INFORMATION AND COMPUTATION **124**, 32–61 (1996)
Article No. 0004

A Typed Pattern Calculus

DELIA KESNER AND LAURENCE PUEL

CNRS and Laboratoire de Recherche en Informatique, Bat. 490, Université de Paris-Sud, 91405 Orsay Cedex, France
E-mail: {kesner,puel}@lri.fr

AND

VAL TANNEN

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389, USA
E-mail: val@cis.upenn.edu

The theory of programming with pattern-matching function definitions has been studied mainly in the framework of first-order rewrite systems. We present a typed functional calculus that emphasizes the strong connection between the structures of whole pattern definitions and their types. In this calculus, type-checking guarantees the absence of runtime errors caused by non-exhaustive pattern-matching definitions. Its operational semantics is deterministic in a natural way, without the imposition of ad hoc solutions such as clause order or "best fit". In the spirit of the Curry–Howard isomorphism, we design the calculus as a computational interpretation of the Gentzen sequent proofs for the intuitionistic propositional logic. We prove the basic properties connecting typing and evaluation: subject reduction and strong normalization. We believe that this calculus offers a rational reconstruction of the pattern-matching features found in successful functional languages. © 1996 Academic Press, Inc.

CONTENTS

1. INTRODUCTION

Programming with pattern-matching function definitions is a very attractive feature that accounts for much of the popularity of functional languages such as Hope [4, 7], SML [18], Miranda [24], Caml Light [17], and Haskell [11]. So far only those aspects of pattern matching that fit in the framework of first-order rewrite systems have been studied (e.g., [12, 22]). We find it desirable to understand pattern constructs as well as we now understand Algol-like and functional programming constructs. A crucial role in understanding these has been played by the lambda calculus and its various type disciplines. We propose a corresponding "calculus" that models programs with pattern-matching.

One of our goals is to be able to more or less directly represent function definitions such as the following ones in, for example, ML:

```
- fun suffixlist []=[[]]
=| suffixlist(z as _::l)=z::suffixlist(l);
(* eg. suffixlist [1,2,3]
               =[[1,2,3],[2,3],[3],[]] *)
val suffixlist=fn : 'a list → 'a list list
-
- fun flatten []=[]
=| flatten([]::L)=flatten L
=| flatten((x::l)::L)=x::flatten(l::L);
(* eg. flatten [[1,2],[],[3,4,5]]
                         =[1,2,3,4,5] *)
val flatten=fn : 'a list list → 'a list
-
- fun merge ([],z2)=z2
=| merge (z1,[])=z1
=| merge (x1::l1,x2::l2)
=                 =x1::x2::merge(l1,l2);
(* eg. merge ([1,2,3,4],[5,6])
                       =[1,5,2,6,3,4] *)
val merge=fn : 'a list * 'a list → 'a list
```

In existing languages, the typing and operational semantics of pattern-matching definitions are treated like those of sets of rewrite rules [15]. Each rewrite rule (definition

0890-5401/96 $12.00

Copyright © 1996 by Academic Press, Inc.
All rights of reproduction in any form reserved.

clause) is typed, but there is no notion of *globally* typing the set of clauses of a definition. The fact that pattern overlapping (redundancy) and pattern exhaustiveness are treated in an ad hoc manner constitutes a symptom of this problem. The problem is that the actual operational semantics of these languages does not have a concept of "pattern of a given type" that would cover all the possible constructors. More specifically, the separation into clauses is not related to the treatment of sum types.

The treatment of overlapping patterns faces two constraints: we must stay within deterministic semantics, and equivalence of program phrases is undecidable, so one cannot check redundant patterns for compatibility. On the other hand, irredundancy itself and exhaustiveness are decidable properties for a given set of patterns, so one could, in principle, forbid redundant and inexhaustive sets of patterns. In fact, the static semantics of SML does just that [18], but not through typing constraints, and in the same paragraph where this is stipulated, the authors add that redundant and inexhaustive patterns should be compiled with warnings to the programmers (for example, the compilers SML of NJ and Caml-Light of 78 issue such warnings). This approach is motivated practically, since the operational semantics of redundant patterns is resolved in SML (as well as in Miranda and Haskell), brutally but sensibly by using the order in which the clauses were written, leading to useful programming techniques, when not abused. In Hope, the operational semantics uses a more complicated "best-fit" proviso whose practical impact is unclear. Both these solutions stay within the framework of first-order rewrite systems. We should also add that virtually all compilation techniques for pattern-matching lead to exhaustive and irredundant matching trees in the object code. Beyond these practical aspects, there remains the issue of whether the semantics of sets of patterns can be explained in a global and typed manner.

Thus, in a calculus that models programs with pattern-matching, overlapping and exhaustiveness should really be typing issues, and there should be an intimate connection between the structures of the whole pattern definitions and their types. Crucially, this requires a new idea for the concept of pattern of sum type.

Searching for a uniform paradigm that would provide a rational reconstruction of pattern-matching features, we have been inspired by the Curry–Howard isomorphism. The isomorphism explains the simply typed lambda calculus as a computational interpretation of the natural deduction proofs for intuitionistic propositional logic. For reference, Appendix A contains this interpretation; reviewing it might be helpful for understanding our approach.

The constructor terms of the simply typed lambda calculus correspond to those natural deduction proofs built using only the introduction rules. In the languages we have mentioned, patterns have the same syntax as constructor terms, but operationally they are *dual* to them. There is one formulation of logical proof systems in which this duality is made clear, and this is Gentzen's sequent proof system. Thus, in the spirit of the Curry–Howard isomorphism, our idea is to design a typed pattern calculus as a computational interpretation of the Gentzen sequent proofs for the intuitionistic (actually minimal) propositional logic. In a sense, we are looking for new syntax and we use the sequent proof rules for inspiration. The sequent proof system has right rules, which are the same as the introduction rules of natural deduction, left rules, which we use to build nested patterns as variable generalizations, and the cut rule, which is interpreted as a general *let* construct and where all computations originate. The left contraction and left weakening rules correspond to the layered and wildcard patterns in ML and Haskell.

Abramsky [2] gives a term assignment for the intuitionistic sequent proofs, but the terms are the same as those that arise from the term assignment to natural deduction proofs. His interpretation of the sequent proof rules gives an alternative, but equivalent, set of type-checking rules. In the same paper, Abramsky gives a term assignment for the sequent proofs of intuitionistic propositional linear logic. He notes that the left rules correspond to pattern-matching constructs, but the resulting syntax does not allow nested patterns as generalizations of variables. Gallier [8] gives a novel term assignment to sequent proofs and describes the cut-elimination rules on it, but his syntax also does not build nested patterns. Van Oostrom [20] studies a lambda calculus with patterns which are arbitrary lambda terms. The idea of reducing inside a pattern is a very interesting but radical departure from programming language practice and it leads to many technical difficulties. Unilluminating restrictions must be applied to obtain substantive results. Peyton Jones and Wadler [15] extend the lambda calculus with a pattern-matching facility that generalizes ordinary abstraction. However, patterns are first-order constructor terms and the calculus is just an abstract syntax for the concrete one in the functional languages. Howard [10] uses a pattern notation (without nesting) as syntactic sugar for expressions in various typed lambda calculi, notably for recursive types.

We have recently learned of the very interesting lecture notes of Lafont [16] in which he proposes, among other things, precisely a computational interpretation of the sequent proofs for intuitionistic propositional logic under the name *clausal calculus*. There seem to be many technical differences between our treatment and his, the most evident one being the interpretation of the left disjunction rule, which appears to make the clausal calculus nondeterministic, somewhat like an unordered set of ML-like pattern-matching clauses. But there is no question that he also saw that sequent left rules can be interpreted to build nested patterns as variable generalizations.

The rest of this paper is organized as follows. In Section 2 we present the typed pattern calculus as a pattern and term assignment to Gentzen's sequent proofs. We show that type-checking is decidable and that types are unique and computable, all in linear time.

We make this assignment into a computational interpretation, as well as clarify it, by specifying in Section 3 two evaluators for closed terms, one lazy and the other one eager, both in natural semantics style. We show that these evaluators are deterministic. We also state the basic properties connecting typing and evaluation: type preservation through evaluation and convergence of well-typed terms. Their proof is postponed since they follow from looking at lazy and eager evaluation as particular reduction strategies in a general nondeterministic reduction system defined on open terms.

Section 4 illustrates programming in the pattern calculus, showing that the simply typed lambda calculus can be translated with just a constant factor overhead, showing how to introduce recursive types, and giving equivalents to the ML programs on lists seen above. We conclude with ideas for further work. The appendix recalls the well-known presentation of the simply typed lambda calculus as a computational interpretation of the natural deduction proofs for intuitionistic logic and two evaluators for it, a lazy one and an eager one.

A reduction system is studied in Section 5. We prove subject reduction and strong normalization for well-typed terms, and the Church–Rosser property for all (raw) terms. In order to relate the evaluators in natural semantics style to the general reduction system, we define in Section 6 two evaluators (lazy and eager) in structured operational semantics style, show that they are equivalent to the ones in natural semantics style, and also that they represent particular reduction strategies in the general reduction system.

2. PATTERN AND TERM ASSIGNMENTS FOR GENTZEN'S SEQUENT PROOFS

2.1. Syntax

For clarity, it is convenient to stipulate the following disjoint sets of variables:

- usual variables: $x, y, z, ...$
- communication variables:[1] ξ, ρ, ψ, etc.

Types

$A ::= \iota \mid A \times A \mid A + A \mid A \to A$ (where ι ranges over some set of base types).

Patterns

$$P ::= \_ \mid x \mid \sharp x \mid \langle P, P \rangle \mid (P \mid_{\xi} P) \mid P @ P$$

Communication Terms

$$T ::= \xi \mid \mathbf{L} \mid \mathbf{R} \text{ (where } \mathbf{L} \text{ and } \mathbf{R} \text{ are constants).}$$

Terms

$$\begin{aligned} M ::= {} & x \mid \langle M, M \rangle \mid inl_A(M) \mid inr_A(M) \mid [M \mid_T M] \\ & \mid \lambda P : A . M \mid x \; of \; M \; is \; P : A \; in \; M \\ & \mid (\lambda P : A . M) \; of \; M \; is \; P : A \; in \; M \mid \\ & \mid let \; M \; be \; P : A \; in \; M \end{aligned}$$

Free and bound occurrences of usual and communication variables are defined as usual with the understanding that the terms of the form $\lambda P : A . M$, $L \; of \; N \; is \; P : A \; in \; M$, and $let \; N \; be \; P : A \; in \; M$ define bindings whose scope is M for all the variables occurring in P. We denote by $Var(P)$ the set of usual and communication variables that occurs in the pattern P and by $FV(M)$ the set of free variables that occurs in the term M. They can be defined by induction as follows:

$$\begin{aligned}
&Var(\_) && && = \varnothing \\
&Var(x) && = Var(\sharp x) && = \{x\} \\
&Var(\langle P, Q \rangle) && = Var(P @ Q) && = Var(P) \cup Var(Q) \\
&Var((P \mid_{\xi} Q)) && && = Var(P) \cup Var(Q) \cup \{\xi\} \\
&FV(\mathbf{L}) && = FV(\mathbf{R}) && = \varnothing \\
&FV(\xi) && && = \{\xi\} \\
&FV(x) && && = \{x\} \\
&FV(inl_A(M)) && = FV(inr_A(M)) && = FV(M) \\
&FV(\langle M, N \rangle) && && = FV(M) \cup FV(N) \\
&FV(\lambda P . M) && && = FV(M) - Var(P) \\
&FV(let \; N \; be \; P : A \; in \; M) && && = FV(N) \cup (FV(M) - Var(P)) \\
&FV([M \mid_T N]) && && = FV(T) \cup FV(M) \cup FV(N) \\
&FV(L \; of \; N \; is \; Q : A \; in \; M) && && = FV(L) \cup FV(N) \cup (FV(M) - Var(Q))
\end{aligned}$$

We write $[N_1, ..., N_n / x_1, ..., x_n]$ (often abbreviated $[\bar{N}/\bar{x}]$) for the typed substitution mapping each variable $x_i : A_i$ to a term $N_i : A_i$ and $M[\bar{N}/\bar{x}]$ for the term M where each variable x_i free in M is replaced by N_i. what follows, for every substitution θ, we assume We identify terms that differ only in the name of their bound variables.

[1] The role of the communication variables will be apparent in the operational semantics.

TABLE 1

Sequent Proof Rules and Corresponding Typing Rules

	$A_1, ..., A_n \vdash A_i$	$x_1:A_1, ...x_n:A_n \rhd x_i:A_i$ where the x_j's are distinct	$(proj)$
$(\wedge right)$	$\dfrac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B}$	$\dfrac{\Gamma \rhd M:A \quad \Gamma \rhd N:B}{\Gamma \rhd \langle M, N\rangle : A \times B}$	$(\times right)$
$(\wedge left)$	$\dfrac{A, B, \Gamma \vdash C}{A \wedge B, \Gamma \vdash C}$	$\dfrac{P:A, Q:B, \Gamma \rhd M:C}{\langle P, Q\rangle : A \times B, \Gamma \rhd M:C}$	$(\times left)$
$(\vee right 1)$	$\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$	$\dfrac{\Gamma \rhd M:A}{\Gamma \rhd inl_B(M):A+B}$	$(+right 1)$
$(\vee right 2)$	$\dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$	$\dfrac{\Gamma \rhd N:B}{\Gamma \rhd inr_A(N):A+B}$	$(+right 2)$
$(\vee left)$	$\dfrac{A, \Gamma \vdash C \quad B, \Delta \vdash C}{A \vee B, \Gamma, \Delta \vdash C}$	$\dfrac{P:A, \Gamma \rhd M:C \quad Q:B, \Gamma \rhd N:C}{(P \mid_{\xi} Q):A+B, \Gamma \rhd [M \mid_{\xi} N]:C}$ where $\begin{cases} \xi \text{ is a fresh communication variable} \\ Var(P) \cap Var(Q) = \varnothing \end{cases}$	$(+left)$
$(\supset right)$	$\dfrac{A, \Gamma \vdash B}{\Gamma \vdash A \supset B}$	$\dfrac{P:A, \Gamma \rhd M:B}{\Gamma \rhd \lambda P:A.M:A \rightarrow B}$	$(\rightarrow right)$
$(\supset left)$	$\dfrac{\Gamma \vdash A \quad B, \Delta \vdash C}{A \supset B, \Gamma, \Delta \vdash C}$	$\dfrac{\Gamma \rhd N:A \quad Q:B, \Gamma \rhd M:C}{\sharp a:A \rightarrow B, \Gamma \rhd z\ of\ N\ is\ Q:B\ in\ M:C}$ where z is a fresh variable	$(\rightarrow left)$
(cut)	$\dfrac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}$	$\dfrac{\Gamma \rhd M:A \quad P:A, \Gamma \rhd N:B}{\Gamma \rhd let\ M\ be\ P:A\ in\ N:B}$	(let)
$(contraction)$	$\dfrac{A, A, \Gamma \vdash B}{A, \Gamma \vdash B}$	$\dfrac{P:A, Q:A, \Gamma \rhd M:B}{P@Q:A, \Gamma \rhd M:B}$	$(layered)$
$(weakening)$	$\dfrac{\Gamma \vdash B}{A, \Gamma \vdash B}$	$\dfrac{\Gamma \rhd M:B}{\_:A, \Gamma \rhd M:B}$	$(wildcard)$

2.2. Typing Rules

Typing judgments have the form $\Gamma \rhd M:A$ where Γ, called a *pattern type assignment*, is a multiset[2] of elements of the form $P:A$. The syntax introduced in Section 2.1 will be referred to as *raw*, to emphasize the fact that it may or may not type-check. For example, raw patterns are not necessarily linear but the well-typed ones are (a pattern or a pattern type assignment is said to be *linear*[3] if variables can occur at most once in it). We write $Var(\Gamma)$ to denote the set $\bigcup_{P:A \in \Gamma} Var(P)$.

The typing rules are in Table 1 and Table 2. Each rule of Table 1 is shown next to the Gentzen sequent proof rule it interprets. The sequent proof rules are as in [8], where the sequents have the form $\Gamma \vdash A$ with A a proposition and Γ a multiset of propositions (slight abuse of notation: we use the same meta-variables for propositions as for types, and for sequent antecedents as for pattern type assignments). The reader will notice the inclusion of the left contraction and left weakening rules, which will not add new propositions to those provable in the system (with cut):[4] we include them because weakening and contraction have simple computational interpretations that correspond to devices long used in programming languages. The formulation of the *proj* rule prevents a judgment like $(x \mid_{\xi} y)$: $a+b, z:C \rhd z:C$ but judgments like $x:A, \_:B \rhd x:A$ are perfectly possible by using first an axiom $x:A \rhd x:A$, then the *wildcard* rule. The form of functional patterns, variables prefixed by the

[2] For judgments which are derivable without the weakening rule, Γ is in fact a set; the pattern of type **1** added later causes the same problem as weakening.

[3] As in the *left-linear* rewrite rule.

[4] Jean Gallier has pointed out to us that the cut rule cannot be eliminated from this system without the contraction rule being present. It is not clear what the significance of this fact is for the computational interpretation that we are considering, where the interpretation of the cut rule is the essential computational engine.

TABLE 2

Extended Typing Rules

$$\frac{\Gamma \rhd M : C}{\Gamma \rhd [M \mid_{\mathbf{L}} N] : C} \quad (\mathbf{L})$$

$$\frac{\Gamma \rhd N : C}{\Gamma \rhd [M \mid_{\mathbf{R}} N] : C} \quad (\mathbf{R})$$

$$\frac{\Gamma \rhd \lambda P : A . L : A \to B \quad \sharp z : A \to B, \Delta \rhd z \textit{ of } N \textit{ is } Q : B \textit{ in } M : C}{\Gamma \rhd (\lambda P : A . L) \textit{ of } N \textit{ is } Q : B \textit{ in } M : C} \quad (\textit{app})$$

symbol $\sharp$, may seem surprising at first: why not just variables? This prefix will allow us to treat the evaluation of pattern-matching against function type patterns analoguously to the way we treat product and sum types, see Section 3.

The rules appearing in the second table are used to type some terms obtained via substitution as intermediate expressions in the evaluation of terms appearing in the first table: $[M \mid_{\mathbf{L}} N]$, $[M \mid_{\mathbf{R}} N]$ are obtained by substituting ξ by either **L** or **R** in the term $[M \mid_{\xi} N]$; and $(\lambda P : AJ)$ *of* N *is* $Q : B$ *in* M is obtained by substituting z by $\lambda P : AJ$ in the term z *of* N *is* $Q : B$ *in* M. Note that the term N in the two first expressions is not necessarily well-typed, i.e., it is a raw term.

Finally, we note that here, as opposed to the simply typed lambda calculus, a typing judgment may have several derivations.

2.3. Decidability of Type-Checking

One can immediately check that if $\Gamma \rhd M : A$ is derivable then Γ is linear. We show now that even if a judgment may have several derivations, type-checking is decidable and types are unique.

We first define the *deconstruction* of a raw pattern type assignment Γ, noted $Decon(\Gamma)$, by induction on the structure of patterns:

$$\begin{aligned}
Decon(\_ : A, \Gamma) &= Decon(\Gamma)\\
Decon(x : A, \Gamma) &= x : A, Decon(\Gamma)\\
Decon(\sharp x : A, \Gamma) &= \sharp x : A, Decon(\Gamma)\\
Decon(\langle P, Q \rangle : A \times B, \Gamma) &= Decon(P : A), Decon(Q : B),\\
&\quad\ Decon(\Gamma)\\
Decon(P @ Q : A, \Gamma) &= Decon(P : A), Decon(Q : A),\\
&\quad\ Decon(\Gamma)\\
Decon((P \mid_{\xi} Q) : A, \Gamma) &= (P \mid_{\xi} Q) : A, Decon(\Gamma)
\end{aligned}$$

The function $Decon(\Gamma)$ eliminates all the wildcard patterns, and replace *repetitively*, *while possible*, $\langle P, Q \rangle : A \times B$ with $P : A, Q : B$ and $P @ Q : A$ with $P : A, Q : A$. Clearly, $Decon(\Gamma)$ is well-defined, computable in linear time and has only patterns of the forms x, $(P \mid_{\xi} Q)$ and $\sharp z$. On the other hand, by Proposition 2.2, we can state the relation between two derivations $\Gamma \rhd M : C$ and $Decon(\Gamma) \rhd M : C$ in the following way:

LEMMA 2.1. *For any type-checking derivation $\mathscr{D}$ that ends with $\Gamma \rhd M : C$ there is type-checking derivation of height at most that of $\mathscr{D}$, which ends with $Decon(\Gamma) \rhd M : C$.*

Proof. By induction on the height of the derivation $\Gamma \rhd M : C$, then by cases according to the last rule used in the derivation, using the Proposition 2.2 below.

End of Proof.

PROPOSITION 2.2 (Commutation). *Withing type derivations, the rules* $(\times \textit{left})$, $(\textit{layered})$ *and* $(\textit{wildcard})$ *commute with all the other rules.*

Proof. The proof is by case-analysis and is quite straightforward, we only show three cases to illustrate how it works.

- $(\times \textit{left})$ commutes with $(\times \textit{right})$

$$\frac{\dfrac{P : A, Q : B, \Gamma \rhd M : C \qquad P : A, Q : B, \Gamma \rhd N : D}{P : A, Q : B, \Gamma \rhd \langle M, N \rangle : C \times D}\ (\times \textit{right})}{\langle P, Q \rangle : A \times B, \Gamma \rhd \langle M, N \rangle : C \times D}\ (\times \textit{left})$$

$$\frac{\dfrac{P : A, Q : B, \Gamma \rhd M : C}{\langle P, Q \rangle : A \times B, \Gamma \rhd M : C}\ (\times \textit{left}) \qquad \dfrac{P : A, Q : B, \Gamma \rhd N : D}{\langle P, Q \rangle : A \times B, \Gamma \rhd N : D}\ (\times \textit{left})}{\langle P, Q \rangle : A \times B, \Gamma \rhd \langle M, N \rangle : C \times D}\ (\times \textit{right})$$

- (*layered*) commutes with (+*left*)

$$\dfrac{\dfrac{P{:}A,\ Q{:}A,\ R{:}C,\ \Gamma \rhd M{:}E \qquad P{:}A,\ Q{:}A,\ S{:}D,\ \Gamma \rhd N{:}E}{P{:}A,\ Q{:}A,\ (R \mid_{\xi} S){:}\ C+D,\ \Gamma \rhd [M \mid_{\xi} N]{:}E}\ (+\mathit{left})}{P @ Q{:}A,\ (R \mid_{\xi} S){:}C+D,\ \Gamma \rhd [M \mid_{\xi} N]{:}E}\ (\mathit{layered})$$

$$\dfrac{\dfrac{P{:}A,\ Q{:}A,\ R{:}C,\ \Gamma \rhd M{:}E}{P @ Q{:}A,\ R{:}C,\ \Gamma \rhd M{:}E}\ (\mathit{layered}) \qquad \dfrac{P{:}A,\ Q{:}A,\ S{:}D,\ \Gamma \rhd N{:}E}{P @ Q{:}A,\ S{:}D,\ \Gamma \rhd N{:}E}\ (\mathit{layered})}{P @ Q{:}A,\ (R \mid_{\xi} S){:}C+D,\ \Gamma \rhd [M \mid_{\xi} N]{:}E}\ (+\mathit{left})$$

- (*wildcard*) commutes with (→*left*)

$$\dfrac{\dfrac{\Gamma \rhd N{:}C \qquad R{:}D,\ \Gamma \rhd M{:}E}{\sharp z{:}C \to D,\ \Gamma \rhd z\ \mathit{of}\ N\ \mathit{is}\ R{:}D\ \mathit{in}\ M{:}E}\ (\to\mathit{left})}{\_{:}A,\ \sharp z{:}C \to D,\ \Gamma \rhd z\ \mathit{of}\ N\ \mathit{is}\ R{:}D\ \mathit{in}\ M{:}E}\ (\mathit{wildcard})$$

$$\dfrac{\dfrac{\Gamma \rhd N{:}C}{\_{:}A,\ \Gamma \rhd N{:}C}\ (\mathit{wildcard}) \qquad \dfrac{R{:}D,\ \Gamma \rhd M{:}E}{\_{:}A,\ R{:}D,\ \Gamma \rhd M{:}E}\ (\mathit{wildcard})}{\_{:}A,\ \sharp z{:}C \to D,\ \Gamma \rhd z\ \mathit{of}\ N\ \mathit{is}\ R{:}D\ \mathit{in}\ M{:}E}\ (\to\mathit{left})$$

End of Proof.

LEMMA 2.3. *For any type-checking derivation $\mathscr{D}$ that ends with $Decon(\Gamma) \rhd M{:}C$ there is a type-checking derivation which ends with $\Gamma \rhd M{:}C$.*

Proof. By application of the rules (×*left*), (*layered*), and (*wildcard*).

End of Proof.

It is not clear, *a priori*, that terms have unique types. Hence, for any raw pattern type assignment Γ and any raw term M we define a finite *set* of types $Types(\Gamma, M)$, recursively, as follows:

$$\begin{aligned}
Types(\Gamma, x) &\overset{\text{def}}{=} \{A \mid x{:}A \in Decon(\Gamma) \text{ and } \nexists (P \mid_{\xi} Q)\ \nexists B \text{ such that } (P \mid_{\xi} Q){:}B \in Decon(\Gamma)\\
&\qquad \text{and } \nexists \sharp z\ \nexists C \text{ such that } \sharp z{:}C \in Decon(\Gamma)\}\\
Types(\Gamma, inl_B(M)) &\overset{\text{def}}{=} \{A+B \mid A \in Types(\Gamma, M)\}\\
Types(\Gamma, inr_A(N)) &\overset{\text{def}}{=} \{A+B \mid B \in Types(\Gamma, N)\}\\
Types(\Gamma, \lambda P{:}A.M) &\overset{\text{def}}{=} \{A \to B \mid B \in Types((P{:}A, \Gamma), M)\}\\
Types(\Gamma, \langle M, N\rangle) &\overset{\text{def}}{=} \{A \times B \mid A \in Types(\Gamma, M), B \in Types(\Gamma, N)\}\\
Types(\Gamma, \mathit{let}\ M\ \mathit{be}\ P{:}A\ \mathit{in}\ N) &\overset{\text{def}}{=} \{B \mid A \in Types(\Gamma, M)\ \mathit{and}\ B \in Types((P{:}A, \Gamma), N)\}\\
Types(\Gamma, [M \mid_{\xi} N]) &\overset{\text{def}}{=} \{C \mid \exists P, Q, A, B, \Gamma' \text{ such that } Decon(\Gamma) = (P \mid_{\xi} Q){:}A+B, \Gamma'\\
&\qquad \text{and } C \in Types((P{:}A, \Gamma'), M) \cap Types((Q{:}B, \Gamma'), N)\}\\
Types(\Gamma, [M \mid_{\mathbf{L}} N]) &\overset{\text{def}}{=} Types(\Gamma, M)\\
Types(\Gamma, [M \mid_{\mathbf{R}} N]) &\overset{\text{def}}{=} Types(\Gamma, N)\\
Types(\Gamma, z\ \mathit{of}\ N\ \mathit{is}\ Q{:}B\ \mathit{in}\ M) &\overset{\text{def}}{=} \{C \mid \exists A, \Gamma'\ Decon(\Gamma) = \sharp z{:}A \to B, \Gamma'\\
&\qquad \text{and } A \in Types(\Gamma', N) \text{ and } C \in Types((Q{:}B, \Gamma'), M)\}\\
Types(\Gamma, (\lambda P{:}A.L)\ \mathit{of}\ N\ \mathit{is}\ Q{:}B\ \mathit{in}\ M) &\overset{\text{def}}{=} \{C \mid \exists B, z, \Gamma'\ Decon(\Gamma) = \Gamma' \text{ and } A \to B \in Types(\Gamma', \lambda P{:}A.L)\\
&\qquad \text{and } C \in Types((\sharp z{:}A \to B, \Gamma'), z\ \mathit{of}\ N\ \mathit{is}\ Q{:}B\ \mathit{in}\ M)\}
\end{aligned}$$

It is easy to show by induction on raw terms that if Γ is linear then $Types(\Gamma, M)$ has at most one element and is computable in linear time. Correctness follows from:

LEMMA 2.4. *$\Gamma \rhd M : A$ is derivable if and only if $A \in Types(\Gamma, M)$.*

Proof. The *if* direction is easily shown by induction on M and Lemma 2.3. The *only if* direction is shown by induction on the height of the derivation of $\Gamma \rhd M : A$, using Lemma 2.1.

End of Proof.

To keep the type-checking algorithm in linear time for an arbitrary raw input, we halt and answer "no" whenever *Types*( ) gets more than one element.

COROLLARY 2.5 (Type-Checking). *Given Γ and M, it is decidable in linear time whether there exists an A such that $\Gamma \rhd M : A$ is derivable. If it exists, A is unique and computable, also in linear time.*

3. EVALUATORS IN NATURAL SEMANTICS STYLE

The next step is to present the operational semantics for the typed pattern calculus by means of evaluators. We follow the method currently known as Natural Semantics [13]. Evaluators in natural semantics style are proof systems for deriving assertions of the form $M \Downarrow K$ where M, K are closed terms and which have the informal meaning that K is the *final* result of the evaluation of M or that M evaluates to the result K. Typically, K has a special shape, which we shall call a *canonical form*. We present a *lazy* and an *eager* evaluator: the first one requires as little evaluation as possible in each phase of the computation, in the spirit of call-by-name, and it does not evaluate under constructors, while the second one requires as much evaluation as possible in all phases, corresponding to call-by-value strategies (except that, of course, it does not evaluate under λ-abstractions).

In description of the operational semantics the type decorations are omitted from the syntax to avoid cluttering the notation. No formal type-erasure operation takes place.

3.1. A Lazy Evaluator

The salient feature of this lazy evaluator is that evaluation and matching "call" each other. In order to derive assertions of the form $M \Downarrow_1 K$, our rules use auxiliary assertions of the form *match M on $P \Downarrow_1 \sigma$*, where M is a closed term, P is a pattern, and σ is a closed substitution. *Substitutions* are understood to be finite partial functions mapping the usual variables to terms and the communication variables to either one of two special forms $\mathbf{L}$, $\mathbf{R}$. The operation of substitution itself, for which we use the meta-notation $M[\sigma]$, is defined as usual. The rules are in Table 3.

The meaning of the terms $[M \mid_{\mathbf{L}} N]$ and $[N \mid_{\mathbf{R}} M]$ only depends on the subterm M, this is the reason we define them to be well typed even if N is not. In other words, these terms are *intermediate* expressions used by the evaluators, they are never considered as results. Anyway, if a term $[M \mid_{\mathbf{L}} N]$ or $[N \mid_{\mathbf{R}} M]$ is obtained by substituting a well-typed term (which is always the case), then the subterm N is also a well-typed subterm.

Note also that if function type patterns were simply variables, then no evaluation would be required before substitution, not even to a lambda abstraction form. This

TABLE 3

Lazy Evaluator in Natural Semantics Style

$$\frac{\mathit{match}\ M\ \mathit{on}\ P \Downarrow_1 \sigma \qquad N[\sigma] \Downarrow_1 K}{\mathit{let}\ M\ \mathit{be}\ P\ \mathit{in}\ N \Downarrow_1 K} \qquad \frac{}{\mathit{match}\ L\ \mathit{on}\ x \Downarrow_1 [L/x]}$$

$$\frac{L \Downarrow_1 \langle M, N \rangle \quad \mathit{match}\ M\ \mathit{on}\ P \Downarrow_1 \sigma \quad \mathit{match}\ N\ \mathit{on}\ Q \Downarrow_1 \theta \quad \mathit{Dom}(\sigma) \cap \mathit{Dom}(\theta) = \varnothing}{\mathit{match}\ L\ \mathit{on}\ \langle P, Q \rangle \Downarrow_1 \sigma, \theta}$$

$$\frac{L \Downarrow_1 \mathit{inl}(M) \quad \mathit{match}\ M\ \mathit{on}\ P \Downarrow_1 \sigma \quad \xi \notin \mathit{Dom}(\sigma)}{\mathit{match}\ L\ \mathit{on}\ (P \mid_{\xi} Q) \Downarrow_1 [\mathbf{L}/\xi], \sigma} \qquad \frac{L \Downarrow_1 \mathit{inr}(N) \quad \mathit{match}\ N\ \mathit{on}\ Q \Downarrow_1 \sigma \quad \xi \notin \mathit{Dom}(\sigma)}{\mathit{match}\ L\ \mathit{on}\ (P \mid_{\xi} Q) \Downarrow_1 [\mathbf{R}/\xi], \sigma}$$

$$\frac{M \Downarrow_1 K}{[M \mid_{\mathbf{L}} N] \Downarrow_1 K} \qquad \frac{N \Downarrow_1 K}{[M \mid_{\mathbf{R}} N] \Downarrow_1 K}$$

$$\frac{L \Downarrow_1 \lambda P.J}{\mathit{match}\ L\ \mathit{on}\ \sharp z \Downarrow_1 [\lambda P.J/z]} \qquad \frac{\mathit{match}\ N\ \mathit{on}\ P \Downarrow_1 \sigma \quad \mathit{let}\ J[\sigma]\ \mathit{be}\ Q\ \mathit{in}\ MM \Downarrow_1 K}{(\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M \Downarrow_1 K}$$

$$\frac{\mathit{match}\ L\ \mathit{on}\ P \Downarrow_1 \sigma \quad \mathit{match}\ L\ \mathit{on}\ Q \Downarrow_1 \theta \quad \mathit{Dom}(\sigma) \cap \mathit{Dom}(\theta) = \varnothing}{\mathit{match}\ L\ \mathit{on}\ P@Q \Downarrow_1 \sigma, \theta} \qquad \frac{}{\mathit{match}\ L\ \mathit{on}\ \_ \Downarrow_1 \varnothing}$$

$$\frac{}{\langle M, N \rangle \Downarrow_1 \langle M, N \rangle} \qquad \frac{}{\lambda P.M \Downarrow_1 \lambda P.M} \qquad \frac{}{\mathit{inl}(M) \Downarrow_1 \mathit{inl}(M)} \qquad \frac{}{\mathit{inr}(N) \Downarrow_1 \mathit{inr}(N)}$$

would differ essentially from the treatment of product and sum patterns.

We define a *lazy canonical form* to be a closed term K given by the the grammar:

$$K ::= \langle M, N\rangle \mid \mathit{inl}(M) \mid \mathit{inr}(M) \mid \lambda P{:}A.M$$

where P, M range over patterns, respectively terms.

The lazy evaluator is deterministic and enjoys the following elementary property:

PROPOSITION 3.1. *If $M \Downarrow_1 K$ then K is a lazy-canonical form, and if K is a lazy-canonical form then $K \Downarrow_1 K$.*

Proof. The proof is by a straightforward induction on the height of the derivation of $M \Downarrow_1 K$.

End of Proof.

THEOREM 3.2. *The lazy evaluator is deterministic.*

Proof. We have to show that all the lazy evaluations of a term M yield the same result. The same for a substitution σ. We proceed by induction on the height of a lazy evaluation.

- If $M \Downarrow_1 M$, then either $M \equiv \mathit{inl}(N)$, or $M \equiv \mathit{inr}(N)$, or $M \equiv \lambda P{:}A.N$ or $M \equiv \langle M_1, M_2\rangle$ and so the property trivially holds.
- $M \equiv [M_1 \mid_{\mathbf{L}} M_2] \Downarrow_1 K_1$ where $M_1 \Downarrow_1 K_1$. Suppose $[M_1 \mid_{\mathbf{L}} M_2] \Downarrow_1 K_2$ where $M_1 \Downarrow_1 K_2$. Then, we have $K_1 \equiv K_2$ by induction hypothesis. The case $M \equiv [M_2 \mid_{\mathbf{R}} M_1]$ is symmetrical.
- $M \equiv (\lambda P{:}A.J)$ *of* N *is* $Q{:}B$ *in* $L \Downarrow_1 K_1$ where *match* N *on* $P \Downarrow_1 \sigma_1$ and *let* $J[\sigma_1]$ *be* Q *in* $L \Downarrow_1 K_1$. Then if $M \Downarrow_1 K_2$, where *match* N *on* $P \Downarrow_1 \sigma_2$ and *let* $J[\sigma_2]$ *be* Q *in* $L \Downarrow_1 K_2$, we have $\sigma_1 \equiv \sigma_2$ by i.h. and thus $J[\sigma_1] \equiv J[\sigma_2]$. Once again by i.h. $K_1 \equiv K_2$ which concludes the proof of this case.
- $M \equiv$ *let* R *be* $P{:}A$ *in* $N \Downarrow_1 K_1$, where *match* R *on* $P \Downarrow_1 \sigma_1$ and $N[\sigma_1] \Downarrow_1 K_1$. Then if $M \Downarrow_1 K_2$ where *match* R *on* $P \Downarrow_1 \sigma_2$ and $N[\sigma_2] \Downarrow_1 K_2$, then we have $\sigma_1 \equiv \sigma_2$ by i.h. and thus $N[\sigma_2] \equiv N[\sigma_2]$. Once again by i.h. we obtain $K_1 \equiv K_2$.
- If *match* M *on* $\_ \Downarrow_1 \varnothing$ or *match* M *on* $x \Downarrow_1 [M/x]$, the property trivially holds.
- *match* M *on* $\sharp z \Downarrow_1 [\lambda P_1.J_1/z] \equiv \sigma_1$, where $M \Downarrow_1 \lambda P_1.J_1$. By i.h. the result of evaluation for M is unique and thus also that of match M on $\sharp z$.
- *match* M *on* $\langle P_1, P_2\rangle \Downarrow_1 \sigma_1$, where $M \Downarrow_1 \langle M_1, M_2\rangle$, *match* M_i *on* $P_i \Downarrow_1 \theta_i$ (for $i = 1, 2$) and $\sigma_1 = \theta_1, \theta_2$. Suppose *match* M *on* $\langle P_1, P_2\rangle \Downarrow_1 \sigma_2$, where $M \Downarrow_1 \langle M'_1, M'_2\rangle$, *match* M'_i *on* $P_i \Downarrow_1 \theta'_i$ (for $i = 1, 2$) and $\sigma_2 = \theta'_1, \theta'_2$. We obtain by i.h. $\langle M_1, M_2\rangle \equiv \langle M'_1, M'_2\rangle$ and $\theta_i \equiv \theta'_i$ (for $i = 1, 2$) and thus $\sigma_1 \equiv \sigma_2$.
- *match* M *on* $(P_1 \mid_\xi P_2) \Downarrow_1 \sigma$, where $M \Downarrow_1 \mathit{inl}(N)$ and *match* N *on* $P_1 \Downarrow_1 \rho$ and $\sigma = [\mathbf{L}/\xi], \rho$. By i.h. the evaluation of M and *match* N *on* P_1 is deterministic and thus that of *match* M *on* $(P_1 \mid_\xi P_2)$ *also. The case* $M \Downarrow_1 \mathit{inr}(N)$ is symmetrical.
- *match* M *on* $P_1 @ P_2 \Downarrow_1 \sigma$, where *match* M *on* $P_1 \Downarrow_1 \theta_1$, *match* M *on* $P_2 \Downarrow_1 \theta_2$ and $\sigma_1 = \theta_1, \theta_2$. By i.h. the results of *match* M *on* P_1 and *match* M *on* P_2 are unique and so that of *matchM on* $P_1 @ P_2$ also.

End of Proof.

3.2. An Eager Evaluator

In this evaluator, evaluation and pattern matching are independent as evaluation is not needed in order to match terms against patterns. The rules are in Table 4. Note that in accordance with programming language practice, even in an eager evaluator there is a bit of laziness, that is, we do not evaluate under lambda abstraction.

We say that the pattern matching *fails* if none of the cases described above applies.

We define an *eager canonical form* to be a closed term K given by the grammar

$$K ::= \langle K, K\rangle \mid \mathit{inl}(K) \mid \mathit{inr}(K) \mid \lambda P{:}A.M$$

where P, M range over patterns, respectively terms.

TABLE 4

Eager Evaluator in Natural Semantics Style

$$\frac{\mathit{Match}(M_i, P_i) = \sigma_i \text{ (for } i = 1, 2\text{) and } \mathit{Dom}(\sigma_1) \cap \mathit{Dom}(\sigma_2) = \varnothing}{\mathit{Match}(\langle M_1, M_2\rangle, \langle P_1, P_2\rangle) = \sigma_1, \sigma_2}$$

$$\frac{\mathit{Match}(M, P_i) = \sigma_i \text{ (for } i = 1, 2\text{) and } \mathit{Dom}(\sigma_1) \cap \mathit{Dom}(\sigma_2) = \varnothing}{\mathit{Match}(M, P_1 @ P_2) = \sigma_1, \sigma_2}$$

$$\frac{\mathit{Match}(M, P) = \sigma \text{ and } \xi \notin \mathit{Dom}(\sigma)}{\mathit{Match}(\mathit{inl}(M), (P \mid_\xi Q)) = [\mathbf{L}/x], \sigma} \qquad \frac{\mathit{Match}(M, Q) = \sigma \text{ and } \xi \notin \mathit{Dom}(\sigma)}{\mathit{Match}(\mathit{inr}(M), (P \mid_\xi Q)) = [\mathbf{R}/\xi], \sigma}$$

$$\frac{}{\mathit{Match}(\lambda P.J, \sharp z) = [\lambda P.J/z]} \qquad \frac{}{\mathit{Match}(M, \_) = \varnothing} \qquad \frac{}{\mathit{Match}(M, x) = [M/x]}$$

The eager evaluator is deterministic and enjoys the following elementary property:

PROPOSITION 3.3. *If $M \Downarrow_e K$ then K is an eager-canonical form, and if K is an eager-canonical form then $K \Downarrow_e K$.*

Proof. The proof is by induction on the height of the derivation of $M \Downarrow_e K$ and is straightforward.

End of Proof.

LEMMA 3.4. *The result of $Match(M, P)$ is unique.*

Proof. By induction on the structure of P.

- $P = \_$. Then $Match(M, \_) = \varnothing$.
- $P = x$. Then $Match(M, x) = [M/x]$.
- $P = \sharp z$. Then $M = \lambda P.J$ and $Match(\lambda P.J, \sharp z) = [\lambda P.J/z]$.
- $P = \langle P_1, P_2 \rangle$. Then $M = \langle M_1, M_2 \rangle$ and by i.h. the result of $Match(M_i, P_i)$ (for $i = 1, 2$) is unique. By definition $Match(M, P) = Match(M_1, P_1), Match(M_2, P_2)$ and thus it is also unique.
- $P = (P_1 \mid_\xi P_2)$. By hypothesis either $M = inl(N)$ or $M = inr(N)$. If $M = inl(N)$, then by i.h. the result of $Match(N, P_1)$ is unique (say ρ) and thus $Match(M, P) = [\mathbf{L}/\xi], \rho$ is unique. The case $M = inr(N)$ is symmetrical.
- $P = P_1 @ P_2$. By i.h. the results of $Match(M, P_1)$ and $Match(M, P_2)$ are unique and so $Match(M, P) = Match(M, P_1), Match(M, P_2)$ is also unique.

End of Proof.

THEOREM 3.5. *The eager evaluator is deterministic.*

Proof. We have to show that $M \Downarrow_e K_1$ and $M \Downarrow_e K_2$ implies $K_1 \equiv K_2$ which is done by induction on the height of the first derivation $M \Downarrow_e K_1$ and Lemma 3.4.

End of Proof.

3.3. Basic Properties Connecting Typing and Computation

Our computational interpretation is defined by its syntax, its typing rules and its evaluation rules. Two results can be offered as evidence that these hold together well.

THEOREM 3.6 (Type Preservation). *Let $\Downarrow$ be either $\Downarrow_1$ or $\Downarrow_e$. If $\rhd M : A$ and $M \Downarrow K$ then $\rhd K : A$.*

THEOREM 3.7 (Convergence). *Let $\Downarrow$ be either $\Downarrow_1$ or $\Downarrow_e$. If $\rhd M : A$ then $M \Downarrow K$ for some K.*

Recalling also the decidability of type-checking, we conclude that the pattern calculus enjoys the same basic properties as the simply typed lambda calculus, the Girard–Reynolds polymorphic lambda calculus, etc.

Decidability of type-checking together with Theorems 3.6 and 3.7 shows that the evaluation of a closed well-typed term will not get "stuck" in a term that is not an acceptable result (no "run-time type error"). Indeed, in either evaluator, if M type-checks (that is decidable), then by Convergence it will evaluate to some term K which by Type Preservation and Propositions 3.3 and 3.1, is a well-typed canonical form, which is an acceptable result for computation.[5] In other words, these results ensure that neither eager nor lazy matching can fail during the evaluation of a closed well-typed term.

Rather than giving separate proofs to these theorems for each of the two evaluators, we prefer to see them as corollaries of subject reduction and strong normalization results for a general nondeterministic reduction system (Section 5). The lazy and eager evaluators given above can then be shown to describe particular deterministic reduction strategies.

4. PROGRAMMING EXAMPLES

4.1. Programming as in the Simply Typed Lambda Calculus

The simply typed lambda calculus can be immediately translated into the pattern calculus. The introduction rules/constructs are already here, they have a trivial translation. Following the usual translation of natural deduction proofs into sequent proofs (see, e.g., [9]), the elimination rules/constructs are translated by the corresponding left rules followed by a (let) (which interprets the cut rule), as follows:

$$(\times elim1) \quad \frac{\Delta \rhd M : A \times B}{\Delta \rhd \pi_1(M) : A} \mapsto$$

$$\frac{\Delta \rhd M : A \times B \qquad \dfrac{x : A,\ y : B,\ \Delta \rhd x : A}{\langle x, y \rangle : A \times B,\ \Delta \rhd x : A}\ (\times left)}{\Delta \rhd let\ M\ be\ \langle x, y \rangle : A \times B\ in\ x : A}$$

[5] This argument will not hold if we extend the calculus with some form of divergence such as recursion. But a variation of the type safety property should still hold: the evaluation of a closed well-typed term either terminates with an acceptable result, or it diverges. One can prove such a result quite easily for evaluators in the style of Plotkin's structured operational semantics.

where x, y are fresh. Similarly for $(\times elim2)$.

$$(+elim)\quad \frac{\varDelta \rhd L : A+B \qquad x : A, \varDelta \rhd M : C \qquad y : B, \varDelta \rhd N : C}{\varDelta \rhd case\ L\ of\ x : A.M \mid y : B.N : C} \mapsto$$

$$\frac{\varDelta \rhd L : A+B \qquad \dfrac{x : A, \varDelta \rhd M : C \qquad y : B, \varDelta \rhd N : C}{(x \mid_{\xi} y) : A+B, \varDelta \rhd [M \mid_{\xi} N] : C}\ (+left)}{\varDelta \rhd let\ L\ be\ (x \mid_{\xi} y) : A+B\ in\ [M \mid_{\xi} N] : C}$$

where ξ is fresh.

$$(\to elim)\quad \frac{\varDelta \rhd M : A \to B \qquad \varDelta \rhd N : A}{\varDelta \rhd MN : B} \mapsto$$

$$\frac{\varDelta \rhd M : A \to B \qquad \dfrac{\varDelta \rhd N : A \qquad y : B, \varDelta \rhd y : B}{\sharp z : A \to B, \varDelta \rhd z\ of\ N\ is\ y : B\ in\ y : B}\ (\to left)}{\varDelta \rhd let\ M\ be\ \sharp z : A \to B\ in\ (z\ of\ N\ is\ y : B\ in\ y) : B}$$

where y, z are fresh.

At the level of terms, denoting by M^* the pattern calculus translation of the simply typed term M, we obtain:

$$(\pi_i(M))^*$$

$$\overset{\text{def}}{=} let\ M^*\ be\ \langle x_1, x_2 \rangle : A \times B\ in\ x_i$$

$$(case\ L\ of\ x : A.M \mid y : B.N)^*$$

$$\overset{\text{def}}{=} let\ L^*\ be\ (x \mid_{\xi} y) : A+B\ in\ [M^* \mid_{\xi} N^*]$$

$$(MN)^*$$

$$\overset{\text{def}}{=} let\ M^*\ be\ \sharp z : A \to B\ in\ (z\ of\ N^*\ is\ w : B\ in\ w)$$

where x_1, x_2, ξ, w, z are fresh. It is easy to see that the translation is type preserving (since it mirrors a translation of proofs!) and that $M^*[L^*/x] = (M[L/x])^*$. Moreover, assuming the usual lazy and eager natural semantics evaluators for the simply typed lambda calculus (see the appendix for reference) we have:

Proposition 4.1. *If $M \Downarrow N$ in the simply typed lambda calculus, then $M^* \Downarrow N^*$ in the pattern calculus, where $\Downarrow$ is either $\Downarrow_l$ or $\Downarrow_e$. Moreover, the height of the derivation of $M^* \Downarrow N^*$ depends linearly on the height of the derivation of $M \Downarrow N$ hence computation complexity is preserved by the translation with just a constant overhead.*

Proof. The proof is by induction on $M \Downarrow N$ and is straightforward.

End of Proof.

In fact, the abbreviation $MN =^{\text{def}} let\ M\ be\ \sharp z : A \to B\ in\ (z\ of\ N\ is\ y : B\ in\ y)$ introduced in Section 5.4 satisfies

$$\frac{M \Downarrow_l \lambda P.L \qquad match\ N\ on\ P \Downarrow_l \sigma \qquad L[\sigma] \Downarrow_l K}{MN \Downarrow_l K}$$

and

$$\frac{M \Downarrow_e \lambda P.J \qquad N \Downarrow_e K \qquad J[\sigma] \Downarrow_e L}{MN \Downarrow_e L}$$

where $Match(K, P) = \sigma$.

4.2. Programming with Lists

We now wish to add a type of lists in order to express programs such as the examples in Section 1. We expect that nil and cons are constructor terms of this type but what is a pattern of type list? Rather than offering an ad hoc guess, we derive this in a more general setting since lists are an instance of recursive types. Hence we add, in the spirit of the formalism developed so far, recursive types and also recursion and a type "with one element." This exercise can be seen as a test of the robustness of the pattern calculus paradigm.

Add a type constant **1** and type variables with X ranging over them. Add to types, patterns and terms

$$A ::= \cdots \mid \mathbf{1} \mid X \mid recX.A$$

$$P ::= \cdots \mid \bigstar \mid fold(P)$$

$$M ::= \cdots \mid \bigstar \mid fold_{X.A}(M) \mid \mu x : A.M$$

Add the type checking rules in Table 5. The decidability of type-checking and uniqueness of types property easily extends to this calculus. Then add the rules in Table 6 to the eager evaluator and the rules in Table 7 to the lazy evaluator of Section 3:

Add also to the set of eager-canonical forms

$$K ::= \bigstar \mid fold_{X.A}(K)$$

TABLE 5

Additional Type-Checking Rules

$$\frac{\Gamma \rhd M\colon A}{\star\colon \mathbf{1}, \Gamma \rhd M\colon A} \qquad \Gamma \rhd \star\colon \mathbf{1} \qquad \frac{x\colon A, \Gamma \rhd M\colon A}{\Gamma \rhd \mu x\colon A . M\colon A}$$

$$\frac{P\colon A[\mathit{rec}X.A], \Gamma \rhd M\colon B}{\mathit{fold}(P)\colon \mathit{rec}X.A, \Gamma \rhd M\colon B} \qquad \frac{\Gamma \rhd M\colon A[\mathit{rec}X.A/X]}{\Gamma \rhd \mathit{fold}_{X.A}(M)\colon \mathit{rec}X.A}$$

and to the set of lazy-canonical forms

$$K ::= \star \mid \mathit{fold}_{X.A}(M)$$

Propositions 3.3 and 3.1 are easily extended.

Now, we introduce represent lists as follows. We define the type

$$\mathit{listA} \overset{\text{def}}{=} \mathit{rec}X.\mathbf{1} + A \times X$$

TABLE 6

Additional Eager Evaluation Rules

$$\frac{M[\mu x.M/x] \Downarrow_{\mathrm{e}} K}{\mu x.M \Downarrow_{\mathrm{e}} K} \qquad \frac{}{\star \Downarrow_{\mathrm{e}} \star} \qquad \frac{M \Downarrow_{\mathrm{e}} K}{\mathit{fold}_{X.A}(M) \Downarrow_{\mathrm{e}} \mathit{fold}_{X.A}(K)}$$

$$\mathit{Match}(\star, \star) = \varnothing$$

$$\mathit{Match}(\mathit{fold}_{X.A}(M), \mathit{fold}(P)) = \sigma \qquad \text{where} \quad \mathit{Match}(M, P) = \sigma$$

Terms and patterns of type *listA* are defined as follows:

$$\text{Terms} \quad \begin{cases} \mathit{nil} \overset{\text{def}}{=} \mathit{fold}_{X.\mathbf{1} + A \times X}(\mathit{inl}_{A \times \mathit{listA}}(\star)) \\ \mathit{cons}(M, L) \overset{\text{def}}{=} \mathit{fold}_{X.\mathbf{1} + A \times X}(\mathit{inr}_{\mathbf{1}}(\langle M, L \rangle)) \end{cases}$$

$$\text{Pattern} \quad \{ (\mathit{nil} \mid_{\xi} \mathit{cons}(P, W)) \overset{\text{def}}{=} \mathit{fold}((\star \mid_{\xi} \langle P, W \rangle))$$

We should have tagged *nil* and *cons* with *A* but that seems to clutter the notation a lot. So we use this elliptic notation and we ask the reader to reconstruct the type tags, especially in *flatten* and *suffixlist* below.

As expected,

$$\frac{}{\Gamma \rhd \mathit{nil}\colon \mathit{listA}}$$

$$\frac{\Gamma \rhd M\colon A \qquad \Gamma \rhd L\colon \mathit{listA}}{\Gamma \rhd \mathit{cons}(M, L)\colon \mathit{listA}}$$

$$\frac{\Gamma \rhd M\colon B \qquad P\colon A, W\colon \mathit{listA}, \Gamma \rhd N\colon B}{(\mathit{nil} \mid_{\xi} \mathit{cons}(P, W))\colon \mathit{listA}, \Gamma \rhd [M \mid_{\xi} N]\colon B}$$

are immediately derivable from the type-checking rules, and

$$\frac{}{\mathit{nil} \Downarrow_{\mathrm{e}} \mathit{nil}} \qquad \frac{M \Downarrow_{\mathrm{e}} K \qquad N \Downarrow_{\mathrm{e}} L}{\mathit{cons}(M, N) \Downarrow_{\mathrm{e}} \mathit{cons}(K, L)}$$

$$\mathit{Match}(\mathit{nil}, (\mathit{nil} \mid_{\xi} \mathit{cons}(P, W))) = [\mathbf{L}/\xi]$$

$$\mathit{Match}(\mathit{cons}(M, K), (\mathit{nil} \mid_{\xi} \mathit{cons}(P, W))) = [\mathbf{R}/\xi], \sigma, \theta \qquad \text{where} \quad \begin{cases} \mathit{Match}(M, P) = \sigma \\ \mathit{Match}(K, W) = \theta \\ \mathit{Dom}(\sigma) \cap \mathit{Dom}(\theta) = \varnothing \end{cases}$$

$$\frac{}{\mathit{nil} \Downarrow_{\mathrm{l}} \mathit{nil}} \qquad \frac{}{\mathit{cons}(M, N) \Downarrow_{\mathrm{l}} \mathit{cons}(M, N)}$$

$$\frac{L \Downarrow_{\mathrm{l}} \mathit{nil}}{\mathit{match}\ L\ \mathit{on}\ (\mathit{nil} \mid_{\xi} \mathit{cons}(P, W)) \Downarrow_{\mathrm{l}} [\mathbf{L}/\xi]}$$

$$\frac{L \Downarrow_{\mathrm{l}} \mathit{cons}(M, K) \qquad \mathit{match}\ M\ \mathit{on}\ P \Downarrow_{\mathrm{l}} \sigma \qquad \mathit{match}\ K\ \mathit{on}\ W \Downarrow_{\mathrm{l}} \theta \qquad \mathit{Dom}(\sigma) \cap \mathit{Dom}(\theta) = \varnothing}{\mathit{match}\ L\ \mathit{on}\ (\mathit{nil} \mid_{\xi} \mathit{cons}(P, W)) \Downarrow_{\mathrm{l}} [\mathbf{R}/\xi], \sigma, \theta}$$

are immediately derivable from the evaluation rules, an argument in favor of the robustness of this paradigm.

With this syntactic sugar, we proceed to express the pattern-matching programs listed in the introduction.

For a little more clarity, we write $F =^{\text{def}} M[F/f]$ instead of $F =^{\text{def}} \mu f.M$ and also we omit the type tags on lambda abstraction. The closed terms in Table 8, corresponding to the ML programs shown in Section 1, type-check and have the expected operational behavior in the eager pattern calculus. In the lazy pattern calculus, they correspond to equivalent programs in a language such as Miranda or Haskell.

TABLE 7

Additional Lazy Evaluation Rules

$$\frac{M[\mu x.M/x] \Downarrow_1 K}{\mu x.M \Downarrow_1 K} \qquad \frac{}{\star \Downarrow_1 \star} \qquad \frac{}{fold_{X.A}(M) \Downarrow_1 fold_{X.A}(M)}$$

$$\frac{L \Downarrow_{\mathrm{e}} \star}{match\ L\ on\ \star \Downarrow_{\mathrm{e}} \emptyset} \qquad \frac{L \Downarrow_{\mathrm{e}} fold_{X.A}(M) \qquad match\ M\ on\ P \Downarrow_{\mathrm{e}} \sigma}{match\ L\ on\ fold(P) \Downarrow_{\mathrm{e}} \sigma}$$

TABLE 8

Examples of Programs in the Typed Pattern Calculus

$$suffixlist \overset{\text{def}}{=} \lambda(z\,@((nil \mid_\xi cons(\_, l)))).$$

$$[cons(nil, nil) \mid_\xi cons(z, suffixlist\ l)]$$

$$: listA \to listlistA$$

$$flatten \overset{\text{def}}{=} \lambda((nil \mid_\xi cons(nil \mid_\zeta cons(x, l)), L))).$$

$$[nil \mid_\xi [flatten\ L \mid_\xi cons(x, flatten(cons(l, L)))]]$$

$$: listlistA \to listA$$

$$merge \overset{\text{def}}{=} \lambda\langle z_1\,@((nil \mid_\xi cons(x_1, l_1))), z_2\,@((nil \mid_\xi cons(x_2, l_2)))\rangle.$$

$$[[nil \mid_\zeta z_2] \mid_\xi [z_1 \mid_\zeta cons(x_1, cons(x_2, merge\langle l_1, l_2\rangle))]]$$

$$: listA \times listA \to listA$$

5. A GENERAL REDUCTION SYSTEM AND ITS PROPERTIES

We now consider a rewrite system on all (raw) terms (see Table 9). The reduction relation $\Rightarrow$[6] is defined to be the closure under *all* contexts of the following six reduction rules, where the function $Match(M, P)$ is the same that appears in Section 3.2. The reflexive transitive closure of $\Rightarrow$ is denoted $\Rightarrow^*$.

The specific two last rules for *let* M *e* $P_1 @ P_2 : A$ *in* N and $(\lambda P_1 @ P_1 : A . J)$ *of* M *is* $Q : B$ *in* N produce the independence of the term M with respect to the patterns P_1 and P_2; i.e., M can be computed in two different and independent ways in order to be matched latter with P_1 and P_2. This rule is essential in order to simulate the lazy and eager evaluators of Sections 3.1 and 3.2.

We first show that this notion of reduction is compatible with matching and substitution. Then, we proceed to prove the basic properties of this system, namely *subject reduction*, *confluence*, and *strong-normalization*.

5.1. Reduction, Matching, and Substitution

The relation $\Rightarrow$ is stable by substitution as shown by Lemmas 5.3 and 5.4 using properties 5.1 and 5.2 as well.

Remark 5.1. Let L, M and N be terms and P and Q be patterns such that $Var(P) \cap Var(Q) = \emptyset$, $Var(P) \cap FV(N) = \emptyset$, $Match(N, Q)$ is defined and $Match(M, P)$ is defined. Then

$$L[Match(N, Q)][Match(M[Match(N, Q)], P)]$$
$$= L[Match(M, P)][Match(N, Q)]$$

PROPOSITION 5.2. *If* $Match(M, P) = \theta$ *and* $M \Rightarrow M'$, *then* $Match(M', P) = \theta'$ *and* $\theta(x) \Rightarrow^* \theta'(x)$, $\forall x \in Dom(\theta)$.

Proof. We show the property by induction on the structure of P.

- $P = \_$. Then $Match(M, \_) = \emptyset$ and $Match(M', \_) = \emptyset$ and the property trivially holds.

- $P = x$. Then $Match(M, x) = [M/x]$, $Match(M', x) = [M'/x]$ and $M \Rightarrow M'$ holds by hypothesis.

- $P = \langle P_1, P_2\rangle$. We have $M = \langle M_1, M_2\rangle$, and $Match(\langle M_1, M_2\rangle, \langle P_1, P_2\rangle) = \sigma_1$, σ_2, where $Match(M_1, P_1) = \sigma_1$, $Match(M_2, P_2) = \sigma_2$ and $Dom(\sigma_1) \cap Dom(\sigma_2) = \emptyset$. By definition $\langle M_1, M_2\rangle \Rightarrow M'$ implies either that $M' = \langle M'_1, M_2\rangle$, where $M_1 \Rightarrow M'_1$, or that $M' = \langle M_1, M'_2\rangle$, where $M_2 \Rightarrow M'_2$. W.l.g. suppose the first case holds. By i.h. $Match(M'_i, P_i) = \sigma'_i$, where $\sigma_i(x) \Rightarrow^* \sigma'_i(x)$ for all $x \in Dom(\sigma_i)$, and so we are done.

- $P = (P_1 {}_{|}^{\xi} P_2)$. By hypothesis either $M = inl(N)$ or $M = inr(N)$. W.l.g. suppose the first case holds. Then $\theta = [\mathbf{L}/\xi]$, σ where $Match(N, P_1) = \sigma$. By definition of $\Rightarrow$, $M' = inl(N')$, where $N \Rightarrow N'$ and by i.h. $Match(N', P_1) = \sigma'$, where $\sigma(x) \Rightarrow^* \sigma'(x)$, $\forall x \in Dom(sig)$. As $\mathbf{L} \Rightarrow^* \mathbf{L}$ and $Match(inl(N'), (P_1 \mid_\xi P_2)) = [\mathbf{L}/\xi]$, σ' we are done.

- $P = \sharp z$. Then $M = \lambda P.J$ and $M' = \lambda P.J'$, where $J \Rightarrow J'$. We have $Match(\lambda P.J', \sharp z) = [\lambda P.J'/z]$ and the property holds since $\lambda P.J \Rightarrow \lambda P.J'$.

- $P = P_1 @ P_2$. Then $Match(M, P_1 @ P_2) = \sigma_1, \sigma_2$, where $Match(M, P_1) = \sigma_1$, $Match(M, P_2) = \sigma_2$, and $Dom(\sigma_1) \cap Dom(\sigma_2) = \emptyset$. By i.h. $Match(M', P_i) = \sigma'_i$ and $\sigma_i(x) \Rightarrow^* \sigma'_i(x)$, $\forall x \in Dom(\sigma_i)$ and so we are done.

End of Proof.

LEMMA 5.3. *If* $N \Rightarrow N'$ *and* $Match(N, P)$ *is defined, then* $M[Match(N, P)]$ *reduces to* $M[Match(N', P)]$. *Moreover, if* $Var(P) \cap FV(M) \neq \emptyset$, *then* $M[Match(N, P)] \Rightarrow^+ M[Match(N', P)]$.

Proof. Let $Match(N, P) = \theta$. By Proposition 5.2 $\exists \theta'$ such that $Match(N', P) = \theta'$. We show the property by induction on the structure of M.

- $x \notin Dom(\theta)$. Since $Dom(\theta) = Dom(\theta')$ and $x[\theta] = x \Rightarrow^* x = x[\theta']$ the property trivially holds.

[6] We use here the symbol $\Rightarrow$ instead of $\to$ to differentiate from the arrows in types.

TABLE 9

A General Reduction System

$$\frac{\mathit{Match}(M, P) = \sigma}{\mathit{let}\ M\ \mathit{be}\ P{:}A\ \mathit{in}\ N \Rightarrow N[\sigma]}\ (\mathit{let}) \qquad \frac{\mathit{Match}(M, P) = \sigma}{(\lambda P{:}A.J)\ \mathit{of}\ M\ \mathit{is}\ Q{:}B\ \mathit{in}\ N \Rightarrow \mathit{let}\ J[\sigma]\ \mathit{be}\ Q{:}B\ \mathit{in}\ N}\ (\mathit{of})$$

$$\frac{}{[M \mid_{\mathbf{L}} N] \Rightarrow M}\ (\mathit{left}) \qquad \frac{}{[M \mid_{\mathbf{R}} N] \Rightarrow N}\ (\mathit{right})$$

$$\frac{}{\mathit{let}\ M\ \mathit{be}\ P_1 @ P_2{:}A\ \mathit{in}\ N \Rightarrow \mathit{let}\ \langle M, M\rangle\ \mathit{be}\ \langle P_1, P_2\rangle{:}A \times A\ \mathit{in}\ N}\ (\mathit{cont}-\mathit{let})$$

$$\frac{}{(\lambda P_1 @ P_2{:}A.J)\ \mathit{of}\ M\ \mathit{is}\ Q{:}B\ \mathit{in}\ N \Rightarrow (\lambda\langle P_1, P_2\rangle{:}A \times A.J)\ \mathit{of}\ \langle M, M\rangle\ \mathit{is}\ Q{:}B\ \mathit{in}\ N}\ (\mathit{cont}-\mathit{of})$$

- $x \in Dom(\theta)$. Then $x[\theta] = \theta(x) \Rightarrow^* theta'(x) = x[\theta']$ by Lemma 5.2. If $\emptyset \neq Var(P) \subseteq \{x\}$, then $\theta(x) = N$, $\theta'(x) = N'$ and thus $x[\theta] = N \Rightarrow N' = x[\theta']$.

- In all the other cases the result follows from the induction hypothesis and the fact that $\Rightarrow$ is closed for all contexts.

End of Proof.

LEMMA 5.4. *Let $M \Rightarrow M'$ and $Match(N, P)$ is defined. Then $M[Match(N, P)] \Rightarrow^+ M'[Match(N, P)]$.*

Proof. Let $Match(N, P) = \theta$. We show the property by induction on the structure of M. In all the cases where $M \Rightarrow M'$ is an internal reduction, there is a context $C[\ ]$ such that $M \equiv C[L] \Rightarrow C[L'] \equiv M'$ and $L \Rightarrow L'$. By induction hypothesis $L[\theta] \Rightarrow^+ L'[\theta]$ and the result follows from the fact that $\Rightarrow$ is closed for all contexts. We detail now the case where $M \Rightarrow M'$ is an external reduction.

- $M = let\ M_1\ be\ Q{:}B\ in\ M_2$. We have $(let\ M_1\ be\ Q{:}B\ in\ M_2)[\theta] = let\ M_1[\theta]\ be\ Q{:}B\ in\ M_2[\theta]$ *and* $M' = M_2[Match(M_1, Q)]$. By α-conversion we can rename the bound variables of M in such a way that $Var(P) \cap Var(Q) = \emptyset$ and $Var(Q) \cap FV(N) = \emptyset$. Therefore we have

$$\begin{aligned}
&(let\ M_1\ be\ Q{:}B\ in\ M_2)[Match(N, P)]\\
&\quad = let\ M_1[Match(N, P)]\ be\ Q{:}B\ in\ M_2[Match(N, P)]\\
&\quad \Rightarrow M_2[Match(N, P)][Match(M_1[Match(N, P)], Q)]\\
&\quad =_{\text{Lemma 5.1}} M_2[Match(M_1, Q)][Match(N, P)]
\end{aligned}$$

- $M = T\ of\ M_1\ is\ Q{:}B\ in\ M_2$. We have $(T\ of\ M_1\ is\ Q{:}B\ in\ M_2)[\theta] = T[\theta]\ of\ M_1[\theta]\ is\ Q{:}B\ in\ M_2[\theta]$ and $M' = let\ J[Match(M_1, R)]\ be\ Q{:}B\ in\ M_2$, *where* $T = \lambda R{:}A.J$. By α-conversion we can rename the bound variables of M in such a way that $Var(P) \cap Var(R) = \emptyset$ and $Var(R) \cap FV(N) = \emptyset$. Therefore we have

$$\begin{aligned}
&(\lambda R{:}A.J)\ of\ M_1\ is\ Q{:}B\ in\ M_2[Match(N, P)]\\
&\quad = (\lambda R{:}A.J[Match(N, P)])\ of\ M_1[Match(N, P)]\\
&\qquad is\ Q{:}B\ in\ M_2[Match(N, P)]\\
&\quad \Rightarrow let\ J[Match(N, P)][Match(M_1[Match(N, P)],\\
&\qquad R)]\ be\ Q{:}B\ in\ M_2[Match(N, P)]\\
&\quad =_{\text{Lemma 5.1}} let\ J[Match(M_1, R)][Match(N, P)]\\
&\qquad be\ Q{:}B\ in\ M_2[Match(N, P)]\\
&\quad = (let\ J[Match(M_1, R)]\ be\ Q{:}B\ in\ M_2)[Match(N, P)]
\end{aligned}$$

- $M = [M_1 \mid_T M_2]$, where $T \in \{\xi, \mathbf{L}, \mathbf{R}\}$. We have $[M_1 \mid_T M_2][\theta] = [M_1[\theta] \mid_{T[\theta]} M_2[\theta]]$ and $T[\theta] = T[\theta']$. There are two cases to consider:

  1. $M' = M_1$. Then $T = \mathbf{L}$, $T[\theta] = \mathbf{L}$ and $[M_1[\theta] \mid_{\mathbf{L}} M_2[\theta]] \Rightarrow^+ M_1[\theta]$.
  2. $M' = M_2$. Then $T = \mathbf{R}$, $T[\theta] = \mathbf{R}$ and $[M_1[\theta] \mid_{\mathbf{R}} M_2[\theta]] \Rightarrow^+ M_2[\theta]$.

End of Proof.

From Lemmas 5.3 and 5.4 we obtain:

LEMMA 5.5. *If $M \Rightarrow M'$ and $N \Rightarrow N'$, then $M[Match(N, P)] \Rightarrow^* M'[Match(N', P)]$.*

5.2. Subject Reduction

We first establish that type-checking rules are compatible with matching-substitution then we state and prove the subject reduction property.

LEMMA 5.6. *If $P{:}B, \Gamma \rhd M{:}A$ and $Var(P) \cap FV(M) = \emptyset$, then $\Gamma \rhd M{:}A$.*

Proof. By induction on the height of the derivation $P{:}B, \Gamma \rhd M{:}A$.

End of Proof.

Since in what follows we will need to reason by induction on the structure of the derivation of a judgment and since here, as we have remarked previously, a judgment may have several derivations, it will be useful to associate "canonical" derivations to judgments in such a way that the *last* rule applied in a canonical derivation of $\Gamma \rhd M{:}E$ depends on the structure of the term M itself. We list below the rule associated to each nonvariable term:

Term	Canonical Rule
$\langle M, N\rangle$	$(\times right)$
$inl_B(M)$	$(+right1)$
$inr_B(M)$	$(+right2)$
$[M \mid_\xi N]$	$(+left)$
$[M \mid_{\mathbf{L}} N]$	$(\mathbf{L})$
$[M \mid_{\mathbf{R}} N]$	$(\mathbf{R})$
$\lambda P{:}B.M$	$(\rightarrow right)$
$z\ of\ N\ is\ Q{:}B\ in\ M$	$(\rightarrow left)$
$(\lambda P{:}A.L)\ of\ N\ is\ Q{:}B\ in\ M$	(app)
$let\ M\ be\ P{:}A\ in\ N$	(let)

It is an immediate consequence of Proposition 2.2 that this is possible.

Lemma 5.7 (Substitution Lemma). *Let $P{:}B, \Gamma \rhd M{:}A$, $\Gamma \rhd N{:}B$ and $Match(N, P) = \theta$. Then $\Gamma \rhd M[\theta]{:}A$.*

Proof. The proof is by induction on the height of the derivation $P{:}B, \Gamma \rhd M{:}A$ using the previous lemma.

- $P{:}B, \Gamma \rhd M{:}A$ is an axiom. Then $M = y$, $P = x$ and $\{x\} = Dom(\theta)$. There are two possibilities:

— if $y \neq x$, then $y[\theta] = y$ and $\Gamma \rhd y{:}A$ holds by Lemma 5.6.

— if $y = x$, then $y[\theta] = x[\theta] = N$, and $\Gamma \rhd N{:}B$ holds by hypothesis.

- $P{:}B, \Gamma \rhd M{:}A$ is not an axiom. The possible cases are

—

$$\frac{P_1{:}S_1, P_2{:}S_2, \Gamma \rhd M{:}A}{\langle P_1, P_2\rangle{:}S_1 \times S_2, \Gamma \rhd M{:}A} \quad (\times left)$$

By definition $N = \langle N_1, N_2\rangle$ and $B = S_1 \times S_2$ so that $\theta = Match(\langle N_1, N_2\rangle, \langle P_1, P_2\rangle) = \theta_1, \theta_2$ where $Match(N_i, P_i) = \theta_i$, for $i = 1, 2$. As $\Gamma \rhd N_i{:}S_i$ $(i = 1, 2)$ by the remarks above on canonical derivations, $P_2{:}S_2, \Gamma \rhd M[\theta_1]{:}A$ and $\Gamma \rhd M[\theta_1][\theta_2]{:}A$ hold by i.h. By the assumption on substitutions and the fact that $Dom(\theta_1) \cap Dom(\theta_2) = \varnothing$ then $M[\theta_1][\theta_2] = M[\theta]$, so $\Gamma \rhd M[\theta]{:}A$ holds.

—

$$\frac{P_1{:}B, P_2{:}B, \Gamma \rhd M{:}A}{P_1 @ P_2{:}B, \Gamma \rhd M{:}A} \quad (layered)$$

By definition $\theta = \theta_1, \theta_2$ where $Match(N, P_i) = \theta_i$, for $i = 1, 2$. The proof proceeds as in the previous case.

—

$$\frac{\Gamma \rhd M{:}A}{\_{:}B, \Gamma \rhd M{:}A} \quad (wildcard)$$

Then θ is empty and the property trivially holds.

—

$$\frac{P{:}B, \Gamma \rhd M_1{:}B_1 \qquad P{:}B, \Gamma \rhd M_2{:}B_2}{P{:}B, \Gamma \rhd \langle M_1, M_2\rangle{:}B_1 \times B_2} \quad (\times right)$$

By i.h. $\Gamma \rhd M_i[\theta]{:}B_i$ $(i = 1, 2)$ and then $\Gamma \rhd \langle M_1[\theta], M_2[\theta]\rangle{:}B_1 \times B_2$. Since $\langle M_1[\theta], M_2[\theta]\rangle = \langle M_1, M_2\rangle[\theta]$ we are done.

—

$$\frac{P{:}B, \Gamma \rhd L{:}C}{P{:}B, \Gamma \rhd inl_D(L){:}C + D} \quad (+right)$$

By i.h. $\Gamma \rhd L[\theta]{:}C$ and so $\Gamma \rhd inl_D(L[\theta]){:}C + D$. Since $inl_D(L[\theta]) = inl_D(L)[\theta]$ we are done. The case $M \equiv inr_D(L)$ is symmetrical.

—

$$\frac{P{:}B, Q{:}C, \Gamma \rhd L{:}D}{P{:}B, \Gamma \rhd \lambda Q{:}C.L{:}C \rightarrow D} \quad (\rightarrow right)$$

By i.h. $Q{:}C$, $\Gamma \rhd L[\theta]{:}D$ and so $\Gamma \rhd \lambda Q{:}C.L[\theta]{:} C \rightarrow D$. Since $Var(P) \cap Var(Q) = \varnothing$, then $\lambda Q{:}C.L[\theta] = (\lambda Q{:}C.L)[\theta]$ and we are done.

—

$$\frac{P{:}B, \Gamma \rhd K{:}C \qquad P{:}B, Q{:}C, \Gamma \rhd L{:}A}{P{:}B, \Gamma \rhd let\ K\ be\ Q{:}C\ in\ L{:}A} \quad (let)$$

By i.h. $\Gamma \rhd K[\theta]{:}C$ and $Q{:}C, \Gamma \rhd L[\theta]{:}A$ so that $\Gamma \rhd let\ K[\theta]\ be\ Q{:}C\ in\ L[\theta]{:}A$. Since $Var(P) \cap Var(Q) = \varnothing$, then $let\ K[\theta]\ be\ Q{:}C\ in\ L[\theta] = (let\ K\ be\ Q{:}C\ in\ L)[\theta]$ and we are done.

—

$$\frac{\Gamma \rhd L{:}C \qquad Q{:}D, \Gamma \rhd M{:}A}{\sharp z{:}C \rightarrow D, \Gamma \rhd z\ of\ L\ is\ Q{:}D\ in\ M{:}A} \quad (\rightarrow left)$$

∗ If $P = \sharp z$, then $N = \lambda R{:}C.J$ and $(z\ of\ L\ is\ Q{:}D\ in\ M)[\theta] = (\lambda R.J)\ of\ L\ is\ Q{:}D\ in\ M$. Since $\Gamma \rhd \lambda R{:}C.J{:}C \rightarrow D$ holds by hypothesis, then $\Gamma \rhd (\lambda R{:}C.J)\ of\ L\ is\ Q{:}D\ in\ M{:}A$ by the extended notion of typability.

∗ If $P \neq \sharp z$, we have

$$\frac{P{:}B, \Delta \rhd L{:}C \qquad P{:}B, Q{:}D, \Delta \rhd M{:}A}{P{:}B, \sharp z{:}C \rightarrow D, \Delta \rhd z\ of\ L\ is\ Q{:}D\ in\ M{:}A} \quad (\rightarrow left)$$

By i.h. $\Delta \rhd L[\theta]{:}C$ and $Q{:}D$, $\Delta \rhd M[\theta]{:}A$ and by the $(\rightarrow left)$ rule $\sharp z{:}C \rightarrow D, \Delta \rhd z\ of\ L[\theta]\ is\ Q{:}D\ in\ M[\theta]{:}A$.

Since z is a fresh variable and patterns P and Q have no common variables, then $(z\ \mathit{of}\ L\ \mathit{is}\ Q{:}D\ \mathit{in}\ M)[\theta] = z\ \mathit{of}\ L[\theta]\ \mathit{is}\ Q{:}D\ \mathit{in}\ M[\theta]$ and we are done.

—

$$\frac{\begin{array}{c}\Gamma, P{:}B \rhd \lambda R{:}C.J{:}C \to D\\ \sharp z{:}C \to D, \Gamma, P{:}B \rhd z\ \mathit{of}\ M_1\ \mathit{is}\ Q{:}D\ \mathit{in}\ M_2{:}A\end{array}}{\Gamma, P{:}B \rhd (\lambda R{:}C.J)\ \mathit{of}\ M_1\ \mathit{is}\ Q{:}D\ \mathit{in}\ M_2{:}A}\ (\mathit{app})$$

By i.h. $\Gamma \rhd (\lambda R{:}C.J)[\theta]{:}C \to D$ and $\sharp z{:}C \to D, \Gamma \rhd (z\ \mathit{of}\ M_1\ \mathit{is}\ Q{:}D\ \mathit{in}\ M_2)[\theta]{:}A$. Since $\mathit{Var}(\sharp z) \cap \mathit{Var}(P) = \emptyset$, then $(z\ \mathit{of}\ M_1\ \mathit{is}\ Q{:}D\ \mathit{in}\ M_2)[\theta] = z\ \mathit{of}\ M_1[\theta]\ \mathit{is}\ Q{:}D\ \mathit{in}\ M_2[\theta]$ and by the (app) rule $\Gamma \rhd (\lambda R{:}C.J)[\theta]\ \mathit{of}\ M_1[\theta]\ \mathit{is}\ Q{:}D\ \mathit{in}\ M_2[\theta]{:}A$.

—

$$\frac{P_1{:}S_1, \Gamma \rhd M_1{:}A \qquad P_2{:}S_2, \Gamma \rhd M_2{:}A}{(P_1 \mid_\xi P_2){:}S_1 + S_2, \Gamma \rhd [M_1 \mid_\xi M_2]{:}A}\ (+\mathit{left})$$

∗ If $P = (P_1 \mid_\xi P_2)$, then $N = \mathit{inl}(N')$ where $\theta = [\mathbf{L}/\xi]$, ρ and $\mathit{Match}(N', P_1) = \rho$ or $N = \mathit{inr}(N')$ where $\theta = [\mathbf{R}/\xi]$, ρ and $\mathit{Match}(N', P_2) = \rho$. In both cases $\mathit{Var}(P_1) \cap \mathit{Var}(P_2) = \emptyset$.

In the first case $[M_1 \mid_\xi M_2][\theta] = [M_1[\rho] \mid_{\mathbf{L}} M_2[\rho]]$ and by i.h. $\Gamma \rhd M_1[\rho]{:}A$. By the extended rule $(\mathbf{L})$ $\Gamma \rhd [M_1[\rho] \mid_{\mathbf{L}} M_2[\rho]]{:}A$.

The second case is symmetrical.

∗ $P \neq (P_1 \mid_\xi P_2)$. Then

$$\frac{P_1{:}S_1, P{:}B, \Delta \rhd M_1{:}A \qquad P_2{:}S_2, P{:}B, \Delta \rhd M_2{:}A}{(P_1 \mid_\xi P_2){:}S_1 + S_2, P{:}B, \Delta \rhd [M_1 \mid_\xi M_2]{:}A}$$

By i.h. $P_1{:}S_1, \Delta \rhd M_1[\theta]{:}A$ and $P_2{:}S_2, \Delta \rhd M_2[\theta]{:}A$ and then $(P_1 \mid_\xi P_2){:}S_1 + S_2, \Delta \rhd [M_1[\theta] \mid_\xi M_2[\theta]]{:}A$. Since ξ is fresh, then $\theta(\xi) = \xi$, $[M_1[\theta] \mid_\xi M_2[\theta]] = [M_1 \mid_\xi M_2][\theta]$ and $(P_1 \mid_\xi P_2){:}S_1 + S_2, \Delta \rhd [M_1 \mid_\xi M_2][\theta]{:}A$ holds.

—

$$\frac{\Gamma, P{:}B \rhd M_1{:}C}{\Gamma, P{:}B \rhd [M_1 \mid_{\mathbf{L}} M_2]{:}C}\ (\mathbf{L})$$

By i.h. $\Gamma \rhd M_1[\theta]{:}C$ and then we obtain $\Gamma \rhd [M_1[\theta] \mid_{\mathbf{L}} M_2[\theta]]{:}C$ by application of the $(\mathbf{L})$ rule. The case $M \equiv [M_1 \mid_{\mathbf{R}} M_2]$ is symmetrical.

End of Proof.

Theorem 5.8 (Subject Reduction). *If* $\Gamma \rhd M{:}A$ *and* $M \Rightarrow M'$, *then* $\Gamma \rhd M'{:}A$.

Proof. The proof is by induction on height of the derivation of $\Gamma \rhd M{:}A$ and then by analyzing the position of the redex and the Substitution Lemma. We only show the interesting cases, corresponding to the four possible external reductions.

- $[M_1 \mid_{\mathbf{L}} M_2] \to M_1$. By the extended notion of typability $\Gamma \rhd M_1{:}A$ holds.
- $[M_1 \mid_{\mathbf{R}} M_2] \to M_2$. By the extended notion of typability $\Gamma \rhd M_2{:}A$ holds.
- $\mathit{let}\ R\ \mathit{be}\ P{:}B\ \mathit{in}\ N \to N[\theta]$, where $\mathit{Match}(R, P) = \theta$. By the remarks above there exists a canonical derivation ending in

$$\frac{\Gamma \rhd R{:}B \qquad P{:}B, \Gamma \rhd N{:}A}{\Gamma \rhd \mathit{let}\ R\ \mathit{be}\ P{:}B\ \mathit{in}\ N{:}A}\ (\mathit{let})$$

and by Lemma 5.7 we have $\Gamma \rhd N[\theta]{:}A$.

- $(\lambda P{:}B.J)\ \mathit{of}\ N\ \mathit{is}\ Q{:}D\ \mathit{in}\ M \to \mathit{let}\ J[\theta]\ \mathit{be}\ Q{:}D\ \mathit{in}\ M$, where $\mathit{Match}(N, P) = \theta$. By the extended notion of typability there are a type D and a variable z such that

$$\frac{\begin{array}{c}\Gamma \rhd \lambda P{:}B.J{:}B \to D\\ \sharp z{:}B \to D, \Gamma \rhd z\ \mathit{of}\ N\ \mathit{is}\ Q{:}D\ \mathit{in}\ M{:}A\end{array}}{(\lambda P{:}B.J)\ \mathit{of}\ N\ \mathit{is}\ Q{:}D\ \mathit{in}\ M}\ (\mathit{app})$$

Now, there are canonical derivations ending in

$$\frac{\Gamma \rhd N{:}B \qquad Q{:}D, \Gamma \rhd M{:}A}{\sharp z{:}B \to D, \Gamma \rhd z\ \mathit{of}\ N\ \mathit{is}\ Q{:}D\ \mathit{in}\ M{:}A}\ (\to \mathit{left})$$

$$\frac{P{:}B, \Gamma \rhd J{:}D}{\Gamma \rhd \lambda P{:}B.J{:}B \to D}\ (\to \mathit{right})$$

By Lemma 5.7 we have $\Gamma \rhd J[\theta]{:}D$ and then by application of the (let) rule

$$\frac{\Gamma \rhd J[\theta]{:}D \qquad Q{:}D, \Gamma \rhd M{:}A}{\Gamma \rhd \mathit{let}\ J[\theta]\ \mathit{be}\ Q{:}D\ \mathit{in}\ M{:}A}\ (\mathit{let})$$

End of Proof.

5.3. Confluence

By an analysis of critical pairs one can show that

Proposition 5.9 (Weak Church–Rosser). *For reduction on all (raw) terms, if* $L \Rightarrow L'$ *and* $L \Rightarrow L''$, *then there exists* L''' *such that* $L' \Rightarrow^* L'''$ *and* $L'' \Rightarrow^* L'''$.

By Newman's Lemma, together with the strong normalization property for well-typed terms (proved in Section 5.4) and with the subject reduction property (Section 5.2), this implies that the (strong) Church–Rosser property holds for reduction on *well-typed terms*.

But in fact we can prove a more general result, namely that the Church–Rosser property holds for reduction on all (raw) terms. We use a method due to Tait and Martin–Löf, relating the reduction relation $\Rightarrow$ to the parallel reduction relation $\blacktriangleright$, defined as follows:

- $M \blacktriangleright M$
- If $M \blacktriangleright M'$, then
  - — $[M \mid_{\mathbf{L}} N] \blacktriangleright M'$
  - — $[N \mid_{\mathbf{R}} M] \blacktriangleright M'$
  - — $\lambda P.M \blacktriangleright \lambda P.M'$
  - — $inl(M) \blacktriangleright inl(M')$
  - — $inr(M) \blacktriangleright inr(M')$
- If $M \blacktriangleright M'$ and $N \blacktriangleright N'$, then
  - — $[M \mid_T N] \blacktriangleright [M' \mid_T N']$
  - — $\langle M, N\rangle \blacktriangleright \langle M', N'\rangle$
  - — $let\ M\ be\ P\ in\ N \blacktriangleright let\ M'\ be\ P\ in\ N'$
  - — If also $M \blacktriangleright M''$, then $let\ M\ be\ P_1 @ P_2\ in\ N \blacktriangleright let\ \langle M', M''\rangle\ be\ \langle P_1, P_2\rangle\ in\ N'$
  - — If $Match(M, P)$ is defined, then $let\ M\ be\ P\ in\ N \blacktriangleright N'[Match(M', P)]$
- If $M \blacktriangleright M'$, $N \blacktriangleright N'$ and $L \blacktriangleright L'$, then
  - — $M\ of\ N\ is\ Q\ in\ L \blacktriangleright M'\ of\ N'\ is\ Q\ in\ L'$
  - — If also $N \blacktriangleright N''$, then $(\lambda P_1 @ P_2.M)\ of\ N\ is\ Q\ in\ L \blacktriangleright (\lambda\langle P_1, P_2\rangle.M)\ of\ \langle N', N''\rangle\ is\ Q\ in\ L$
  - — If $Match(N, P)$ is defined, then $(\lambda P.M)\ of\ N\ is\ Q\ in\ L \blacktriangleright let\ M'[Match(N', P)]\ be\ Q\ in\ L'$.

The equivalence between the relations $\blacktriangleright^*$ and $\Rightarrow^*$ is proved using the fact that $\blacktriangleright$ is stable by substitution, as stated by Lemma 5.11.

Lemma 5.10. *If $Match(M, P) = \theta$ and $M \blacktriangleright M'$, then $Match(M', P) = \theta'$, where $\theta(x) \blacktriangleright \theta'(x)$, $\forall x \in Dom(\theta)$.*

Lemma 5.11. *If $M \blacktriangleright M'$, $N \blacktriangleright N'$ and $Match(N, P)$ is defined, then*

$$M[Match(N, P)] \blacktriangleright M'[Match(N', P)]$$

Proof. By induction on the structure of M (as in Lemma 5.5) using Lemma 5.10.

End of Proof.

Lemma 5.12. *The reflexive transitive closures of $\blacktriangleright$ and $\Rightarrow$ are the same relation.*

Proof. To show the inclusion $\blacktriangleright^* \subseteq \Rightarrow^*$, we show that $\blacktriangleright \subseteq \Rightarrow^*$ by induction on the definition of $\blacktriangleright$.

- If $M \blacktriangleright M$, then $M \Rightarrow^* M$ trivially holds.
- In the case where $\blacktriangleright$ comes from internal reductions, we have that $M \blacktriangleright M'$, $N \blacktriangleright N'$ and $L \blacktriangleright L'$ imply

$$\begin{array}{ll}
\lambda P.M & \blacktriangleright \lambda P.M' \\
M\ of\ N\ is\ Q\ in\ L & \blacktriangleright M'\ of\ N'\ is\ Q\ in\ L' \\
let\ M\ be\ P\ in\ N & \blacktriangleright let\ M'\ be\ P\ in\ N' \\
\langle M, N\rangle & \blacktriangleright \langle M', N'\rangle \\
inl(M) & \blacktriangleright inl(M') \\
inr(M) & \blacktriangleright inr(M') \\
[M \mid_T N] & \blacktriangleright [M' \mid_T N']
\end{array}$$

As $M \Rightarrow^* M'$, $N \Rightarrow^* N'$ and $L \Rightarrow^* L'$ hold by i.h. and $\Rightarrow$ is closed under all contexts, we obtain the result.

- In the cases where $M \blacktriangleright M'$ implies

$$[M \mid_{\mathbf{L}} N] \blacktriangleright M' \qquad \text{or} \qquad [N \mid_{\mathbf{R}} M] \blacktriangleright M'$$

we have $M \Rightarrow^* M'$ by i.h. and therefore $[M \mid_{\mathbf{L}} N] \Rightarrow M \Rightarrow^* M'$ and $[N \mid_{\mathbf{R}} M] \Rightarrow^* M \Rightarrow^* M'$.

- If $let\ M\ be\ P_1 @ P_2\ in\ N \blacktriangleright let\ \langle M', M''\rangle\ be\ \langle P_1, P_2\rangle\ in\ N'$, where $M \blacktriangleright M'$, $M \blacktriangleright M''$ and $N \blacktriangleright N'$, by i.h. $M \Rightarrow^* M'$, $M \Rightarrow^* M''$ and $N \Rightarrow^* N'$ and then $let\ M\ be\ P_1 @ P_2\ in\ N \Rightarrow let\ \langle M, M\rangle\ be\ \langle P_1, P_2\rangle\ in\ N \Rightarrow^* let\ \langle M', M''\rangle\ be\ \langle P_1, P_2\rangle\ in\ N \Rightarrow^* let\ \langle M', M''\rangle\ be\ \langle P_1, P_2\rangle\ in\ N'$.
- If $let\ M\ be\ P\ in\ N \blacktriangleright N'[Match(M', P)]$, where $M \blacktriangleright M'$, $N \blacktriangleright N'$ and $Match(M, P) = \sigma$, by i.h. $M \Rightarrow^*{}'$ and $N \Rightarrow^* N'$ and then $let\ M\ be\ P\ in\ N \Rightarrow N[Match(M, P)] \Rightarrow^*_{\text{Lemma 5.5}} N'[Match(M', P)]$.
- If $(\lambda P_1 @ P_2.M)\ of\ N\ is\ Q\ in\ L \blacktriangleright (\lambda\langle P_1, P_2\rangle.M')\ of\ \langle N', N''\rangle\ is\ Q\ in\ L'$, where $M \blacktriangleright M'$, $N \blacktriangleright N'$, $N \blacktriangleright N''$ and $L \blacktriangleright L'$, by i.h. $M \Rightarrow^* M'$, $N \Rightarrow^* N'$, $N \Rightarrow^* N''$ and $L \Rightarrow^* L'$, so that

$$\begin{aligned}
&(\lambda P_1 @ P_2.M)\ of\ N\ is\ Q\ in\ L \\
&\quad \Rightarrow (\lambda\langle P_1, P_2\rangle.M)\ of\ \langle N, N\rangle\ is\ Q\ in\ L \\
&\quad \Rightarrow^* (\lambda\langle P_1, P_2\rangle.M')\ of\ \langle N, N\rangle\ is\ Q\ in\ L \\
&\quad \Rightarrow^* (\lambda\langle P_1, P_2\rangle.M')\ of\ \langle N', N''\rangle\ is\ Q\ in\ L \\
&\quad \Rightarrow^* (\lambda\langle P_1, P_2\rangle.M')\ of\ \langle N', N''\rangle\ is\ Q\ in\ L'
\end{aligned}$$

- If $(\lambda P.M)$ *of* N *is* Q *in* $L \blacktriangleright$ *let* $M'[Match(N', P)]$ *be* Q *in* L', where $M \blacktriangleright M'$, $N \blacktriangleright N'$, $L \blacktriangleright L'$ and $Match(N, P) = \sigma$. By i.h. $M \Rightarrow^* M'$, $N \Rightarrow^* N'$ and $L \Rightarrow^* L'$ so

$$\begin{aligned}
&(\lambda P.M)\ of\ N\ is\ Q\ in\ L\\
&\qquad \Rightarrow (\lambda P.M)\ of\ N\ is\ Q\ in\ L'\\
&\qquad \Rightarrow let\ M[Match(N, P)]\ be\ Q\ in\ L'\\
&\qquad \Rightarrow^*_{\text{Lemma 5.5}}\ let\ M'[Match(N', P)]\ be\ Q\ in\ L'\\
&\qquad \Rightarrow
\end{aligned}$$

To show the inclusion $\Rightarrow^* \subseteq \blacktriangleright^*$ we show that $\Rightarrow \subseteq \blacktriangleright$ by induction on the definition of $\Rightarrow$.

- In the case where $\Rightarrow$ comes from internal reductions, we have that $M \Rightarrow M'$, $N \Rightarrow N'$, and $L \Rightarrow L'$ imply

$$\begin{aligned}
&M\ of\ N\ is\ Q\ in\ L \Rightarrow M'\ of\ N\ is\ Q\ in\ L\\
&M\ of\ N\ is\ Q\ in\ L \Rightarrow M\ of\ N'\ is\ Q\ in\ L\\
&M\ of\ N\ is\ Q\ in\ L \Rightarrow M\ of\ N\ is\ Q\ in\ L'\\
&let\ M\ be\ P\ in\ N && \Rightarrow let\ M'\ be\ P\ in\ N\\
&let\ M\ be\ P\ in\ N && \Rightarrow let\ M\ be\ P\ in\ N'\\
&\langle M, N\rangle && \Rightarrow \langle M', N\rangle\\
&inl(M) && \Rightarrow inl(M')\\
&inr(M) && \Rightarrow inr(M')\\
&\lambda P.M && \Rightarrow \lambda P.M'\\
&[M \mid_T N] && \Rightarrow [M' \mid_T N]\\
&[M \mid_T N] && \Rightarrow [M \mid_T N']\\
&\langle M, N\rangle && \Rightarrow \langle M, N'\rangle
\end{aligned}$$

As $M \blacktriangleright M'$, $N \blacktriangleright N'$ and $L \blacktriangleright L'$ hold by i.h. and $\blacktriangleright$ is closed under all contexts, we obtain the result.

- If $[M \mid_{\mathbf{L}} N] \Rightarrow M$ or $[N \mid_{\mathbf{R}} M] \Rightarrow M$, the result comes from the fact that $M \blacktriangleright M$.

- If *let* M *be* P *in* $N \Rightarrow N[Match(M, P)]$, then *let* M *be* P *in* $N \blacktriangleright N[Match(M, P)]$ because $M \blacktriangleright M$, $N \blacktriangleright N$, and $Match(M, P)$ is defined by hypothesis.

- If $(\lambda P.M)$ *of* N *is* Q *in* $L \Rightarrow$ *let* $M[Match(N, P)]$ *be* Q *in* L, where the substitution $Match(N, P)$ is defined, then from $M \blacktriangleright M$, $N \blacktriangleright N$ and $L \blacktriangleright L$ we get $(\lambda P.M)$ *of* N *is* Q *in* $L \blacktriangleright$ *let* $N[Match(N, P)]$ *be* Q *in* L.

- If *let* M *be* $P_1 @ P_2$ *in* $N \Rightarrow$ *let* $\langle M, M\rangle$ *be* $\langle P_1, P_2\rangle$ *in* N, by definition $M \blacktriangleright M$ and $N \blacktriangleright N$ so that *let* M *be* $P_1 @ P_2$ *in* $N \blacktriangleright$ *let* $\langle M, M\rangle$ *be* $\langle P_1, P_2\rangle$ *in* N.

- If $(\lambda P_1 @ P_2.M)$ *of* N *is* Q *in* $L \Rightarrow (\lambda\langle P_1, P_2\rangle.M)$ *of* $\langle N, N\rangle$ *is* Q *in* L, by definition $M \blacktriangleright M$, $N \blacktriangleright N$, and $L \blacktriangleright L$ so that $(\lambda P_1 @ P_2.M)$ *of* N *is* Q *in* $L \blacktriangleright (\lambda\langle P_1, P_2\rangle.M)$ *of* $\langle N, N\rangle$ *is* Q *in* L.

End of Proof.

PROPOSITION 5.13. *The relation* $\blacktriangleright$ *satisfies the diamond property: for every term* L *such that* $L \blacktriangleright L_1$ *and* $L \blacktriangleright L_2$, *there exists* L_3 *such that* $L_1 \blacktriangleright L_3$ *and* $L_2 \blacktriangleright L_3$.

Proof. There are two cases:

1. If $L \blacktriangleright L_1$ and $L \blacktriangleright L2$ come from internal reductions, then there exists a context $C[\,]$ such that

$$\begin{tikzcd}
L = C[M^1 \cdots M^n] \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & C[M^1_1 \cdots M^n_1] = L_1\\
L_2 = C[M^1_2 \cdots M^n_2],
\end{tikzcd}$$

where $M^i \blacktriangleright M^i_1$ and $M^i \blacktriangleright M^i_2$ for $i = 1 \cdots n$. By i.h. there are subterms $M^1_3 \cdots M^n_3$ such that $M^i_1 \blacktriangleright M^i_3$ and $M^i_2 \blacktriangleright M^i_3$ for $i = 1 \cdots n$. Let $L_3 = C[M^1_3 \cdots M^n_3]$; then $L_1 \blacktriangleright L_3$ and $L_2 \blacktriangleright L_3$ because $\blacktriangleright$ is closed under all contexts.

2. If $L \blacktriangleright L_1$ or $L \blacktriangleright L_2$ does not come from internal reductions, we show the property by induction on the structure of L by case analysis. In all the cases that follow, the terms L, L_1, and L_2 contain some subterms for which we can apply i.h. so to abbreviate let suppose that $N \blacktriangleright N_1$, $N \blacktriangleright N_2$, $N \blacktriangleright N_3$, $N \blacktriangleright N_4$, $J \blacktriangleright J_1$, $J \blacktriangleright J_2$, $M \blacktriangleright M_1$, and $M \blacktriangleright M_2$. By i.h. there are terms N_5, N_6, N_7, J_3, and M_3 that make it possible to close the diagrams as follows:

$$\begin{tikzcd}
N \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & N_2 \arrow[d, phantom, "\blacktriangledown"] & N \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & N_3 \arrow[d, phantom, "\blacktriangledown"] & N \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & N_3 \arrow[d, phantom, "\blacktriangledown"]\\
N_1 \arrow[r, phantom, "\blacktriangleright"] & N_5 & N_1 \arrow[r, phantom, "\blacktriangleright"] & N_6 & N_4 \arrow[r, phantom, "\blacktriangleright"] & N_7
\end{tikzcd}$$

$$\begin{tikzcd}
M \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & M_2 \arrow[d, phantom, "\blacktriangledown"] & J \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & J_2 \arrow[d, phantom, "\blacktriangledown"]\\
M_1 \arrow[r, phantom, "\blacktriangleright"] & M_3 & J_1 \arrow[r, phantom, "\blacktriangleright"] & J_3
\end{tikzcd}$$

The cases to consider are:

- If $L = [M \mid_T N]$, there are two cases:

$$\begin{tikzcd}
{[M \mid_T N]} \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & {[M_2 \mid_T N_2]} \arrow[d, phantom, "\blacktriangledown"] & {[M \mid_T N]} \arrow[r, phantom, "\blacktriangleright"] \arrow[d, phantom, "\blacktriangledown"] & M_2 \arrow[d, phantom, "\blacktriangledown"]\\
M_1 \arrow[r, phantom, "\blacktriangleright"] & M_3 & M_1 \arrow[r, phantom, "\blacktriangleright"] & M_3
\end{tikzcd}$$

The case $L = [N \mid_T M]$ is symmetrical.

- If $L = \mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ M$, there are five cases:

$$\begin{tikzcd}
\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ M \arrow[r] \arrow[d] & M_2[\mathit{Match}(N_2, P)] \arrow[d, "\text{Lemma 5.11}"] \\
\mathit{let}\ N_1\ \mathit{be}\ P\ \mathit{in}\ M_1 \arrow[r, "\text{Lemma 5.10}"'] & M_3[\mathit{Match}(N_5, P)]
\end{tikzcd}$$

$$\begin{tikzcd}
\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ M \arrow[r] \arrow[d] & M_2[\mathit{Match}(N_2, P)] \arrow[d, "\text{Lemma 5.11}"] \\
M_1[\mathit{Match}(N_1, P)] \arrow[r, "\text{Lemma 5.11}"'] & M_3[\mathit{Match}(N_5, P)]
\end{tikzcd}$$

$$\begin{tikzcd}
\mathit{let}\ N\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M \arrow[r] \arrow[d] & \mathit{let}\ \langle N_2, M_3\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M_2 \arrow[d] \\
\mathit{let}\ N_1\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M_1 \arrow[r] & \mathit{let}\ \langle N_5, N_6\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M_3
\end{tikzcd}$$

$$\begin{tikzcd}
\mathit{let}\ N\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M \arrow[r] \arrow[d] & \mathit{let}\ \langle N_2, N_3\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M_2 \arrow[dd, "\text{Lemma 5.10}"] \\
M_1[\mathit{Match}(N_1, P_1 @ P_2)] \arrow[d, equal] & \\
M_1[\mathit{Match}(\langle N_1, N_1\rangle, \langle P_1, P_2\rangle)] \arrow[r] & M_3[\mathit{Match}(\langle N_5, N_6\rangle, \langle P_1, P_2\rangle)]
\end{tikzcd}$$

$$\begin{tikzcd}
\mathit{let}\ N\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M \arrow[r] \arrow[d] & \mathit{let}\ \langle N_2, N_3\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M_2 \arrow[d] \\
\mathit{let}\ \langle N_1, N_4\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M_1 \arrow[r] & \mathit{let}\ \langle N_5, N_7\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M_3
\end{tikzcd}$$

- If $L = (\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M$, *there are five cases*:

$$\begin{tikzcd}
(\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M \arrow[r] \arrow[d] & \mathit{let}\ J_2[\mathit{Match}(N_2, P)]\ \mathit{be}\ Q\ \mathit{in}\ M_2 \arrow[d, "\text{Lemma 5.11}"] \\
(\lambda P.J_1)\ \mathit{of}\ N_1\ \mathit{is}\ Q\ \mathit{in}\ M_1 \arrow[r, "\text{Lemma 5.10}"'] & \mathit{let}\ J_3[\mathit{Match}(N_5, P)]\ \mathit{be}\ Q\ \mathit{in}\ M_3
\end{tikzcd}$$

$$\begin{tikzcd}
(\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M \arrow[r] \arrow[d] & \mathit{let}\ J_2[\mathit{Match}(N_2, P)]\ \mathit{be}\ Q\ \mathit{in}\ M_2 \arrow[d, "\text{Lemma 5.11}"] \\
\mathit{let}\ J_1[\mathit{Match}(N_1, P)]\ \mathit{be}\ Q\ \mathit{in}\ M_1 \arrow[r, "\text{Lemma 5.11}"'] & \mathit{let}\ J_3[\mathit{Match}(N_5, P)]\ \mathit{be}\ Q\ \mathit{in}\ M_3
\end{tikzcd}$$

$$\begin{tikzcd}
(\lambda P_1 @ P_2.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M \arrow[r] \arrow[d] & (\lambda\langle P_1, P_2\rangle.J)\ \mathit{of}\ \langle N_2, N_3\rangle\ \mathit{is}\ Q\ \mathit{in}\ M_2 \arrow[d] \\
(\lambda P_1 @ P_2.J_1)\ \mathit{of}\ N_1\ \mathit{is}\ Q\ \mathit{in}\ M_1 \arrow[r] & (\lambda\langle P_1, P_2\rangle.J_3)\ \mathit{of}\ \langle N_5, N_6\rangle\ \mathit{is}\ Q\ \mathit{in}\ M_3
\end{tikzcd}$$

$$\begin{tikzcd}
(\lambda P_1 @ P_2.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M \arrow[r] \arrow[d] & (\lambda\langle P_1, P_2\rangle.J_2)\ \mathit{of}\ \langle N_2, N_3\rangle\ \mathit{is}\ Q\ \mathit{in}\ M_2 \arrow[dd, "\text{Lemma 5.10}"] \\
\mathit{let}\ J_1[\mathit{Match}(N_1, P_1 @ P_2)]\ \mathit{be}\ Q\ \mathit{in}\ M_1 \arrow[d, equal] & \\
\mathit{let}\ J_1[\mathit{Match}(\langle N_1, N_1\rangle, \langle P_1, P_2\rangle)]\ \mathit{be}\ Q\ \mathit{in}\ M_2 \arrow[r] & \mathit{let}\ J_3[\mathit{Match}(\langle N_5, N_6\rangle, \langle P_1, P_2\rangle)]\ \mathit{be}\ Q\ \mathit{in}\ M_3
\end{tikzcd}$$

$$\begin{tikzcd}
(\lambda P_1 @ P_2.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M \arrow[r] \arrow[d] & (\lambda\langle P_1, P_2\rangle.J_2)\ \mathit{of}\ \langle N_2, N_3\rangle\ \mathit{is}\ Q\ \mathit{in}\ M_2 \arrow[d] \\
(\lambda\langle P_1, P_2\rangle.J_1)\ \mathit{of}\ \langle N_1, N_4\rangle\ \mathit{is}\ Q\ \mathit{in}\ M_1 \arrow[r] & (\lambda\langle P_1, P_2\rangle.J_3)\ \mathit{of}\ \langle N_5, N_7\rangle\ \mathit{is}\ Q\ \mathit{in}\ M_3
\end{tikzcd}$$

End of Proof.

It is easy to see that if a relation R satisfies the diamond property then its reflexive transitive closure R^* also satisfies it. Putting it all together we obtain:

THEOREM 5.14 (Confluence). $\Rightarrow$ *is confluent* (*Church–Rosser*).

5.4. Strong Normalization

The technique used to prove the strong normalization property is an adaptation of Tait's method, with refinements by Girard and Prawitz.

We first define the notion of stability for well-typed terms, using the notation MN as an abbreviation of the term $\mathit{let}\ M\ \mathit{be}\ \sharp z{:}B \to C\ \mathit{in}\ z\ \mathit{of}\ N\ \mathit{is}\ y{:}C\ \mathit{in}\ y$, where $B \to C$ is the type of M and C the type of N.

DEFINITION 5.15 (Stable Terms). A term M of type A is defined to be stable as following:

- If A is an atomic type, M is stable if and only if it is strongly normalizing.
- If $A \equiv A_1 \times A_2$, M is stable if and only if it is strongly normalizing and whenever M reduces to $\langle M_1, M_2\rangle$, then M_1 and M_2 are both stable.
- If $A \equiv A_1 + A_2$, M is stable if and only if it is strongly normalizing and whenever M reduces to $\mathit{inl}_{A_1 + A_2}(M')$ or to $\mathit{inr}_{A_1 + A_2}(M')$, then M' is stable.
- If $A \equiv A_1 \to A_2$, M is stable if and only if for every stable term N of type A_1, MN is stable.

The goal of this section is to show that every *typed* term is strongly normalizing. For that, as traditionally done, we show that every stable term is strongly normalizing (Lemma 5.19), and that every typed term is stable (Theorem 5.29). The main difference of this proof from existing proofs of strong normalization of λ-calculi in the literature is the extension of the technique to the case of patterns tackled by the notion of *set-stable sets* proposed in Definition 5.23.

PROPOSITION 5.16. *Let M be a term of type $A \rightarrow B$. The term M is stable if and only if $MN_1 \cdots N_k$ is stable for arbitrary stable $N_1 \cdots N_k$ of appropriate types.*

Proof. By induction on k.

End of Proof.

Remark 5.17. We can then use equivalently as a definition for stability $k > 1$ or $k = 1$. In the following we will use the most suitable one for each case and we denote a sequence $N_1 \cdots N_k$ as $\bar{N}$.

The theorem follows from the following sequence of lemmas. In these lemmas "term" means well-typed term, but we have omitted the pattern type assignment to simplify the notation.

LEMMA 5.18. *If N, M_1, M_2, and $\bar{K}$ are strongly normalizing, Q is a pattern, and x is a variable, then the following terms are all strongly normalizing*:

$$x\bar{K} \qquad inl_A(N) \qquad inr_A(N) \qquad \langle M_1, M_2 \rangle$$
$$(x \text{ of } M_1 \text{ is } Q \text{ in } M_2)\ \bar{K}$$

Proof. The argument of the proof is the same for all the cases: it is sufficient to see that a generic reduction sequence starting from the given term always terminates.

End of Proof.

LEMMA 5.19.

1. *Every stable term M is strongly normalizing.*

2. *A well-typed term $xK_1 \cdots K_n$ is stable for arbitrary strongly normalizing terms $K_1 \cdots K_n$.*

Proof. We show the two properties at the same time by induction on the type A of the term.

- If A is not a functional type:

1. By definition.

2. By Lemma 5.18 the term $x\bar{K}$ is strongly normalizing and the reduction sequences starting at $x\bar{K}$ can only proceed in the K_i's. Therefore, $x\bar{K}$ cannot reduce to a pair, nor to an *inl*, nor to an *inr*, and thus $x\bar{K}$ is stable by definition.

- If A is a functional type:

1. Let $A \equiv B \rightarrow C$ and let x be a variable of type B. By the second induction hypothesis (with $n = 0$) $x{:}B$ is stable and by definition Mx is of type C and it is stable. By the first induction hypothesis Mx is strongly normalizing. Suppose now that M is not strongly normalizing. Then there is an infinite reduction sequence $M \Rightarrow M_1 \Rightarrow \cdots$ and thus an infinite reduction sequence $Mx \Rightarrow M_1 x \Rightarrow \cdots$ which leads to a contradiction. Therefore M is strongly normalizing.

2. Let $xK_1 \cdots K_n$ be of type $B \rightarrow C$ with all the K_i's strongly normalizing. Let N be any stable term of type B. From the first induction hypothesis N is also strongly normalizing, by the second induction hypothesis $xK_1 \cdots K_n N$ is stable and by Proposition 5.16 $xK_1 \cdots K_n$ is stable.

End of Proof.

COROLLARY 5.20. *Every variable is stable.*

LEMMA 5.21. *If M is a stable term and $M \Rightarrow N$, then N is stable.*

Proof. Let A be the type of M. We first recall that N is also of type A, by Theorem 5.8. We show the property by induction on A.

- A is not a functional type: M is strongly normalizing and then also N is strongly normalizing since every reduction sequence starting at N can be embedded in a reduction sequence starting at M, which terminates by hypothesis. On the other hand, when $N \Rightarrow^* \langle M_1, M_2 \rangle$, then M_1 and M_2 are stable, because $M \Rightarrow^* N \Rightarrow^* \langle M_1, M_2 \rangle$ and M is stable. Similarly, if $N \Rightarrow^* inl_B(L)$ or $N \Rightarrow^* inr_B(L)$, we have $M \Rightarrow^* N \Rightarrow^* inl_B(L)$ or $M \Rightarrow^* N \Rightarrow^* inr_B(L)$ and then L is stable. Therefore N is stable.

- A is a functional type: by definition of stability on arrow types, it suffices to show that NL is stable for any stable term L. Now, given a stable L, ML is stable because M is, and $ML \Rightarrow NL$, so by the induction hypothesis NL is stable.

End of Proof.

LEMMA 5.22.

- *$inl_A(M)$ and $inr_A(M)$ are stable if M is stable.*
- *$\langle M_1, M_2 \rangle$ is stable if M_1 and M_2 are stable.*
- *$[M_1 \mid_T M_2]$ is stable if M_1 and M_2 are stable and T is a communication term.*
- *z of M_1 is Q in M_2 is stable if M_1 and M_2 are stable.*

Proof. By Lemma 5.19 the terms M, M_1 and M_2 are strongly normalizing. Then use Lemmas 5.18 and 5.21 for the first and second case, and also Lemma 5.16 for the third and fourth.

End of Proof.

DEFINITION 5.23 (Set-Stable Sets). Let

$$[\![M, N, P]\!] = \{M[\mathit{Match}(N', P)] \mid N \Rightarrow^* N' \text{ and }$$
$$\mathit{Match}(N', P) \text{ is defined}\}.$$

We define the set $[\![M, N, P]\!]$ to be set-stable by cases, in the following way:

- When P is not a layered pattern, $[\![M, N, P]\!]$ is set-stable if and only if

1. Every term in $[\![M, N, P]\!]$ is stable, and
2. If $Match(N, P)$ fails or $Var(P) \cap FV(M) = \varnothing$, then the terms N and M are stable.

- When P is a layered pattern $P_1 @ P_2$, $[\![M, N, P_1 @ P_2]\!]$ is set-stable if and only if $[\![M, \langle N, N\rangle, \langle P_1, P_2\rangle]\!]$ is set-stable.

The reason to use a different notion of set-stable sets when P is a layered pattern comes directly from the two last rules of Table 9.

Lemma 5.24. *If* $[\![M, N, P]\!]$ *is set-stable then* M *and* N *are strongly normalizing.*

Proof. Suppose first that P is not a layered pattern and thus $[\![M, N, P]\!]$ is set-stable. If M and N are stable, then they are strongly normalizing by Lemma 5.19. Otherwise, we are in the case $Var(P) \cap FV(M) \neq \varnothing$ and $Match(N, P)$ is defined.

- Suppose that M is not strongly normalizing. Then there is a nonterminating reduction sequence

$$M \Rightarrow M_1 \Rightarrow M_2 \Rightarrow \cdots$$

from which we can construct, by Lemma 5.4, a nonterminating reduction sequence

$$M[Match(N, P)] \Rightarrow^+ M_1[Match(N, P)] \Rightarrow^+ M_2[Match(N, P)] \Rightarrow^+ \cdots$$

As $M[Match(N, P)]$ belongs to the set $[\![M, N, P]\!]$, it is stable by hypothesis and so strongly normalizing by Lemma 5.19, which leads to a contradiction.

- Suppose that N is not strongly normalizing. Then there is a nonterminating reduction sequence

$$N \Rightarrow N_1 \Rightarrow N_2 \Rightarrow \cdots$$

from which we can construct, by 5.4, a nonterminating reduction sequence

$$M[Match(N, P)] \Rightarrow^+ M[Match(N_1, P)] \Rightarrow^+ M[Match(N_2, P)] \cdots$$

As $M[Match(N, P)]$ belongs to $[\![M, N, P]\!]$, it is stable by hypothesis, so strongly normalizing by Lemma 5.19, which leads to a contradiction.

Now, if $P = P_1 @ P_2$, then $[\![M, \langle N, N\rangle, \langle P_1, P_2\rangle]\!]$ is set-stable and by the previous case M and $\langle N, N\rangle$ are strongly normalizing, so N is also strongly normalizing.

End of Proof.

Lemma 5.25. *If* $N \Rightarrow N'$ *and* $[\![M, N, P]\!]$ *is set-stable, then*$[\![M, N', P]\!]$ *is set-stable.*

Proof. Suppose first that P is not a layered pattern.

Let $M[Match(N'', P)]$ be a term in $[\![M, N', P]\!]$. Then $N \Rightarrow N' \Rightarrow^* N''$, so the term $M[Match(N'', P)]$ is in $[\![M, N, P]\!]$, and thus stable by hypothesis. Suppose M or N' are not stable. Then by Lemma 5.21, M or N are not stable and thus $Match(N, P)$ is defined and $Var(P) \cap FV(M) \neq \varnothing$. As a consequence $Match(N', P)$ is also defined by Proposition 5.2, and then $[\![M, N', P]\!]$ turns out to be set-stable by definition.

Now, suppose $P = P_1 @ P_2$. By hypothesis $[\![M, \langle N, N\rangle, \langle P_1, P_2\rangle]\!]$ is set-stable and by applying twice the previous case we have $[\![M, \langle N', N'\rangle, \langle P_1, P_2\rangle]\!]$ set-stable so that $[\![M, N', P_1 @ P_2]\!]$ is set-stable by definition.

End of Proof.

Lemma 5.26. *If* $[\![M, N, P]\!]$ *is set-stable, then* (*let N be P in M*) *is stable.*

Proof. Let $\bar{K}$ be stable terms such that (*let N be P in M*) $\bar{K}$ is not of functional type. If we show that the term (*let N be P in M*) $\bar{K}$ is stable, then the lemma follows from Proposition 5.16.

For that, let us first show that (*let N be P in M*) $\bar{K}$ is strongly normalizing. Consider any reduction sequence starting at (*let N be P in M*) $\bar{K}$.

- If the outermost constructor *let* is never removed, and the $(cont-let)$ is never applied to the root position, then reductions proceed only inside M, N, and $\bar{K}$. The terms M and N are strongly normalizing by Lemma 5.24 and the terms $\bar{K}$ are stable by hypothesis, so they are strongly normalizing by Lemma 5.19. As a consequence, the reduction sequence terminates.
- If the outermost construct *let* is removed, there are three cases:

— P is not a layered pattern. The reduction sequence looks like

$$(let\ N\ be\ P\ in\ M)\ \bar{K} \Rightarrow^* (let\ N'\ be\ P\ in\ M')\ \bar{K}' \Rightarrow M'[Match(N', P)]\ \bar{K}' \Rightarrow \cdots$$

where $M \Rightarrow^* M'$, $N \Rightarrow^* N'$ and $\bar{K} \Rightarrow^* \bar{K}'$.

Since $M[Match(N', P)]$ belongs to $[\![M, N, P]\!]$, it is stable by hypothesis, and so $M'[Match(N', P)]$ is stable by Lemma 5.4 and Lemma 5.21. Then $M'[Match(N', P)]\ \bar{K}'$ is stable by Proposition 5.16 and strongly normalizing by Lemma 5.19. As a consequence, such a reduction sequence also terminates.

— $P = P_1 @ P_2$ and the pattern $P_1 @ P_2$ is removed *before* the outermost constructor *let*. The reduction sequence looks like

$$\begin{aligned}
&(\mathit{let}\ N\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M)\ \bar{K}\\
&\qquad \Rightarrow^* (\mathit{let}\ N'\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M')\ \bar{K}'\\
&\qquad \Rightarrow (\mathit{let}\ \langle N', N'\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M')\ \bar{K}'\\
&\qquad \Rightarrow^* (\mathit{let}\ \langle N'', N'''\rangle\ \mathit{be}\ \langle P_1, P_2\rangle\ \mathit{in}\ M'')\ \bar{K}''\\
&\qquad \Rightarrow M''[\mathit{Match}(\langle N'', N'''\rangle, \langle P_1, P_2\rangle)]\ \bar{K}''\\
&\qquad \Rightarrow \cdots
\end{aligned}$$

where $N \Rightarrow^* N''$, $N \Rightarrow^* N'''$, $M \Rightarrow^* M''$ and $\bar{K} \Rightarrow^* \bar{K}''$.

Since $M[\mathit{Match}(\langle N'', N'''\rangle, \langle P_1, P_2\rangle)]$ belongs to $\llbracket M, \langle N, N\rangle, \langle P_1, P_2\rangle\rrbracket$, it is stable by hypothesis, and so $M''[\mathit{Match}(\langle N'', N'''\rangle, \langle P_1, P_2\rangle)]$ is stable by Lemma 5.4 and Lemma 5.21. We also have $\bar{K}''$ stable by Lemma 5.21, so $M''[\mathit{Match}(\langle N'', N'''\rangle, \langle P_1, P_2\rangle)]\ \bar{K}''$ is stable by Proposition 5.16 and strongly normalizing by Lemma 5.19.

— $P = P_1 @ P_2$ and the pattern $P_1 @ P_2$ is removed *after* the outermost constructor *let*. The reduction sequence looks like

$$\begin{aligned}
&(\mathit{let}\ N\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M)\ \bar{K}\\
&\qquad \Rightarrow^* (\mathit{let}\ N'\ \mathit{be}\ P_1 @ P_2\ \mathit{in}\ M')\ \bar{K}'\\
&\qquad \Rightarrow M'[\mathit{Match}(N', P_1 @ P_2)]\ \bar{K}'\\
&\qquad = M'[\mathit{Match}(\langle N', N'\rangle, \langle P_1, P_2\rangle)]\ \bar{K}'\\
&\qquad \Rightarrow
\end{aligned}$$

where $N \Rightarrow^* N'$, $M \Rightarrow^* M'$ and $\bar{K} \Rightarrow^* \bar{K}'$.

Since $M'[\mathit{Match}(\langle N', N'\rangle, \langle P_1, P_2\rangle)]$ belongs to $\llbracket M, \langle N, N\rangle, \langle P_1, P_2\rangle\rrbracket$, it is stable by hypothesis, and $\bar{K}'$ are stable by Lemma 5.21, so that $M'[\mathit{Match}(\langle N', N'\rangle, \langle P_1, P_2\rangle)]\ \bar{K}'$ is stable by Proposition 5.16 and strongly normalizing by Lemma 5.19.

To finish the proof suppose that $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ M)\ \bar{K}$ reduces to a pair $\langle L_1, L_2\rangle$ or to $\mathit{inl}(L)$ or to $\mathit{inr}(L)$. Then we have necessarily to remove the outermost constructor $\mathit{let}\_\mathit{be}\_\mathit{in}\_$ and we obtain a reduction sequence similar to one of the three last ones with more steps leading to one of the terms $\langle L_1, L_2\rangle$, $\mathit{inl}(L)$, or $\mathit{inr}(L)$. Since in the three cases, we have shown that we reach stable terms, then L_1, L_2 and L are stable, so we can conclude that $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ M)\ \bar{K}$ and thus $\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ M$ are stable.

End of Proof.

LEMMA 5.27. *If* $(\lambda P.J)\ N$ *is stable*, $\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J$ *is stable.*

Proof. Let $\bar{K}$ be stable terms such that $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \bar{K}$ is not a functional type. If we show that $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \bar{K}$ is stable, then the lemma follows from Proposition 5.16.

For that, let us first show that $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \bar{K}$ is strongly normalizing. Consider any reduction sequence starting at $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \bar{K}$.

- If the outermost constructor *let* is never removed, then reductions proceed inside N, J and $\bar{K}$. Since $(\lambda P.J)\ N$ is stable, then it is strongly normalizing by Lemma 5.19 and then J and N are strongly normalizing. The terms $\bar{K}$ are stable by hypothesis, so strongly normalizing by Lemma 5.19, so we can conclude that the reduction sequence necessarily terminates in this case.

- If the outermost constructor *let* is removed, then by reasoning in the same way we did in Lemma 5.26, we know that the term $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \bar{K}$ reduces to some term $(J'[\sigma])\ \bar{K}'$ such that $(\lambda P.J)\ N \Rightarrow^* J'[\sigma]$ and $\bar{K} \Rightarrow \bar{K}'$. By hypothesis the term $(\lambda P.J)\ N$ is stable, so $J'[\sigma]$ turns out to be stable by Lemma 5.21. Now, the terms $\bar{K}$ are stable, so $\bar{K}'$ are stable by Lemma 5.21 and $(J'[\sigma])\ \bar{K}'$ is stable by Proposition 5.16, thus strongly normalizing by Lemma 5.19. The reduction sequence also terminates in this case.

To finish the proof suppose the term $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \bar{K}$ reduces to a pair $\langle L_1, L_2\rangle$ or to $\mathit{inl}(L)$ or to $\mathit{inr}(L)$. Then we have necessarily to remove the outermost constructor *let* we obtain a reduction sequence from the term $(J[\sigma])\ \bar{K}'$ to one of the terms $\langle L_1, L_2\rangle$, or $\mathit{inl}(L)$ or $\mathit{inr}(L)$. Since this term is stable, then L_1, L_2 and L are stable, so we can conclude that $(\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \bar{K}$ and thus $\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J$ are stable.

End of Proof.

LEMMA 5.28. *If* $(\mathit{let}\ (\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \mathit{be}\ Q\ \mathit{in}\ M)$ *is stable, then* $((\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M)$ *is stable.*

Proof. Let $\bar{K}$ be stable terms such that $((\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M)\ \bar{K}$ is not of functional type. If we show that $((\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M)\ \bar{K}$ is stable, then the lemma follows from Proposition 5.16.

For that, let us first show that $((\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M)\ \bar{K}$ is strongly normalizing. Consider any reduction sequence starting at $((\lambda P.J)\ \mathit{of}\ N\ \mathit{is}\ Q\ \mathit{in}\ M)\ \bar{K}$.

Since $(\mathit{let}\ (\mathit{let}\ N\ \mathit{be}\ P\ \mathit{in}\ J)\ \mathit{be}\ Q\ \mathit{in}\ M)$ is stable, then it is strongly normalizing by Lemma 5.19 and then J, M, and N are strongly normalizing.

- If the outermost constructor $(\_\mathit{of}\_\mathit{is}\_\mathit{in}\_)$ is never removed, then reductions proceed inside M, N, J, and $\bar{K}$. The terms M and N and J are all strongly normalizing and the terms $\bar{K}$ are stable by hypothesis, so strongly normalizing by Lemma 5.19. That means that the reduction sequence necessarily terminates.

- If the outermost constructor $(\_\mathit{of}\_\mathit{is}\_\mathit{in}\_)$ is removed, then by reasoning in the same way we did in Lemma 5.26, we know that the term $((\lambda P.J)$ *of N is Q in M*$)\,\bar{K}$ reduces to some term (*let K be Q in M'*) $\bar{K}'$ such that (*let N be P in J*) $\Rightarrow^* K$, $M \Rightarrow M'$, and $\bar{K} \Rightarrow \bar{K}'$. By hypothesis the term (*let* (*let N be P in J*) *be Q in M*) is stable, so (*let* (*let N be P in J*) *be Q in M*) $\bar{K}$ is stable by Proposition 5.16, and (*lt K be Q in M'*) $\bar{K}'$ turns out to be stable by Lemma 5.21 and strongly normalizing by Lemma 5.19.

To finish the proof suppose the term $((\lambda P.J)$ *of N is Q in M*$)\,\bar{K}$ reduces to a pair $\langle L_1, L_2\rangle$ or to *inl*(L) or to *inr*(L). Then we have necessarily to remove the outermost constructors (*_of_is_in_*) and we obtain a reduction sequence from the term (*let K be Q in M'*) $\bar{K}'$ to one of the terms $\langle L_1, L_2\rangle$, or *inl*(L), or *inr*(L). By definition of stability L_1, L_2, and L are stable, so we can conclude that $((\lambda P.J)$ *of N is Q in M*$)\,\bar{K}$ and thus $((\lambda P.J)$ *of N is Q in M*$)$ are stable.

End of Proof.

LEMMA 5.29. *Every typed term is stable.*

Proof. To proof this property we need a stronger property that we expressed as follows:

Let M be a term such that all its free variables are among $\{x_i\}_{i=1\cdots n}$. If $N_1 \cdots N_n$ are stable terms such that $\theta = [N_1 \cdots N_n / x_1 \cdots x_n]$ is a well-typed substitution and $M[\theta]$ is a well-typed term, then $M[\theta]$ is stable.

The theorem will follow by taking $N_i = x_i$, so that the notation $[N_1 \cdots N_n / x_1 \cdots x_n]$ does not exactly correspond to the classical notation of substitutions where we only write N_i if it is different from x_i. We show the property by using the following remarks:

Remark 5.30.

1. If M is a term of type A, P is a pattern of type A, and $\mathit{Match}(M, P) = \sigma$, then σ is a well-typed substitution and $L[\theta]$ is a well-typed term if L is a well-typed term.

2. If M is stable and $\mathit{Match}(M, P) = \sigma$, then σ can be written as $[K_1 \cdots K_m / y_1 \cdots y_m]$, where $\mathit{Var}(P) = \{y_1, ..., y_m\}$ and every K_i is stable.

We proceed by induction on the structure of the term M. For that, in some of the cases that follow, to apply the induction hypothesis to some subterm R of M having a set of free variables $\{y_1, ..., y_m, x_1, ..., x_n\}$, we will also use the substitution $[\bar{y}/\bar{y}]$ which verifies the hypothesis since variables are stable by Corollary 5.20: indeed, the term $R[\theta][\bar{y}/\bar{y}] = R[\theta]$ will be stable, by the induction hypothesis.

In what follows,

- $M \equiv x_i$. Then $x_i[\theta] = N_i$ and N_i is stable by hypothesis.
- If $M \equiv \mathit{inl}_B(L)$, $M \equiv \mathit{inr}_B(L)$ or $M \equiv \langle M_1, M_2\rangle$ we apply the induction hypothesis and Lemma 5.22.
- $M \equiv [M_1 \mid_\xi M_2]$. By i.h. $M_1[\theta]$ and $M_2[\theta]$ are stable and by hypothesis $\xi[\theta]$ is a communication term. Then $[M_1[\theta] \mid_{\xi[\theta]} M_2[\theta]]$ is stable by Lemma 5.22.
- $M \equiv$ *let L be P in R*. We will show that the set$[\![R[\theta], L[\theta], P]\!]$ is set-stable, so $M[\theta] =$ *let* $L[\theta]$ *be P in* $R[\theta]$ will be stable by Lemma 5.26.

Now, let $\mathit{Var}(P) = \{y_1, ..., y_m\}$ $(m \geqslant 0)$. By α-conversion we can assume that $\{y_1, ..., y_m\} \cap \{x_1, ..., x_n\} = \varnothing$, so by i.h. $L[\theta]$ and $R[\theta][\bar{y}/\bar{y}] = R[\theta]$ are stable (variables are stable by Corollary 5.20), and by Lemma 5.22 $\langle L[\theta], L[\theta]\rangle$ is also stable.

Suppose that P is not a layered pattern. Then to show that $[\![R[\theta], L[\theta], P]\!]$ is set-stable it is sufficient to show that every term in the set is stable. Let us take any term $R[\theta][\mathit{Match}(L', P)]$ in the set satisfying the conditions: we have $L[\theta] \Rightarrow^* L'$, so L' is stable by Lemma 5.21. By Remark 5.30 the term $R[\theta][\mathit{Match}(L', P)]$ satisfies the conditions of the hypothesis, so it is stable by i.h., which concludes the proof in this case.

Suppose that P is a layered pattern $P_1 @ P_2$. Then to show that $[\![R[\theta], L[\theta], P]\!]$ is set-stable it is sufficient to show that $[\![R[\theta], \langle L[\theta], L[\theta]\rangle, \langle P_1, P_2\rangle]\!]$ is set-stable. Let us take any term $R[\theta][\mathit{Match}(\langle L', L''\rangle, \langle P_1, P_2\rangle)]$ in the set satisfying the conditions: we have $\langle L[\theta], L[\theta]\rangle \Rightarrow^* \langle L', L''\rangle$, so $\langle L', L''\rangle$ is stable by Lemma 5.21. By Remark 5.30 the term $R[\theta][\mathit{Match}(\langle L', L''\rangle, \langle P_1, P_2\rangle)]$ satisfies the conditions of the hypothesis so it is stable by i.h., which concludes the proof in this case.

- $M \equiv L$ *of K is Q in R*. Let $\mathit{Var}(Q) = \{y_1, ..., y_m\}$ $(m \geqslant 0)$. By α-conversion we can assume that $\{y_1, ..., y_m\} \cap \{x_1, ..., x_n\} = \varnothing$, so by i.h. $K[\theta]$ and $R[\theta][\bar{y}/\bar{y}] = R[\theta]$ are stable (variables are stable by Corollary 5.20).

If $L[\theta]$ is a variable, then the property holds by Lemma 5.22. Otherwise, $L[\theta]$ is a λ-abstraction $\lambda P.J$ (that is stable by i.h.). Let us show that

1. *let* $K[\theta]$ *be P in J* is stable: Since $\lambda P.J$ is stable, then $(\lambda P.J)\, K[\theta]$ is stable by definition and *let* $K[\theta]$ *be P in J* is stable by Lemma 5.27.

2. $[\![R[\theta]$, (*let* $K[\theta]$ *be P in J*), $Q]\!]$ is set-stable: There are two cases to consider

— Q is not a layered pattern: Since $R[\theta]$ is stable by i.h. and *let* $K[\theta]$ *be P in J* is stable by the previous point, it is sufficient to show that every term in the set $[\![R[\theta]$, *let* $K[\theta]$ *be P in J*, $Q]\!]$ is stable. Let us take any term $R[\theta][\mathit{Match}(J', Q)]$ in the set such that *let* $K[\theta]$ *be P in* $J \Rightarrow^* J'$. By Lemma 5.21 J' is stable and by Remark 5.30 the term $R[\theta][\mathit{Match}(J', Q)]$ satisfies the conditions of the hypothesis, so it is stable by i.h.

— Q is a layered pattern $Q_1 @ Q_2$: Since $R[\theta]$ is stable by i.h. and *let* $K[\theta]$ *be P in J* is stable by the previous point, we have $\langle$(*let* $K[\theta]$ *be P in J*), (*let* $K[\theta]$ *be P in J*)$\rangle$ stable by Lemma 5.22 and then it is sufficient to show that every term in $[\![R[\theta], \langle$(*let* $K[\theta]$ *be P in J*), (*let* $K[\theta]$ *be P*

$in\ J)\rangle, Q_1 @ Q_2\rrbracket$ is stable. Let us take any term $R[\theta][Match(\langle J', J''\rangle, Q)]$ in the set such that

$$\langle (let\ K[\theta]\ be\ P\ in\ J), (let\ K[\theta]\ be\ P\ in\ J)\rangle \Rightarrow^* \langle J', J''\rangle.$$

By Lemma 5.21 $\langle J', J''\rangle$ is stable and by Remark 5.30 the term $R[\theta][Match(\langle J', J''\rangle, \langle Q_1, Q_2\rangle)]$ satisfies the conditions of the hypothesis so it is stable by i.h.

Now, by Lemma 5.26, the term $let\ (let\ K[\theta]\ be\ P\ in\ J)\ be\ Q\ in\ R[\theta]$ turns out to be stable and by Lemma 5.28 $M[\theta] = L[\theta]\ of\ K[\theta]\ is\ Q\ in\ R[\theta]$ is stable.

- $M \equiv \lambda P{:}B.J$. Then $(\lambda P{:}B.J)[\theta] \equiv \lambda P{:}B.J[\theta]$. Consider any stable term R of type B. If $(\lambda P{:}B.J[\theta])\ R$ is shown to be stable, $(\lambda P{:}B.J[\theta])$ is stable by definition of stability.

So let us show that $(\lambda P{:}B.J[\theta])\ R = let\ \lambda P{:}B.J[\theta]\ be\ \sharp z\ in\ (z\ of\ R\ is\ y\ in\ y)$ (with z a fresh variable) is stable, using the following properties:

— The set $\llbracket J[\theta], R, P\rrbracket$ is set-stable: Let $Var(P) = \{y_1, ..., y_m\}$ $(m \geq 0)$. By α-conversion we can assume that $\{y_1, ..., y_m\} \cap \{x_1, ..., x_n\} = \varnothing$, so by i.h. and the fact that variables are stable (Corollary 5.20), we get that $J[\theta][\bar{y}/\bar{y}] = J[\theta]$ is stable. The term R is stable by hypothesis, so $\langle R, R\rangle$ is stable by Lemma 5.22. There are two cases to consider:

$*$ P is not a layered pattern, so to show that $\llbracket J[\theta], R, P\rrbracket$ is set-stable it is sufficient to show that every term in the set is stable. Let us take any term $J[\theta][Match(R', P)]$ in the set satisfying the conditions: we have $R \Rightarrow^* R'$, so R' is stable by Lemma 5.21. By Remark 5.30 the term $J[\theta][Match(R', P)]$ satisfies the conditions of the hypothesis so that it is stable by i.h.

$*$ P is a layered pattern $P_1 @ P_2$, so to show that $\llbracket J[\theta], R, P_1 @ P_2\rrbracket$ is set-stable it is sufficient to show that $\llbracket J[\theta], \langle R, R\rangle, \langle P_1, P_2\rangle\rrbracket$ is set-stable. Since $\langle R, R\rangle$ is stable by Lemma 5.22 and $J[\theta]$ is stable by i.h. it will be sufficient to show that every term in the set $\llbracket J[\theta], \langle R, R\rangle, \langle P_1, P_2\rangle\rrbracket$ is stable. Let us take any term $J[\theta][Match(\langle R', R''\rangle, \langle P_1, P_2\rangle\rrbracket$ in the set satisfying the conditions: we have $\langle R, R\rangle \Rightarrow^* \langle R', R''\rangle$, so $\langle R', R''\rangle$ is stable by Lemma 5.21. By Remark 5.30 the term $J[\theta][Match(\langle R', R''\rangle, \langle P_1, P_2\rangle)]$ satisfies the conditions of the hypothesis so that it is stable by i.h.

— $let\ R\ be\ P\ in\ J[\theta]$ is stable: by the previous point and Lemma 5.26.

— $(let\ (let\ R\ be\ P\ in\ J[\theta])\ be\ y\ in\ y)$ is stable: it is sufficient to show that $\llbracket y, let\ R\ be\ P\ in\ L[\theta], y\rrbracket$ is set-stable, so by the previous point and the fact that variables are stable, it is sufficient to show that every term in the set is stable. As $y[Match(let\ R\ be\ P\ in\ L[\theta], y)] = let\ R\ be\ P\ in\ L[\theta]$ is stable by the previous point, then by Lemma 5.21 every term in the set is stable.

— The term $((\lambda P.J[\theta])\ of\ R\ is\ y\ in\ y)$ is stable as a consequence of the previous point and Lemma 5.28.

— The term $(z\ of\ R\ is\ y\ in\ y)[H/z]$ is stable for any H such that $\lambda P{:}B.J[\theta] \Rightarrow^* H$, since z is a fresh variable, this property holds by the previous point and by Lemma 5.21.

— The set $\llbracket (z\ of\ R\ is\ y\ in\ y), \lambda P{:}B.J[\theta], \sharp z\rrbracket$ is set-stable: we have

$*$ $Var(\sharp z) \cap FV(z\ of\ R\ is\ y\ in\ y) \neq \varnothing$

$*$ $Match(\lambda P{:}B.J[\theta], \sharp z)$ is defined

$*$ $(z\ of\ R\ is\ y\ in\ y)[H/z]$ is stable for any H such that $\lambda P{:}B.J[\theta] \Rightarrow^* H$ (by the previous point).

As $\llbracket (z\ of\ R\ is\ y\ in\ y), \lambda P{:}B.J[\theta], \sharp z\rrbracket$ is set-stable, then $let\ \lambda P{:}B.J[\theta]\ be\ \sharp z\ in\ (z\ of\ R\ is\ y\ in\ y)$ is stable by Lemma 5.26.

End of Proof.

As a consequence we obtain:

THEOREM 5.31 (Strong Normalization). *If $\Gamma \rhd M{:}A$ then M is $\Rightarrow$-strongly normalizing.*

Proof. By Theorem 5.29 and Lemma 5.19.

End of Proof.

6. EVALUATORS IN STRUCTURED OPERATIONAL SEMANTICS STYLE

Now, to connect the basic properties of the general reduction system with the basic properties of the evaluators in natural semantics style we gave in Section 3, we define two evaluators (for closed terms) in the style of Plotkin's structured operational semantics (SOS) [21]: $\Rightarrow_{\mathrm{l}}$ and $\Rightarrow_{\mathrm{e}}$ for the lazy, respectively eager, strategy. The informal meaning of $M \Rightarrow_{\varepsilon} N$, where the symbol "ε" stands for either "lazy" or "eager," is that the closed term M evaluates *in one step* to the closed term N. For both evaluators, the relation $\Rightarrow_{\varepsilon}$ will be a subset of $\Rightarrow$ and we will show the following properties:

1. If $M \Downarrow_{\varepsilon} K$ then $M \Rightarrow_{\varepsilon}^* K$ ($\Rightarrow_{\varepsilon}^*$ is the transitive–reflexive closure of $\Rightarrow_{\varepsilon}$).
2. If $M \Rightarrow_{\varepsilon}^* K$ and K is an ε-canonical form then $M \Downarrow_{\varepsilon} K$.
3. If M is well typed and not an ε-canonical form then we have $M \Rightarrow_{\varepsilon} N$ for some N.

These properties are broken down among the propositions that appear below under the headings "adequacy."

The point of it all is that using the previous three properties and the fact that $\Rightarrow_{\varepsilon} \subseteq \Rightarrow$, Theorems 3.6 (type preservation) and 3.7 (convergence) follow immediately from Theorems 5.8 (subject reduction) and 5.31 (strong-normalization).[7]

6.1. A Lazy Evaluator

The lazy evaluator in SOS semantics style appears in Table 10, where the $Match(\_, \_)$ operation is the same as in Section 3.2. The evaluation and matching "call" each other as in the lazy evaluator in natural semantics style presented in Section 3.1. The reduction relation $\overset{P}{\rightsquigarrow}$ makes it possible to evaluate expressions under constructors only when they cannot be matched with the pattern P. When a pair $\langle M_1, M_2 \rangle$ is reduced by a relation $\overset{\langle P_1, P_2 \rangle}{\rightsquigarrow}$, evaluation proceeds in the component that has no enough information to be match with the pattern $\langle P_1, P_2 \rangle$.

As stated in Section 6, well-typed closed terms can always be reduced to a lazy-canonical form with our lazy evaluator described in Table 10.

LEMMA 6.1. *If M is well-typed and not a lazy-canonical form then we have $M \Rightarrow_1 N$ for some N.*

Proof. We show two properties at the same time:

1. If M is well-typed and not a lazy-canonical form then we have $M \Rightarrow_1 N$ for some N.

2. If $P \neq P_1 @ P_2$, $Match(M, P)$ fails, and M is a well-typed lazy-canonical form, then there is M' such that $M \overset{P}{\rightsquigarrow} M'$.

We proceed by induction on the structure of M as follows:

1.

- $M = [M_1 \mid_{\mathbf{L}} M_2] \Rightarrow_1 M_1$.
- $M = [M_1 \mid_{\mathbf{R}} M_2] \Rightarrow_1 M_2$.
- $M = let\ M_1\ be\ P\ in\ M_2$.

If $Match(M_1, P) = \sigma$, then $let\ M_1\ be\ P\ in\ M_2 \Rightarrow_1 M_2[\sigma]$. If $Match(M_1, P)$ fails, there are two cases:

— $P = P_1 @ P_2$. Then, $let\ M_1\ be\ P_1 @ P_2\ in\ M_2 \Rightarrow_1 let\ \langle M_1, M_1 \rangle\ be\ \langle P_1, P_2 \rangle\ in\ M_2$

— $P \neq P_1 @ P_2$. If M_1 is not a lazy-canonical form, then $M_1 \Rightarrow_1 M'_1$ by i.h. and $M_1 \overset{P}{\rightsquigarrow} M'_1$ by definition. Otherwise, there is M'_1 such that $M_1 \overset{P}{\rightsquigarrow} M'_1$ by i.h. In both cases we obtain $let\ M_1\ be\ P\ in\ M_2 \Rightarrow_1 let\ M'_1\ be\ P\ in\ M_2$.

- $M = (\lambda P.J)\ of\ M_1\ is\ Q\ in\ M_2$.

TABLE 10

Lazy Evaluator in SOS Semantics Style

$$\frac{Match(M, P) = \sigma}{let\ M\ be\ P\ in\ N \Rightarrow_1 N[\sigma]} \qquad \frac{Match(M, P) = \sigma}{(\lambda P.J)\ of\ M\ is\ Q\ in\ N \Rightarrow_1 let\ J[\sigma]\ be\ Q\ in\ N}$$

$$\frac{}{[M \mid_{\mathbf{L}} N] \Rightarrow_1 M} \qquad \frac{}{[M \mid_{\mathbf{R}} N] \Rightarrow_1 N}$$

$$\frac{Match(M, P_1 @ P_2)\text{ fails}}{let\ M\ be\ P_1 @ P_2\ in\ N \Rightarrow_1 let\ \langle M, M \rangle\ be\ \langle P_1, P_2 \rangle\ in\ N}$$

$$\frac{Match(M, P_1 @ P_2)\text{ fails}}{(\lambda P_1 @ P_2.J)\ of\ M\ is\ Q\ in\ N \Rightarrow_1 (\lambda \langle P_1, P_2 \rangle.J)\ of\ \langle M, M \rangle\ is\ Q\ in\ N}$$

$$\frac{Match(M, P)\text{ fails} \qquad M \overset{P}{\rightsquigarrow} M'}{let\ M\ be\ P\ in\ N \Rightarrow_1 let\ M'\ be\ P\ in\ N}$$

$$\frac{Match(M, P)\text{ fails} \qquad M \overset{P}{\rightsquigarrow} M'}{(\lambda P.J)\ of\ M\ is\ Q\ in\ N \Rightarrow_1 (\lambda P.J)\ of\ M'\ is\ Q\ in\ N}$$

$$\frac{M \overset{P_1}{\rightsquigarrow} M'}{inl(M) \overset{(P_1 \mid_\xi P_2)}{\rightsquigarrow} inl(M')} \qquad \frac{M \overset{P_2}{\rightsquigarrow} M'}{inr(M) \overset{(P_1 \mid_\xi P_2)}{\rightsquigarrow} inr(M')}$$

$$\frac{Match(M_1, P_1)\text{ fails} \qquad M_1 \overset{P_1}{\rightsquigarrow} M'_1}{\langle M_1, M_2 \rangle \overset{\langle P_1, P_2 \rangle}{\rightsquigarrow} \langle M'_1, M'_2 \rangle} \qquad \frac{Match(M_2, P_2)\text{ fails} \qquad M_2 \overset{P_2}{\rightsquigarrow} M'_2}{\langle M_1, M'_2 \rangle \overset{\langle P_1, P_2 \rangle}{\rightsquigarrow} \langle M_1, M'_2 \rangle}$$

$$\frac{M \Rightarrow_1 M'}{M \overset{P}{\rightsquigarrow} M'}$$

If $Match(M_1, P) = \sigma$, then $(\lambda P.J)\ of\ M_1\ is\ Q\ in\ M_2 \Rightarrow_1 let\ J[\sigma]\ be\ Q\ in\ M_2$.
If $Match(M_1, P)$ fails, there are two cases:

— $P = P_1 @ P_2$. Then, $(\lambda P_1 @ P_2.J)\ of\ M_1\ is\ Q\ in\ M_2 \Rightarrow_1 (\lambda \langle P_1, P_2 \rangle.J)\ of\ \langle M_1, M_1 \rangle\ is\ Q\ in\ M_2$

— $P \neq P_1 @ P_2$. If M_1 is not a lazy-canonical form, then $M_1 \Rightarrow_1 M'_1$ by i.h. and $M_1 \overset{P}{\rightsquigarrow} M'_1$ by definition. Otherwise, there is M'_1 such that $M_1 \overset{P}{\rightsquigarrow} M'_1$ by i.h. In both cases we obtain $(\lambda P.J)\ of\ M_1\ is\ Q\ in\ M_2 \Rightarrow_1 (\lambda P.J)\ of\ M'_1\ is\ Q\ in\ M_2$.

2. We proceed by induction on the structure of P using the induction hypothesis.

End of Proof.

6.1.1. *Adequacy*

In this section we show the connection between the lazy evaluators in Tables 10 and 3. Lemma 6.4 states that any result obtained via the lazy evaluator in 3 can be obtained by several one-step reductions, while Lemma 6.6 states that a lazy-canonical form reached by several one-step reductions can also be reached using the lazy evaluator of Table 3. The connection between the lazy and eager pattern matching predicates is given by the following lemma:

[7] The determinacy of $\Downarrow_1$, $\Downarrow_e$ does not seem to be a direct consequence of the confluence of $\Rightarrow$ because the ε-canonical forms are not necessarily normal forms for $\Rightarrow$, since we can reduce the open subterms under an abstraction.

Lemma 6.2.

1. *If match* M *on* $P \Downarrow_1 \sigma$ *and* $Match(M, P) = \sigma'$, *then* $\sigma \equiv \sigma'$.

2. *If match*$(M, P) = \sigma$ *then Match* M *on* $P \Downarrow_1 \sigma$.

Proof. By induction on the structure of P.

End of Proof.

The converse does not hold: suppose P is the pattern $\langle x, y\rangle$ and M is not a pair but $M \Downarrow_1 \langle M_1, M_2\rangle$. Then *match* M *on* $\langle x, y\rangle \Downarrow_1 [M_1/x] \cup [M_2/y]$ but the substitution $Match(M, \langle x, y\rangle)$ is not defined. The following proposition shows that only terms that are not matched by the pattern P are reducible by the reduction relation $\overset{P}{\rightsquigarrow}$.

Proposition 6.3. *If* $M \overset{P}{\rightsquigarrow}{}^* M'$, $P \neq P_1 @ P_2$, *and* $Match(M, P)$ *fails, then let* M *be* P *in* $N \Rightarrow_1^*$ *let* M' *be* P *in* N *and* $\lambda P.J$ *of* M *is* Q *in* $N \Rightarrow_1^* \lambda P.J$ *of* M' *is* Q *in* N.

Proof. By induction on the number of steps from M to M' and the structure of P.

End of Proof.

The following proposition gives the correspondence between the relations $\Downarrow_1$ and $\Rightarrow_1$. This correspondence shows that the relation $\Rightarrow_1$ represents one step of evaluation, and the relation $\Downarrow_1$ can be stimulated with many steps of $\Rightarrow_1$.

Proposition 6.4. *If* $M \Downarrow_1 K$ *then* $M \Rightarrow_1^* K$.

Proof. We show the following two properties at the same time by induction on the length of the derivation $\Downarrow_1$.

1. If $M \Downarrow_1 K$ then $M \Rightarrow_1^* K$

2. If *match* M *on* $P \Downarrow_1 \sigma$ and $P \neq P_1 @ P_2$, then there is M' such that $M \overset{P}{\rightsquigarrow}{}^* M'$ and $Match(M', P) = \sigma$.

1. The cases to consider are:

- *let* M_1 *be* P *in* $M_2 \Downarrow_1 K$ comes from *match* M_1 *on* $P \Downarrow_1 \sigma$ and $M_2[\sigma] \Downarrow_1 K$.

If $Match(M_1, P) = \sigma'$, then $\sigma = \sigma'$ by Lemma 6.2 and *let* M_1 *be* P *in* $M_2 \Rightarrow_1 M_2[\sigma]$. Since $M_2[\sigma] \Rightarrow_1^* K$ by i.h., then *let* M_1 *be* P *in* $M_2 \Rightarrow_1^* K$.

If $Match(M_1, P)$ fails, we distinguish two cases.

— If $P \neq P_1 @ P_2$, by i.h. there is a term M'_1 such that $M_1 \overset{P}{\rightsquigarrow}{}^* M'_1$ and $Match(M'_1, P) = \sigma$. Therefore there is a reduction sequence *let* M_1 *be* P *in* $M_2 \Rightarrow_1^*$ *let* M'_1 *be* P *in* $M_2 \Rightarrow_1 M_2[\sigma]$ by Proposition 6.3. Since $M_2[\sigma] \Rightarrow_1^* K$ by i.h., then *let* M_1 *be* P *in* $M_2 \Rightarrow_1^* K$.

— If $P = P_1 @ P_2$, then *let* M_1 *be* $P_1 @ P_2$ *in* $M_2 \Rightarrow_1$ *let* $\langle M_1, M_1\rangle$ *be* $\langle P_1, P_2\rangle$ *in* M_2. We can show by the previous case that *let* $\langle M_1, M_1\rangle$ *be* $\langle P_1, P_2\rangle$ *in* $M_2 \Rightarrow_1^* K$ because *match* M_1 *on* $P_1 @ P_2 \Downarrow_1 \sigma$ if and only if *match* $\langle M_1, M_1\rangle$ *on* $\langle P_1, P_2\rangle \Downarrow_1 \sigma$ and both derivations have the same length.

- $(\lambda P.J)$ *of* M_1 *is* Q *in* $M_2 \Downarrow_1 K$ comes from *match* M_1 *on* $P \Downarrow_1 \sigma$, *let* $J[\sigma]$ *be* Q *in* $M_2 \Downarrow_1 K$.

If $Match(M_1, P) = \sigma'$, then $\sigma = \sigma'$ by Lemma 6.2 and $(\lambda P.J)$ *of* M_1 *is* Q *in* $M_2 \Rightarrow_1$ *let* $J[\sigma]$ *be* Q *in* M_2. By i.h. *let* $J[\sigma]$ *be* Q *in* $M_2 \Rightarrow_1^* K$ and then $(\lambda P.J)$ *of* M_1 *is* Q *in* $M_2 \Rightarrow_1^* K$.

If $Match(M_1, P)$ fails, we consider two cases:

— If $P \neq P_1 @ P_2$, by i.h. there is M'_1 such that $M_1 \overset{P}{\rightsquigarrow}{}^* M'_1$ and $Match(M'_1, P) = \sigma$. By Proposition 6.3 we can construct a reduction sequence

$$\begin{aligned} &(\lambda P.J) \textit{ of } M_1 \textit{ is } Q \textit{ in } M_2 \\ &\quad \Rightarrow_1^* (\lambda P.J) \textit{ of } M'_1 \textit{ is } Q \textit{ in } M_2 \\ &\quad \Rightarrow_1 \textit{let } J[\sigma] \textit{ be } Q \textit{ in } M_2 \end{aligned}$$

Since *let* $J[\sigma]$ *be* Q *in* $M_2 \Rightarrow_1^* K$ by i.h., then $(\lambda P.J)$ *of* M_1 *is* Q *in* $M_2 \Rightarrow_1^* K$.

— If $P = P_1 @ P_2$, then $(\lambda P_1 @ P_2.J)$ *of* M_1 *is* Q *in* $M_2 \Rightarrow (\lambda\langle P_1, P_2\rangle.J)$ *of* $\langle M_1, M_1\rangle$ *is* Q *in* M_2. Since *match* M_1 *on* $P_1 @ P_2 \Downarrow_1 \sigma$ if and only if *match* $\langle M_1, M_1\rangle$ *on* $\langle P_1, P_2\rangle \Downarrow_1 \sigma$ and both derivations have the same length, then $(\lambda\langle P_1, P_2\rangle.J)$ *of* $\langle M_1, M_1\rangle$ *is* Q *in* $M_2 \Rightarrow_1^* K$ can be shown by the previous case.

- $[M_1 |_{\mathbf{L}} M_2] \Downarrow_1 K$ comes from $M_1 \Downarrow_1 K$. By i.h. $M_1 \Rightarrow_1^* K$ and then $[M_1 |_{\mathbf{L}} M_2] \Rightarrow_1 M_1 \Rightarrow_1^* K$. The case $[M_1 |_{\mathbf{R}} M_2] \Downarrow_1 K$ is symmetrical.

2. If $Match(M, P) = \sigma'$, then $\sigma \equiv \sigma'$ by Lemma 6.2 and we just take $M' = M$.

Suppose $Match(M, P)$ fails. We proceed by induction on the structure of the pattern P.

- $P = \sharp z$. By definition $\sigma = [\lambda R.L/z]$ where $M \Downarrow_1 \lambda R.L$. By i.h. $M \Rightarrow_1^* \lambda R.L$ and thus $M \overset{\sharp z}{\rightsquigarrow}{}^* \lambda R.L$.

- $P = (P_1 |_{\xi} P_2)$. Suppose $\sigma = [\mathbf{L}/\xi] \cup [\rho]$ where $M \Downarrow_1 inl(L)$ and *match* L *on* $P_1 \Downarrow_1 \rho$. By i.h. $M \Rightarrow_1^* inl(L)$ and by i.h. there is L' such that $L \overset{P_1}{\rightsquigarrow}{}^* L'$ and $Match(L', P_1) = \rho$. Take $M' = inl(L')$. We have $Match(inl(L'), (P_1 |_{\xi} P_2)) = \sigma$ and $M \overset{(P_1 |_{\xi} P_2)}{\rightsquigarrow}{}^* inl(L) \overset{(P_1 |_{\xi} P_2)}{\rightsquigarrow} inl(L')$. The case $\sigma = [\mathbf{R}/\xi] \cup [\rho]$ is symmetrical.

- $P = \langle P_1, P_2\rangle$. By definition $\sigma = \sigma_1, \sigma_2$ where $M \Downarrow_1 \langle L_1, L_2\rangle$ and *match* L_i *on* $P_i \Downarrow_1 \sigma_i$ for $i = 1, 2$. By i.h. $M \Rightarrow_1^* \langle L_1, L_2\rangle$ and then $M \overset{\langle P_1, P_2\rangle}{\rightsquigarrow}{}^* \langle L_1, L_2\rangle$. We will construct $M' = \langle L'_1, L'_2\rangle$ satisfying the conditions of the proposition.

If $Match(L_1, P_1) = \rho_1$, then $\sigma_1 = \rho_1$ by Lemma 6.2 and we just take $L'_1 = L_1$.

If $Match(L_1, P_1)$ fails, then by i.h. there is L'_1 such that $L_1 \overset{P_1}{\rightsquigarrow}{}^* R_1 \overset{P_1}{\rightsquigarrow} L'_1$ and $Match(L'_1, P_1) = \sigma_1$.

Since *Match*(R_1, P_1) fails by Proposition 6.3, we have $\langle L_1, L_2\rangle \overset{\langle P_1, P_2\rangle}{\rightsquigarrow}{}^* \langle R_1, L_2\rangle \overset{\langle P_1, P_2\rangle}{\rightsquigarrow} \langle L'_1, L_2\rangle$. We proceed in the same way according to *Match*$(L_2, P_2)=\rho_2$ or *Match*(L_2, P_2) fails.

End of Proof.

Lemma 6.5. *If $M \Rightarrow_1^* K$ implies that $M \Downarrow_1 K$, where K is a lazy canonical form, then for every pattern $P \neq P_1 @ P_2$ such that $M \overset{P}{\rightsquigarrow}{}^* M'$ and Match$(M', P)=\sigma$, match N on $P \Downarrow_1 \sigma$ holds.*

Proof. By induction on the structure of P.

- $P = \_$ or $P = x$. These cases are trivial.
- $P = \sharp z$. Then $M' = \lambda Q.J$ and $M \Rightarrow_1^* \lambda Q.J$. By hypothesis $M \Downarrow_1 \lambda Q.J$ and then *match* M *on* $\sharp z \Downarrow_1 [\lambda Q.J/z]$.
- $P = \langle P_1, P_2\rangle$. Then $M' = \langle M_1, M_2\rangle$ and $\sigma = \sigma_1, \sigma_2$, where *Match*$(M_i, P_i)=\sigma_i$ for $i=1, 2$. Take the first pair $\langle L_1, L_2\rangle$ appearing in the reduction sequence from M to M'. Necessarily $M \Rightarrow_1^* \langle L_1, L_2\rangle$ and $L_i \overset{P_i}{\rightsquigarrow}{}^* M_i$. By hypothesis $M \Downarrow_1 \langle L_1, L_2\rangle$ and by i.h. *match* L_i *on* $P_i \Downarrow_1 \sigma_i$ so that *match* M *on* $\langle P_1, P_2\rangle \Downarrow_1 \sigma$ holds.
- $P = (P_1 |_\xi P_2)$. If $M' = inl(N)$, then $\sigma = [\mathbf{L}/\xi], \rho$. Take the first term of the form $inl(L)$ term appearing in the reduction sequence from M to M'. Then $M \overset{(P_1 |_\xi P_2)}{\rightsquigarrow}{}^* inl(L) \overset{(P_1 |_\xi P_2)}{\rightsquigarrow}{}^* inl(N)$. We have necessarily $M \Rightarrow_1^* inl(L)$ and $L \overset{P_1}{\rightsquigarrow}{}^* N$. By hypothesis $M \Downarrow_1 inl(L)$ and by i.h. *match* L *on* $P_1 \Downarrow_1 \rho$. We can conclude *match* M *on* $(P_1 |_\xi P_2) \Downarrow_1 \sigma$. The case $M' = inr(N)$ is symmetrical.

End of Proof.

Proposition 6.6. *If $M \Rightarrow_1^* K$ and K is a lazy canonical form, then $M \Downarrow_1 K$.*

Proof. By induction on the number of reduction steps from M to K.

If there are 0 steps from M to K, then M is a lazy canonical form and thus $K=M$ and $K \Downarrow_1 K$ by definition of $\Downarrow_1$. Suppose there are $n>0$ steps from M to K. We proceed by induction on the structure of M, we proceed by induction on M as follows:

$M=\langle M_1, M_2\rangle$, $M=\lambda P.J$, $M=inl(N)$ or $M=inr(M)$, $K=M$ and $K \Downarrow_1 K$ trivially holds. Suppose there are $n>0$ steps from M to K. We proceed by induction on the structure of M, we are left to consider the following possible cases:

- If either $M=\langle M_1, M_2\rangle$ or $M=\lambda P.J$ or $M=inl(N)$ or $M=inr(M)$, then the only possible case is $K=M$ and thus $n \not< 0$.
- $M=[M_1 |_\mathbf{L} M_2]$. Then the reduction sequence looks like

$$[M_1 |_\mathbf{L} M_2] \Rightarrow_1 M_1 \Rightarrow_1^* K$$

By the first induction hypothesis $M_1 \Downarrow_1 K$ and therefore $[M_1 |_\mathbf{L} M_2] \Downarrow_1 K$. The case $M=[M_1 |_\mathbf{R} M_2]$ is symmetrical.

- $M = let\ M_1\ be\ P\ in\ M_2$.

If *Match*$(M_1, P)=\sigma$, then *match* M_1 *on* $P \Downarrow_1 \sigma$ by Lemma 6.2 and the sequence looks like

$$let\ M_1\ be\ P\ in\ M_2 \Rightarrow_1 M_2[\sigma] \Rightarrow_1^* K$$

By the first induction hypothesis $M_2[\sigma] \Downarrow_1 K$ so we can conclude *let* M_1 *be* P *in* $M_2 \Downarrow_1 K$.

If *Match*(M_1, P) fails, there are two cases to consider:

— If $P \neq P_1 @ P_2$, the reduction sequence looks like

$$let\ M_1\ be\ P\ in\ M_2 \Rightarrow_1^* let\ M'_1\ be\ P\ in\ M_2 \Rightarrow_1 M_2[\sigma] \Rightarrow_1^* K$$

where $M_1 \overset{P}{\rightsquigarrow}{}^* M'_1$ and *Match*$(M'_1, P)=\sigma$. The property holds for M_1 by the second i.h., then by Lemma 6.5 *match* M_1 *on* $P \Downarrow_1 \sigma$, by the first i.h. $M_2[\sigma] \Downarrow_1 K$ and thus *let* M_1 *be* P *in* $M_2 \Downarrow_1 K$.

— If $P = P_1 @ P_2$, the reduction sequence looks like

$$\begin{aligned} &let\ M_1\ be\ P_1 @ P_2\ in\ M_2 \\ &\qquad \Rightarrow_1^* let\ \langle M_1, M_1\rangle\ be\ \langle P_1, P_2\rangle\ in\ M_2 \Rightarrow_1^* K \end{aligned}$$

By the first i.h. *let* $\langle M_1, M_1\rangle$ *be* $\langle P_1, P_2\rangle$ *in* $M_2 \Downarrow_1 K$, i.e., we have *match* $\langle M_1, M_1\rangle$ *on* $\langle P_1, P_2\rangle \Downarrow_1 \sigma$ and $M_2[\sigma] \Downarrow_1 K$. From the last match, we have $\sigma=\sigma_1, \sigma_2$ where *match* M_1 *on* $P_1 \Downarrow_1 \sigma_1$ and *match* M_1 *on* $P_2 \Downarrow_1 \sigma_2$ and thus *match* M_1 *on* $P_1 @ P_2 \Downarrow_1 \sigma$ holds, which makes it possible to conclude *let* M_1 *be* $P_1 @ P_2$ *in* $M_2 \Downarrow_1 K$.

- $M = (\lambda P.J)\ of\ M_1\ is\ Q\ in\ M_2$.

If *Match*$(M_1, P)=\sigma$, then *match* M_1 *on* $P \Downarrow_1 \sigma$ by Lemma 6.2 and the sequence looks like

$$(\lambda P.J)\ of\ M_1\ is\ Q\ in\ M_2 \Rightarrow_1 let\ J[\sigma]\ be\ Q\ in\ M_2 \Rightarrow_1^* K$$

By the first i.h. *let* $J[\sigma]$ *be* Q *in* $M_2 \Downarrow_1 K$ so we can conclude that $(\lambda P.J)$ *of* M_1 *is* Q *in* $M_2 \Downarrow_1 K$.

If *Match*(M_1, P) fails, we proceed as in the previous case.

End of Proof.

6.2. An Eager Evaluator

The eager evaluator in SOS semantics style appears in Table 11. The function *Match*(M, P) is defined exactly as in Section 3.2 and C_e ranges over eager canonical forms.

Well-typed closed terms can always be reduced to a eager-canonical forms with the eager evaluator appearing in Table 10.

TABLE 11

Eager Evaluator in SOS Semantics Style

$$\frac{Match(C_e, P)=\sigma}{let\ C_e\ be\ P\ in\ N \Rightarrow_e N[\sigma]} \qquad \frac{M \Rightarrow_e M'}{let\ M\ be\ P\ in\ N \Rightarrow_e let\ M'\ be\ P\ in\ N}$$

$$\frac{}{[M \mid_{\mathbf{L}} N] \Rightarrow_e M} \qquad \frac{}{[M \mid_{\mathbf{R}} N] \Rightarrow_e N}$$

$$\frac{Match(C_e, P)=\sigma}{(\lambda P.J)\ of\ C_e\ is\ Q\ in\ M \Rightarrow_e let\ J[\sigma]\ be\ Q\ in\ M}$$

$$\frac{N \Rightarrow_e N'}{(\lambda P.J)\ of\ N\ is\ Q\ in\ M \Rightarrow_e (\lambda P.J)\ of\ N'\ is\ Q\ in\ M}$$

$$\frac{M \Rightarrow_e M'}{\langle M, N\rangle \Rightarrow_e \langle M', N\rangle} \qquad \frac{M \Rightarrow_e M'}{\langle N, M\rangle \Rightarrow_e \langle N, M'\rangle}$$

$$\frac{M \Rightarrow_e M'}{inl(M) \Rightarrow_e inl(M')} \qquad \frac{M \Rightarrow_e M'}{inr(M) \Rightarrow_e inr(M')}$$

We define a pattern P to specify the type A if and only if P and A correspond to one of the following cases:

- $\_$ and x satisfy any type A.
- $\sharp z$ satisfies any type $A_1 \to A_2$.
- $P_1 @ P_2$ satisfies the type A if both P_1 and P_2 satisfy the type A.
- $\langle P_1, P_2\rangle$ satisfies the type $A_1 \times A_2$ if P_1 satisfies the type A_1 and P_2 satisfies the type A_2.
- $(P_1 \mid_{\xi} P_2)$ satisfies the type $A_1 + A_2$ if P_1 satisfies the type A_1 and P_2 satisfies the type A_2.

LEMMA 6.7. *If M is a well-typed closed term of type A, P satisfies A and M is an eager-canonical form then $Match(M, P)$ is defined.*

Proof. By induction on the structure of P.

PROPOSITION 6.8. *If M well-typed and not an eager-canonical form then we have $M \Rightarrow_e N$ for some N.*

Proof. We proceed by induction on the structure of M.

- If $M = x$, then the property holds vacuously.
- $M = [M_1 \mid_{\mathbf{L}} M_2] \Rightarrow_e M_1$.
- $M = [M_1 \mid_{\mathbf{R}} M_2] \Rightarrow_e M_2$.
- $M = \langle M_1, M_2\rangle$. Then either M_1 or M_2 is not in eager-canonical form. By i.h. $M_i \Rightarrow_e L_i$ and therefore $\langle M_1, M_2\rangle \Rightarrow_e \langle L_1, M_2\rangle$ or $\langle M_1, M_2\rangle \Rightarrow_e \langle M_1, L_2\rangle$.
- $M = inl(N)$. Then N is not in eager-canonical form. By i.h. $N \Rightarrow_e L$ and $inl(N) \Rightarrow_e inl(L)$.
- $M = inr(N)$. Then N is not in eager-canonical form. By i.h. $N \Rightarrow_e L$ and $inr(N) \Rightarrow_e inr(L)$.
- $M = let\ M_1\ be\ P\ in\ M_2$. If M_1 is in eager-canonical form, then $Match(M_1, P) = \sigma$ by Lemma 6.7 and then $let\ M_1\ be\ P\ in\ M_2 \Rightarrow_e M_2[\sigma]$. Otherwise $M_1 \Rightarrow_e M_1'$ by i.h. and then $let\ M_1\ be\ P\ in\ M_2 \Rightarrow_e let\ M_1'\ be\ P\ in\ M_2$.
- $M = (\lambda P.J)\ of\ M_1\ is\ Q\ in\ M_2$. As the previous case.

6.2.1. *Adequacy*

In this section we show the equivalence between the eager evaluators in Tables 11 and 4; i.e., any result obtained via the eager evaluator in 4 can be obtained with the rules in 11 and an eager-canonical form reached by this last one can be also be reached by the first one.

PROPOSITION 6.9.

1. *If $M \Downarrow_e K$ then $M \Rightarrow_e^* K$*
2. *If $M \Rightarrow_e^* K$ and K is an eager-canonical form then $M \Downarrow_e K$.*

Proof.

1. By induction on the derivation $M \Downarrow_e K$, using Proposition 3.3.

2. By induction on the number of steps from M to K. If $M \Rightarrow_e^* K$ in 0 steps, then $M = K$ is an eager-canonical form, and by Proposition 3.3, we have $M \Downarrow_e M$. Suppose $n > 0$ is the number of steps from M to K. We proceed by induction on the structure of M.

- If $M \equiv x$, the property vacuously holds because x is not a strict canonical form.
- $M \equiv let\ M_1\ be\ P\ in\ M_2$. Since the outermost constructor $let\_be\_in\_$ has necessarily been removed (because K is an eager-canonical form, and so cannot be a $let\_be\_in\_$ constructor), the reduction sequence looks like

$$let\ M_1\ be\ P\ in\ M_2 \Rightarrow_e^* let\ M_1'\ be\ P\ in\ M_2 \Rightarrow_e M_2[\sigma] \Rightarrow_e^* K,$$

where $M_1 \Rightarrow_e^* M_1'$, M_1' is an eager-canonical form and $Match(M_1', P) = \sigma$. By the first i.h. $M_1 \Downarrow_e M_1'$ and $M_2[\sigma] \Downarrow_e K$ and so $let\ M_1\ be\ P\ in\ M_2 \Downarrow_e K$ by definition.

- $M \equiv (\lambda P.J)\ of\ M_1\ is\ Q\ in\ M_2$. As the previous case.
- $M \equiv [M_1 \mid_{\mathbf{L}} M_2]$. Then the reduction sequence looks like

$$[M_1 \mid_{\mathbf{L}} M_2] \Rightarrow_e M_1 \Rightarrow_e^* K$$

where $M_1 \Rightarrow_e^* K$. By the second i.h. $M_1 \Downarrow_e K$, which implies that $[M_1 \mid_{\mathbf{L}} M_2] \Downarrow_e K$. The case $M \equiv [M_1 \mid_{\mathbf{R}} M_2]$ is symmetrical.

- $M \equiv \langle M_1, M_2\rangle$. Then the reduction sequence looks like $\langle M_1, M_2\rangle \Rightarrow_e^* \langle K_1, K_2\rangle$, where $M_1 \Rightarrow_e^* K_1$ and $M_2 \Rightarrow_e^* K_2$. By the second i.h. $M_1 \Downarrow_e K_1$ and $M_2 \Downarrow_e K_2$ and therefore $\langle M_1, M_2\rangle \Downarrow_e \langle K_1, K_2\rangle$.

- $M \equiv inl(N)$. Then $inl(N) \Rightarrow_{e}^{*} inl(K)$, where $N \Rightarrow_{e}^{*} K$. By the second i.h. $N \Downarrow_{e} K$ and therefore $inl(N) \Downarrow_{e} inl(K)$. The same happens in the case $M \equiv inr(N)$.

3. This proof is straightforward as $\Rightarrow_{e} \subset \Rightarrow$.

7. CONCLUSIONS AND FURTHER WORK

We have presented a typed pattern calculus that offers a rational reconstruction of the pattern-matching features found in successful functional languages. The salient features of the calculus are that type-checking guarantees the absence of runtime errors such as those caused by non-exhaustive pattern-matching definitions and that its operational semantics is deterministic in a natural way, without the imposition of ad hoc solutions such as clause order or "best fit." We think it is worthwhile to go back and analyze existing language design in a new light. In particular the reader may have noticed some practical differences between the ML programs in Section 1 and the corresponding typed pattern calculus terms in Section 4.

The fact that this calculus can be designed as a computational interpretation of a well-known proof system is evidence for the depth of the insight embodied in the Curry–Howard isomorphism. It will be interesting to investigate whether this interpretation offers any new insight into the proof theory of intuitionistic logic (beyond obvious remarks as to how the disjunction property follows from our results). For example, one should study the connection with the translation from sequent calculus into natural deduction and with the cut elimination rules. (The reader has probably noticed that our operational semantics is quite different from the cut elimination rules; many of these rules do not seem to have computational significance, at least not in the spirit of current programming practice.)

The simply typed lambda calculus is the starting point of many developments in programming language design. It is natural to investigate how these developments would fare when based on the typed pattern calculus. Here are a few we think could be profitably studied in this context: ML-style type inference and polymorphism (Milner's *let*), second-order impredicative polymorphism (via second-order logics), record types (as in [19, 23]; see ML's record type patterns, as well as [14], where a language with nested extended record patterns is studied), and linear types (as in [2, 25]).

The technical aspects of our formalism could use some improvement. In particular, having to type the additional terms $[M \,|_{\mathbf{L}}\, M]$, $[M \,|_{\mathbf{R}}\, M]$, $(\lambda P{:}A.M)$ *of* M *is* $P{:}A$ *in* M is unpleasant. We are thinking about a more uniform alternative in which reduction would take place on an extended set of terms of the form $M[\mu]$ where the square brackets are object-level notation (and not meta-notation for substitution) and they may contain unfinished "matches" of the form *match* M *on* P. We would then have reduction rules such as *let* M *be* P *in* $N \Rightarrow N[$*match* M *on* $P]$, *match* $inl(M)$ *on* $(P \,|_{\xi}\, Q) \Rightarrow [\mathbf{L}/\xi]$, *match* M *on* P, and $[M \,|_{\xi}\, N][\mathbf{L}/\xi, \mu] \Rightarrow M[\mu]$. This naturally raises the issue of treating substitution as a computational process, dual to that of matching, and suggests looking for inspiration in [1].

We should also study denotational semantics for the typed pattern calculus. The space constraints do not permit us to include it, but we can give an interpretation in cartesian closed categories which generalizes that of the simply typed lambda calculus. This shows in particular that the typed pattern calculus is as expressive *extensionally* as the simply typed lambda calculus, as expected. (One should be able to show this also directly, as suggested by the translation from sequent calculi to natural deduction.) Naturally, we should look for an equational axiomatization of the typed pattern calculus which is complete for the ccc interpretation. Clearly the reduction rules in Section 5 are incomplete for this purpose. η is needed for →, and analogous for × and +. This does not seem to be enough. For example, there are at least two ways of writing *apply*: $(A \to B) \times A \to B$. One is $\lambda z.(\pi_1 z)(\pi_2 z)$, where the projections and application are just abbreviations for their translations given above. The other is $\lambda\langle \sharp z, x\rangle.z$ *of* x *is* y *in* y. Another example is $[[M_1 \,|_{\xi}\, N_1] \,|_{\zeta}\, [M_2 \,|_{\xi}\, N_2]] = [[M_1 \,|_{\zeta}\, M_2] \,|_{\xi}\, [N_1 \,|_{\zeta}\, N_2]]$. This seems to be related to *commuting conversions* [9]. The cut-elimination rules may be relevant in the search for a complete equational axiomatization.

Perhaps the most interesting suggestion for future work is the observation that we expect this calculus to be more expressive from an *intensional* point of view. Here is a partial argument. Colson shows in [5] that no *first-order*, even lazy, primitive recursion *algorithm* can compute $inf(m, n)$ in $O(inf(m, n))$ steps. Of course, *inf* is a primitive recursive function, but Colson shows that a first-order primitive recursion algorithm must use arguments sequentially, and so it will take either at least $O(m)$ or at least $O(n)$ steps. Of course lazy pattern-matching can offer such a first-order algorithm,[8] see Table 12.

However, Colson also shows that higher-order (Gödel's T) primitive recursion algorithms exist with this intensional behavior. It remains open then to find better evidence for the intuition that the typed pattern calculus is intensionally more expressive. In any case it is natural to investigate a primitive (structural) recursion generalized to deeper nested patterns, such as those used for the recursion in Colson's counterexample. One should look for a tasteful syntax that stays within well-founded recursion and prove the corresponding strong normalization result. Coquand [6], with

[8] The lazy evaluator must be slightly modified to evaluate the final results under constructors, that is, we must use $\Rightarrow_1$, cf. Section 5.

TABLE 12

***inf* in the Typed Pattern Calculus**

$$\begin{aligned}
nat &\overset{\text{def}}{=} recX.\mathbf{1}+X\\
zero &\overset{\text{def}}{=} fold_{X.\mathbf{1}+X}(inl_{nat}(\bigstar))\\
succ(M) &\overset{\text{def}}{=} fold_{X.\mathbf{1}+X}(inr_{\mathbf{1}}(M))\\
zero \mid_{\xi} succ(P) &\overset{\text{def}}{=} fold((\bigstar \mid_{\xi} P))\\
inf &\overset{\text{def}}{=} \lambda\langle zero \mid_{\xi} succ(x),\, zero \mid_{\zeta} succ(y)\rangle .\\
&\qquad\qquad [[zero \mid_{\zeta} zero] \mid_{\xi} [zero \mid_{\zeta} succ(inf\langle x, y\rangle)]]\\
&\qquad\qquad\quad : nat \times nat \rightarrow nat
\end{aligned}$$

motivating examples that include *inf* above, pursues a similar goal by adding pattern-matching constructs to Martin–Löf's logical framework.

Finally, we are interested in extending the pattern calculus to permit "constants" in patterns and also to deal with patterns for collection types which may have law-abiding constructors but which are useful in database programming [3].

APPENDIX A: THE SIMPLY TYPED LAMBDA CALCULUS

The simply typed lambda calculus as a computational interpretation of natural deduction proofs is presented by the following typing rules:

Propositions as types:

$$A ::= \langle\, propositional\ constants \rangle \mid A \wedge A \mid A \vee A \mid A \supset A$$

$$A ::= \langle\, base\ (ground)\ types \rangle \mid A \times A \mid A + A \mid A \rightarrow A$$

Sequents: $\Delta \vdash A$

Type-checking judgments: $\Delta \rhd M : A$

Proof as type-checking rules:

$$A_1, ..., A_n \vdash A_i \qquad x_1 : A_1, ..., x_n : A_n \rhd x_i : A_i \quad (proj)$$

$\{A_1, ..., A_n\}$ is a multiset but the x_j's are distinct:

$$(\wedge intro)\ \frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \wedge B} \qquad \frac{\Delta \rhd M : A \quad \Delta \rhd N : B}{\Delta \rhd \langle M, N\rangle : A \times B}\ (\times intro)$$

$$(\wedge elim 1)\ \frac{\Delta \vdash A \wedge B}{\Delta \vdash A} \qquad \frac{\Delta \rhd M : A \times B}{\Delta \rhd \pi_1(M) : A}\ (\times elim 1)$$

$$(\wedge elim 2)\ \frac{\Delta \vdash A \wedge B}{\Delta \vdash B} \qquad \frac{\Delta \rhd M : A \times B}{\Delta \rhd \pi_2(M) : B}\ (\times elim 2)$$

$$(\vee intro 1)\ \frac{\Delta \vdash A}{\Delta \vdash A \vee B} \qquad \frac{\Delta \rhd M : A}{\Delta \rhd inl_B(M) : A + B}\ (+ intro 1)$$

$$(\vee intro 2)\ \frac{\Delta \vdash B}{\Delta \vdash A \vee B} \qquad \frac{\Delta \rhd N : B}{\Delta \rhd inr_A(N) : A + B}\ (+ intro 2)$$

$$(\vee elim)\ \frac{\Delta \vdash A \vee B \quad A, \Delta \vdash C \quad B, \Delta \vdash C}{\Delta \vdash C}$$

$$\frac{\Delta \rhd L : A + B \quad x : A, \Delta \rhd M : C \quad y : B, \Delta \rhd N : C}{\Delta \rhd case\ L\ of\ x : A . M \mid y : B . N : C}\ (+ elim)$$

$$(\supset intro)\ \frac{A, \Delta \vdash B}{\Delta \vdash A \supset B} \qquad \frac{x : A, \Delta \rhd M : B}{\Delta \rhd \lambda x : A . M : A \rightarrow B}\ (\rightarrow intro)$$

$$(\supset elim)\ \frac{\Delta \vdash A \supset B \quad \Delta \vdash A}{\Delta \vdash B} \qquad \frac{\Delta \rhd M : A \rightarrow B \quad \Delta \rhd N : A}{\Delta \rhd MN : B}\ (\rightarrow elim)$$

Lazy Evaluator in Natural Semantics Style.

$$\frac{L \Downarrow_1 \langle M, N\rangle \quad M \Downarrow_1 K}{\pi_1(L) \Downarrow_1 K} \qquad \frac{L \Downarrow_1 \langle M, N\rangle \quad N \Downarrow_1 K}{\pi_2(L) \Downarrow_1 K}$$

$$\frac{L \Downarrow_1 inl(I) \quad M[I/x] \Downarrow_1 K}{case\ L\ of\ x.M \mid y.N \Downarrow_1 K} \qquad \frac{L \Downarrow_1 inr(J) \quad N[J/y] \Downarrow_1 K}{case\ L\ of\ x.M \mid y.N \Downarrow_1 K}$$

$$\frac{M \Downarrow_1 \lambda x.L \quad L[N/x] \Downarrow_1 K}{MN \Downarrow_1 K}$$

$$\frac{}{\langle M, N\rangle \Downarrow_1 \langle M, N\rangle} \qquad \frac{}{\lambda x.M \Downarrow_1 \lambda x.M} \qquad \frac{}{inl(M) \Downarrow_1 inl(M)} \qquad \frac{}{inr(N) \Downarrow_1 inr(N)}$$

Eager Evaluator in Natural Semantics Style.

$$\frac{L \Downarrow_{\mathrm{e}} \langle K_1, K_2\rangle}{\pi_1(L) \Downarrow_{\mathrm{e}} K_1} \qquad \frac{L \Downarrow_{\mathrm{e}} \langle K_1, K_2\rangle}{\pi_2(L) \Downarrow_{\mathrm{e}} K_2}$$

$$\frac{L \Downarrow_{\mathrm{e}} inl(I) \quad M[I/x] \Downarrow_{\mathrm{e}} K}{case\ L\ of\ x.M \mid y.N \Downarrow_{\mathrm{e}} K} \qquad \frac{L \Downarrow_{\mathrm{e}} inr(J) \quad N[J/y] \Downarrow_{\mathrm{e}} K}{case\ L\ of\ x.M \mid y.N \Downarrow_{\mathrm{e}} K}$$

$$\frac{M \Downarrow_{\mathrm{e}} \lambda x.J \quad N \Downarrow_{\mathrm{e}} K \quad J[K/x] \Downarrow_{\mathrm{e}} L}{MN \Downarrow_{\mathrm{e}} L}$$

$$\frac{M \Downarrow_{\mathrm{e}} K \quad N \Downarrow_{\mathrm{e}} L}{\langle M, N\rangle \Downarrow_{\mathrm{e}} \langle K, L\rangle} \qquad \frac{M \Downarrow_{\mathrm{e}} K}{inl(M) \Downarrow_{\mathrm{e}} inl(K)} \qquad \frac{N \Downarrow_{\mathrm{e}} L}{inr(N) \Downarrow_{\mathrm{e}} inr(L)} \qquad \frac{}{\lambda x.M \Downarrow_{\mathrm{e}} \lambda x.M}$$

ACKNOWLEDGMENTS

We are grateful to Peter Buneman, Roberto Di Cosmo, Emmanuel Engel, Jean Gallier, Brian Howard, and Dan Suciu for helpful discussions.

Received July 6, 1994; final manuscript received July 26, 1995

REFERENCES

1. Abadi, M., Cardelli, L., Curien, P.-L., and Lévy, J.-J. (1990). "Explicit Substitutions," Technical Report 54, DEC Systems Research Center.
2. Abramsky, S. (1993). Computational interpretations of linear logic, *Theoret. Comput. Sci.* **111**:3–57.
3. Breazu-Tannen, V., and Subrahmanyam (1991). Logical and computational aspects of programming with sets/bags/lists, *in* "Proceedings of the 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991," pp. 60–75, Lecture Notes in Computer Science, Vol. 510, Springer-Verlag, Berlin/New York.
4. Burstall, R. M., MacQueen, D. B., and Sanella, D. T. (1980). Hope: An experimental applicative language, *in* "LISP Conference," pp. 136–143, Stanford University Computer Science Department.

5. Colson, T. (1989). About primitive recursive algorithms, *in* "Proceedings of the 16th International Colloquium on Automata, Languages, and Programming, Stresa, July 1989," pp. 194–206, Lecture Notes in Computer Science, Vol. 372, Springer-Verlag, Berlin/New York.
6. Coquand, T. (1992). Pattern matching with dependent types, *in* "Proceedings of the Workshop on Types for Proofs and Programs, Baastad, June 1992," pp. 66–79; available as pub/baastad/proc.ps.Z on animal.cs.chalmers.se, by anonymous ftp.
7. Field, A., and Harrison, P. (1988). "Functional Programming," Addison–Wesley, Reading, MA.
8. Gallier, J. (1993). Constructive logics. I. A tutorial on proof systems and typed λ-calculi, *Theoret. Comput. Sci.* **110**, 249–339.
9. Girard, J. Y., Lafont, Y., and Taylor, P. (1989). "Proofs and Types," Cambridge Univ. Press, London/New York.
10. Howard, B. T. (1992). "Fixed Points and Extensionality in Typed Functional Programming Languages," Ph.D. thesis, Stanford University, August.
11. Hudak, P., Peyton-Jones, S, Wadler, P., *et al.* (1992). Report on the programming language Haskell, a non-strict purely functional language. Version 1.2, *Sigplan Not.* **27**.
12. Huet, G., and Lévy, J.-J. (1991). Computations in orthogonal rewriting systems. *In* "Computational Logic, Essays in Honor of Alan Robinson" (Jean-Louis Lassez and Gordon Plotkin, Eds.), MIT Press, Cambridge, MA.
13. Kahn, G. (1987). *In* "Proceedings Symposium Theoretical Aspects of Computer Science," pp. 22–39, Lecture Notes in Computer Science, Vol. 247, Springer-Verlag, Berlin/New York.
14. Jategaonkar, L. A., and Mitchell, J. C. (1988). Ml with extended pattern matching and subtypes, *in* "Proceedings of the LISP and Functional Programming Conference," pp. 198–211, Assoc. Comput. Mach., New York.
15. Peyton Jones, S. (1987). "The Implementation of Functional Programming Languages," Prentice–Hall, New York.
16. Lafont, Y. (1989). "Functional Programming and Linear Logic," Lecture notes for the Summer School on Functional Programming and Constructive Logic, Glasgow, September.
17. Leroy, X. (1994). The Caml Light system, Release 0.7—Documentation and user's manual, INRIA.
18. Milner, R., Tofte, M., and Harper, R. (1990). "The Definition of Standard ML," MIT Press, Cambridge, MA.
19. Ohori, A., and Buneman, O. P. (1988). Type inference in a database programming language, *in* "Proceedings ACM Conference on LISP and Functional Programming," pp. 174–183, Snowbird, UT, July.
20. van Oostrom, V. (1990). "Lambda calculus with Patterns," IR 228, Vrije Universiteit, Amsterdam, November.
21. Plotkin, G. (1981). "A Structural Approach to Operational Semantics," Technical Report DAIMI FN-19, Aarhus University.
22. Puel, L., and Suárez, A. (1993). Compiling pattern matching by term decomposition, *J. Symbolic Comput.* **15**, No. 1.
23. Rémy, D. (1989). Typechecking records and variants in a natural extension of ml, *in* "ACM Conference on Principles of Programming Languages" (David MacQueen, Ed.).
24. Turner, D. A. (1985). Miranda: A non-strict functional language with polymorphic types, *in* "Proceedings of the Conference on Functional Programming Languages and Computer Architecture" (J.-P. Jouannaud, Ed.), pp. 1–16, Lecture Notes in Computer Science, Vol. 201, Springer-Verlag, Berlin/New York.
26. Wadler, P. (1990). Linear types can change the world! *In* "Programming Concepts and Methods," North-Holland.