ⊐-rules: (⊐iλ), (⊐e),
  (32) ∧-rules: (∧i), (∧e_i) [i := 1,2],
  (33) ∨-rules: (∨i_i) [i := 1,2], (∨e),
  (34) ∀-rules: (∀i), (∀e),
  (35) ∃-rules: (∃i), (∃e),
  (36) ⊥-rule: ($\omega_{\mathbf{A}}$i).

(4) Evaluation rules:
  (41) ⊐-rules: (β⊐λ), (η⊐λ),
  (42) ∧-rules: (β∧$_i$) [i := 1,2],
  (43) ∨-rules: (β∨$_i$) [i := 1,2],
  (44) ∀-rules: (β∀), (η∀),
  (45) ∃-rule: (β∃),
  (46) No ⊥-rule.

(5) Compatibility rules: [as expected].

In fact, the explicit type-parametrization in proof-terms of the form $\langle a, b\rangle_{A,B}$, $[t,a]_A$, $p_{i:3}[\![A,B]\!](c)$, $j_i[\![A,B]\!](c)$ [i := 1,2], is not necessary; we have used this syntax in order to make transparent the fact that the necessary rules are already available in **λ𝔤!**.

13.13 Remark.

(1) There are several variants of the Heyting calculus, differing in their "extensionality type":

- the variant **λH!** above can be interpreted, in the obvious way, in **λ𝔤!**, (**λH!** is not "&-extensional");
- the "&-extensional" ["surjective"] variant **pλH!** := **λH!** + (η∧) can be interpreted, *mutatis mutandis*, in **pλ𝔤!** [(η∧) or (η&), with & primitive, is "surjectivity of pairing"; this extends the group (42) of rules above] ;
- however, in the current proof-theoretic literature only an "intensional" variant, **λβH!** say, without (η⊃λ), (η∧), is explicitly considered.

(2) Putting aside [Martin-Löf 1984], the proof-theoretic literature around Heyting's first-order logic **HQ** seems to credit **λβH!** as a kind of *kernel standard formalization* for the Heyting first-order proofs. Beyond **λβH!**, one usually considers also *ad hoc* "commutative" [or "permutative"] ⊕- and ∃-reduction rules, as well as a set of patching "⊥-rules", whose rôle is to insure *confluence* properties (cf., e. g., [Troelstra 1973] **4.1.3**). The "extensional" ("η-type") assumptions are usually ignored.

From the above, it is clear that **(p)λH!** is a fragment of **(p)λ𝔤!**, in the definitional sense. This can be made precise, by defining an appropriate *translation* of the **(p)λH!**-proof-syntax into the **(p)λ𝔤!**-proof-syntax.

13.14 Fact.

Let $\vdash_{\langle p\rangle H}$, and $\vdash_{\langle p\rangle C}$ resp. stand for derivability in **(p)λH!**, and **(p)λ𝔤!**, resp. and let $(\ldots)^\circ$ be a mapping of the primitive type- and p-term syntax of **(p)λH!** into **(p)λ𝔤!**, such that $(\ldots)^\circ$

- preserves identically the [(∧)-∨-∃-⊥]-free fragments into the 𝔤-free part of **(p)λ𝔤!**, and
- sends [(∧)-∨-∃-⊥]-primitives into corresponding notions, as *defined* in **(p)λ𝔤!**,

[that is, $(\ldots)^\circ$ maps t-statements into t-statements]. Then, for all p-terms a, b in Term[**(p)λH!**],

(1) $\Phi \vdash_{\langle p\rangle H} a : A \;\Longrightarrow\; \Phi^\circ \vdash_{\langle p\rangle C} a^\circ : A^\circ$,
(2) $\Phi \vdash_{\langle p\rangle H} a \twoheadrightarrow b : A \Longrightarrow \Phi^\circ \vdash_{\langle p\rangle C} a^\circ \twoheadrightarrow b^\circ : A^\circ$,
(3) $\Phi \vdash_{\langle p\rangle H} a = b : A \Longrightarrow \Phi^\circ \vdash_{\langle p\rangle C} a^\circ = b^\circ : A^\circ$.

Proof. By a direct inspection of the rules of $(\mathbf{p})\lambda\mathbf{H}!$ and results established earlier. In the above, $(\mathbf{p})\lambda\mathbf{H}!$ has been so described such as to make $(\ldots)^{\circ} : (\mathbf{p})\lambda\mathbf{H}! \longrightarrow (\mathbf{p})\lambda\mathfrak{G}!$ into a mere syntactic transcription of data. □

From this [viz., **13.14** (3)] and $\mathbf{Cons}[(\mathbf{p})\lambda\mathfrak{G}!]$ [**6.23** and **13.4** (3)], one has immediately a consistency result.

13.15 Theorem.

$\mathbf{Cons}[(\mathbf{p})\lambda\mathbf{H}!]$.

13.16 Remark.

(1) However, several "permutative" rules, as well as some $\bot$-rules (cf. [Troelstra 1973], **4.3**) are *not* $(\mathbf{p})\lambda\mathfrak{G}!$-derivable, in the definitional sense above (some "permutative" rules cannot even be obtained if $\Rightarrow$ is replaced by equality $=$). So, in fact, $\neg\mathbf{CR}[(\mathbf{p})\lambda\mathbf{H}!]$.

(2) In view of $\mathbf{SN}[(\mathbf{p})\lambda\mathfrak{G}!]$ [**11.28** and **13.4** (3)], **13.14** (2) above insures also the required normalization properties for $(\mathbf{p})\lambda\mathbf{H}!$. If *an independent technique* can be found, guaranteeing the *unicity of normal forms* [**UN**] in $(\mathbf{p})\lambda\mathbf{H}!$, this should make superfluous the considerations on "commutative" reductions, etc. This approach is actually taken in [Martin-Löf 1984].

13.17 Remark (*Extensionality for* $\oplus$- *and* $\exists$-*proof-operations*).

(1) One can also add to $\mathbf{p}\lambda\mathbf{H}!$ extra "extensionality" assumptions for $\oplus$- and $\exists$-proof-operations:

$$(\eta\oplus_H):\quad \frac{\Phi \vdash f : A \oplus B \supset C}{\Phi \vdash \lambda z{:}(A\oplus B).\mathbf{D}_{\langle\langle\rangle\rangle}(z,[x{:}A].f(j_{\langle 1\rangle}(x)),[y{:}B].f(j_{\langle 2\rangle}(y))) \Rrightarrow f},$$

[x, y not free in f],

where $j_{\langle i\rangle}(c)$ is shorthand for $j_{\langle i\rangle}[\![A,B]\!](c)$ [$i := 1,2$].

$$(\eta\exists_H):\quad \frac{\Phi \vdash f : \exists u.A[\![u]\!] \supset B}{\Phi \vdash \lambda z{:}\exists u.A[\![u]\!].\tau_B(z,\ [u::\mathcal{U}][x{:}A[\![u]\!]].f[u,x]_A) \Rrightarrow f.}$$

[u and x not free in f].

(2) With $I[\![A\oplus B]\!] := \lambda z{:}(A\oplus B).z$, one has $[\ ] \vdash I[\![A\oplus B]\!] : A\oplus B \supset A\oplus B$. Then, by $(\eta\oplus_H)$, we get, in $\mathbf{p}\lambda\mathbf{H}!$,

$$[\ ] \vdash \lambda z{:}(A\oplus B).\mathbf{D}_{\langle\langle\rangle\rangle}(z,[x{:}A].j_{\langle 1\rangle}[\![A,B]\!](x),[y{:}B].j_{\langle 2\rangle}[\![A,B]\!](y)) = I[\![A\oplus B]\!],$$

whence, if x and y are not free in c,

$$(\eta\oplus{=}):\quad \frac{\Phi \vdash c : A \oplus B}{\Phi \vdash \mathbf{D}_{\langle C\rangle}(c,\ [x{:}A].\,j_{\langle 1\rangle}[\![A,B]\!](x),\ [y{:}B].\,j_{\langle 2\rangle}[\![A,B]\!](y)) = c.}$$

(3) Analogously, with $I[\![\exists u.A]\!] := \lambda z{:}(\exists u.A[\![u]\!]).z$ and $B \equiv \exists u.A[\![u]\!]$ one has, in **pλH!**, $[\ ] \vdash I[\![\exists u.A]\!] : \exists u.A[\![u]\!] \supset \exists u.A[\![u]\!]$. Then, by $(\eta\exists_H)$, $[\ ] \vdash \lambda z{:}\exists u.A[\![u]\!].\tau_B(z,[u::\mathcal{U}[x{:}A[\![u]\!]].[u,x]_A) = I[\![\exists u.A]\!]$, whence, if u and x are not free in c,

$$(\eta\exists{=}):\quad \frac{\Phi \vdash c : \exists u.A[\![u]\!]\ [\ \equiv_{df} B\ ]}{\Phi \vdash \tau_B(c,\ [u::\mathcal{U}[x{:}A[\![u]\!]].[u,x]_A) = c.},$$

(4) This suggests that one could have had, alternatively, slightly weaker ⊕- and ∃-extensionality assumptions:

$$(\eta\oplus):\quad \frac{\Phi \vdash c : A \oplus B}{\Phi \vdash \mathbf{D}_{\langle C\rangle}(c,\ [x{:}A].\,j_{\langle 1\rangle}[\![A,B]\!](x),\ [y{:}B].\,j_{\langle 2\rangle}[\![A,B]\!](y)) \Rightarrow c},$$

[x and y not free in c],

$$(\eta\exists):\quad \frac{\Phi \vdash c : \exists u.A[\![u]\!]\ [\ \equiv_{df} B\ ]}{\Phi \vdash \tau_B(c,\ [u::\mathcal{U}[x{:}A[\![u]\!]].[u,x]_A) \Rightarrow c.},\ [u,\ x \text{ not free in } c],$$

(5) In the end, one can define *fully extensional* Heyting proof-calculi [Rezus 1986a],

$$\mathbf{p\lambda\eta H!} := \mathbf{p\lambda H!} + (\eta\oplus) + (\eta\exists) \text{ and}$$
$$\mathbf{p\lambda\eta_* H!} := \mathbf{p\lambda H!} + (\eta\oplus_H) + (\eta\exists_H).$$

It is immediate that **pληH!** contains **pλH!** and is a subsystem of $\mathbf{p\lambda\eta_* H!}$. However, the classical proof-calculus **pλℐ!** does *not* contain $\mathbf{p\lambda\eta}_{\langle *\rangle}\mathbf{H!}$, in the definitional sense above.

H. P. Barendregt
74 *Pairing without conventional restraints*, **Zeitschrift für math. Logik und Grundlagen der Mathematik 20**, 1974, pp. 289-306.
84 **The Lambda Calculus: Its Syntax and Semantics**, North Holland, Amsterdam, etc. 1981[1], 1984[2] (revised).
H. Barendregt and A. Rezus
83 *Semantics for* ***Classical Automath*** *and related systems*, **Information and Control 59**, 1983, pp. 127-147. [Rev. **MR 86g**:03100 and **Zbl 564.68060**.]
N. G. de Bruijn
80 *A survey of the* ***Automath*** *project*, in: J. P. Seldin and J. R. Hindley (eds.) **To H. B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism**, Academic Press, London, etc., 1980, pp. 579-606.
87 *Generalizing* ***Automath*** *by means of a lambda-typed lambda-calculus*, in: D. W. Kueker *et al.* (eds.) **Mathematical Logic and Theoretical Computer Science**, Marcel Dekker, New York, 1987, pp. 71-92 [**Lecture Notes in Pure and Applied Mathematics 106**].
M. W. Bunder and R. K. Meyer
9* *Condensed detachment and combinators*, The Australian National University, **The Automated Reasoning Project**, Canberra ACT, Report **TR-ARP-8-1988**, September 20, 1988, 78 pp. [to appear]
A. Church
56 **Introduction to Mathematical Logic (1)**, Princeton UP, Princeton NJ, 1956.
H. B. Curry
63 **Foundations of Mathematical Logic**, McGraw Hill, New York 1963. (Reprints: Dover Publications Inc., New York 1977[R], 1984[R].)
H. B. Curry *et alii*
58 **Combinatory Logic (1)**, North Holland, Amsterdam, etc., 1958.
72 **Combinatory Logic (2)**, North Holland, Amsterdam, etc., 1972.
D. T. van Daalen
80 **The Language Theory of Automath**, [Ph. D. Diss., University of Eindhoven 1980], Wibro, Helmond 1980, 307 pp.
F. B. Fitch
52 **Symbolic Logic: An Introduction**, Ronald Press, New York 1952.
G. Gentzen
33 *Über das Verhältnis zwischen intuitionistischer und klassischer Logik*, **Arch. math. Logik 16**, 1974, pp. 119-132. (Cf. also: M. E. Szabo (ed.) **Gerhard Gentzen: Collected Papers**, North Holland, Amsterdam, etc., 1969, pp. 53-67. Paper of 1933.)
35 *Untersuchungen über das logische Schliessen*, **Mathematische Zeitschrift 39**, 1935, pp. 176-210 and 405-431. (Cf. also: M. E. Szabo (ed.) **Gerhard Gentzen: Collected Papers**, North Holland, Amsterdam, etc., 1969, pp. 68-131.)
J.-Y. Girard *et alii* [= P. Taylor and Y. Lafont]
89 **Proofs and Types**, Cambridge UP, Cambridge UK 1989 [**Cambridge Tracts in Theoretical Computer Science 7**]. (Revised lecture notes for a course given in 1986-87, at the Univ. of Paris 7.)
V. I. Glivenko
28 *Sur la logique de M. Brouwer*, **Bull. Cl. Sci., Académie Royale de Belgique [5] 14**, 1928, pp. 225-228.

[111]

K. Gödel
33 *Zur intuitionistischer Arithmetik und Zahlentheorie*, in: **Ergebnisse eines mathematischen Kolloquiums 4**, 1933, pp. 34-38. (Cf. also: S. Feferman *et al.* (eds.) **Kurt Gödel: Collected Works 1**, Oxford UP, Oxford UK, etc. 1986, pp. 286-295).

G. H. Helman
83 *An interpretation of classical proofs*, **Journal of Philosophical Logic 12**, 1983, pp. 39-71.
87 *On the equivalence of proofs involving identity*, **Notre Dame Journal of Formal Logic 28**, 1987, pp. 297-321.

A. Heyting
30 *Die formalen Regeln der intuitionistischen Logik*, **Sitzungsberichte der Preussischen Akademie von Wissenschaften,** *Phys.-math. Klasse*, 1930, pp. 42-56.

J. R. Hindley
89 **BCK-***combinators and linear* λ*-terms have types*, **Theoretical Computer Science 64**, 1989, pp. 97-105.

J. R. Hindley and D. Meredith
90 *Principal type schemes and condensed detachment*, **Journal of Symbolic Logic 55**, 1990, pp. 90-105.

J. R. Hindley and J. P. Seldin
86 **Introduction to Combinators and λ-Calculus**, Cambridge UP, Cambridge UK 1986 [**London Mathematical Society Student Texts 1**].

W. A. Howard
80 *The Formulae-as-Types notion of construction*, in: J. P. Seldin and J. R. Hindley (eds.) **To H. B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism**, Academic Press, London, etc., 1980, pp. 479-490 (written in 1969).

I. Johansson
36 *Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus*, **Compositio Mathematica 4**, 1936, pp. 119-136 (appeared in 1935).

L. S. [van Benthem] Jutting
79 **Checking Landau's "Grundlagen" in the Automath System**, [Ph. D. Diss., University of Eindhoven 1987], Mathematisch Centrum [**CWI**], Amsterdam 1979, 120 pp. [**MC Tracts 83**]:[**CWI**].

J. Kalman
83 *Condensed detachment as a rule of inference*, **Studia Logica 42**, 1983, pp. 443-451.

J. W. Klop
80 **Combinatory Reduction Systems**, [Ph. D. Diss., University of Utrecht 1980], Mathematisch Centrum [**CWI**], Amsterdam 1980, iv + 317 pp. [**MC Tracts 127**]:[**CWI**]

A. N. Kolmogorov
25 *O principe* **tertium non datur** [*On the principle of the excluded middle*] [Russian], **Mat. Sbornik 32**, 1925, pp. 646-667 (Cf. also: J. van Heijenoort, (ed.), **From Frege to Gödel, A Source Book in Mathematical Logic 1879-1931**, Harvard UP, Cambridge Mass., 1967, 1970[R], pp. 414-437.)

S. Kuroda
51 *Intuitionistische Untersuchungen der formalistischen Logik*, **Nagoya Math. Journal 2**, 1951, pp. 35-47.

M. H. Löb
76 *Embedding first-order predicate logic in fragments of intuitionistic logic*, **Journal of Symbolic Logic 41**, 1976, pp. 705-718.

P. Lorenzen
55 **Einführung in die operative Logik und Metamathematik**, Springer Verlag, Berlin, etc. 1955, 1969².

P. Lorenzen and K. Lorenz (eds.)
78 **Dialogische Logik**, Wiss. Buchgesellschaft, Darmstadt, 1978.

J. Łukasiewicz
48 *The shortest axiom of the implicational calculus of propositions*, **Proceedings of the Royal Irish Academy 52A**, 3, 1948, pp. 25-33. (Cf. also: L. Borkowski (ed.) **Jan Łukasiewicz: Selected Works**, North Holland, Amsterdam, etc. 1970, pp. 295-305.)

P. Martin-Löf
84 **Intuitionistic Type Theory**, Bibliopolis, Naples 1984 [**Studies in Proof Theory**]. (Lectures given in Padova, in June 1980.)

C. A. Meredith and A. N. Prior
63 *Notes on the axiomatics of the propositional calculus*, **Notre Dame Journal of Formal Logic 4**, 1963, pp. 171-187.

D. Meredith
77 *In Memoriam: Carew Arthur Meredith (1904-1970)*, **Notre Dame Journal of Formal Logic 18**, 1977, pp. 513-516.

R. Montague and L. Henkin
56 *On the definition of 'formal deduction'*, **Journal of Symbolic Logic 21**, 1956, pp.129-136.

D. Prawitz
65 **Natural Deduction**, Almqvist & Wiksell, Stockholm, etc., 1965.
71 *Ideas and results in proof theory*, in: J. E. Fenstad (ed.) **Proceedings of the Second Scandinavian Logic Symposium**, North Holland, Amsterdam, etc., 1971, pp. 235-307.

A. Rezus
81 **Lambda-conversion and Logic**, [Ph. D. Diss., University of Utrecht 1981], Elinkwijk, Utrecht 1981, ix + 196 pp. [Rev. **Libertas Mathematica 2**, 1982, pp. 182-185, cf. **Zbl 493**.03003.]
82 *On a theorem of Tarski*, **Libertas Mathematica** [Arlington TX] **2**, 1982, pp. 62-95. [Rev. **MR 83c**:03019 and **Zbl 481**.03010.]
83 **Abstract Automath**, Mathematisch Centrum, Amsterdam 1983, vi + 188 pp. [ISBN 90-6196-256-0][**MC Tracts 160**]:[**CWI**]. [Rev. **MR 84j**:03030 and **Zbl 507**.03001.]
83a *Constructive type theory and functional programming*, Lectures for the **Algemeen Informatica Colloquium**, University of Nijmegen, Department of Computer Science, December 1983 (cf. [1986]).
86 *Semantics of constructive type theory*, **Libertas Mathematica** [Arlington TX] **6**, 1986, pp. 1-82. [Rev. **MR 88a**:03034 and **Zbl 632**.03047.]
86a **Impredicative Type Theories** [**Syntax, Model Theory and Formal Pragmatics**], University of Nijmegen, Department of Computer Science, Report **KUN-WN-CS TR-85-1986**, June 1986, 288 pp.
86b *Automath: syntax and semantics*, Talk delivered at the Institute of Advanced Studies, The Australian National University, The **Automated Reasoning Project**, Canberra ACT, September 1986.

87 *Propositions-as-types revisited* [*Higher-order constructive type theory*], University of Nijmegen, Department of Computer Science, Report **KUN-WN-CS TR-97-1987**, February 1987, 91 pp.

87a *Varieties of generalized functionality*, University of Nijmegen, Department of Computer Science, Report **KUN-WN-CS TR-102-1987**, February 1987, 106 pp. (with P. J. de Bruin).

87b *Constructions and propositional types*, University of Nijmegen, Department of Computer Science, Internal Report **KUN-WN-CS 87-1-1987**, April 1987, 64 pp.

87c *Generalized typed lambda-calculi: recent advances*, Paper contributed to the **XII-ème Congrès de L'Académie Roumano-Américaine [ARA]** (*III-ème Section: Mathématiques, Physique*), Sorbonne, France, June 22-27, 1987.

88 *A type-theoretic approach to classical and non-classical logics*, Talk for the "Jumelage" Workshop **Typed Lambda-Calculi**, University of Nijmegen, November 14-18, 1988.

89 *What is a classical proof?*, Talk for a **GMD-Colloquium**, held at the *Forschungsstelle für Programmstrukturen* [Computing Department], University of Karlsruhe, May 1989.

89a *The type theory of classical logic*, Talk for the **Informatica-Colloquium**, University of Groningen, June 1989.

M. Schönfinkel

24 *Über die Bausteine der mathematischen Logik*, **Mathematische Annalen 92**, 1924, pp. 305-316. (Cf. also: J. van Heijenoort (ed.) **From Frege to Gödel, A Source Book in Mathematical Logic 1879-1931**, Harvard UP, Cambridge Mass., 1967, 1970^{R}, pp. 355-366.)

J. P. Seldin

79 *Progress report on generalized functionality*, **Annals of Mathematical Logic 17**, 1979, pp. 29-59.

87 ***Mathesis*: The Mathematical Foundation of *Ulysses***, Rome Air Development Center, AFSC, Griffiss AF Base, New York, **RADC: TR-87-223** Interim Report, November 1987, 160 pp.

S. Stenlund

72 **Combinators, λ-terms and Proof Theory**, D. Reidel, Dordrecht, 1972.

W. W. Tait

67 *Intensional interpretations of functionals of finite type I*, **Journal of Symbolic Logic 32**, 1967, pp. 198-212. (No more.)

M. Takahashi

89 *Parallel reductions in λ-calculus*, **Journal of Symbolic Computation 7**, 1989, pp. 113-123.

A. S. Troelstra (ed.)

73 **Metamathematical Investigation of Intuitionistic Arithmetic and Analysis**, Springer Verlag, Berlin, etc. 1973 [**Lecture Notes in Mathematics 344**] (*Corrections and additions*: **Report 74-16**, Univ. of Amsterdam, Department of Mathematics, November 1974).

A. S. Troelstra

86 *Strong normalization for typed terms with surjective pairing*, **Notre Dame Journal of Formal Logic 27**, 1986, pp. 547-550.

A. S. Troelstra and D. van Dalen

88 **Constructivism in Mathematics, An Introduction (1-2)**, North-Holland, Amsterdam, etc. 1988.

[114]