A General Technique for Analyzing Termination in Symmetric Proof Calculi

Daniel J. Dougherty[1], Silvia Ghilezan[2] and Pierre Lescanne[3]

[1] Worcester Polytechnic Institute, USA, dd@cs.wpi.edu
[2] Faculty of Engineering, University of Novi Sad, Serbia, gsilvia@uns.ns.ac.yu
[3] École Normale Supérieure de Lyon, France, Pierre.Lescanne@ens-lyon.fr

Abstract. Proof-term calculi expressing a computational interpretation of classical logic serve as tools for extracting the constructive content of classical proofs and at the same time can be seen as pure programming languages with explicit representation of control. The inherent symmetry in the underlying logic presents technical challenges to the study of the reduction properties of these systems. We explore the use of intersection types for these calculi, note some of the challenges inherent in applying intersection types in a symmetric calculus, and show how to overcome these difficulties. The approach is applicable to a variety of systems under active investigation in the current literature; we apply our techniques in a specific case study: characterizing termination in the symmetric lambda-calculus of Barbanera and Berardi.

1 Introduction

The Curry-Howard correspondence [10] expresses a fundamental connection between logic and computation. In its traditional form, terms in the 'l-calculus encode proofs in intuitionistic natural deduction and proofs serve as typing derivations for the terms. Griffin extended the Curry-Howard correspondence to classical logic in his seminal 1990 POPL paper [8], by observing that classical tautologies suggest typings for certain control operators. This initiated a vigorous line of research: on the one hand classical calculi can be seen as pure programming languages with explicit representations of control, while at the same time terms can be tools for extracting the constructive content of classical proofs [13, 2]. In particular the $\overline{\lambda}\mu$ calculus of Parigot [15] has been the foundation of a number of investigations [16, 6, 14, 3, 1] into the relationship between classical logic and theories of control in programming languages.

In contrast to natural deduction proof systems (upon which Parigot's $\overline{\lambda}\mu$, for example, is based) sequent calculi exhibit inherent symmetries in proof structures. There are several term calculi based on sequent calculus, in which reduction corresponds to cut elimination. Examples include [9, 19, 4, 20, 12]. This symmetry is appealing in its way, but it actually creates considerable technical difficulty in analyzing the reduction behavior of these calculi. The bedrock traditional technique of reducibility makes essential use of fact that function types are "higher" in a natural sense than argument types, permitting semantic definitions to proceed by induction on types. The symmetry in classical calculi blocks a straightforward adaptation of the traditional reducibility technique.

The Symmetric Lambda Calculus (here denoted λ^{sym}) of Barbanera and Berardi [2] is an elegant calculus designed with the goal of extracting constructive content from classical proofs (actually what we call λ^{sym} here is the propositional version of their calculus, in [2] the system is extended to Peano arithmetic). Barbanera and Berardi proved termination for their calculus using a fundamental insight called the "symmetric candidates" technique. (A completely different approach is the arithmetical proof of termination in a symmetric $\overline{\lambda}\mu$ calculus by David and Nour [5].)

The use of symmetric candidates is a robust technique that applies in a variety of settings [19, 17]. But a fundamental tool for deep analysis of the reduction properties of λ-calculus, as well as semantic investigations, is the extension of simple typing to intersection types — and *the adaptation of the symmetric candidates technique to the intersection-types setting is not straightforward.* Such an adaptation is the topic of this paper. We analyze the problems that arise and show how to overcome them. Our technique applies generally to all of the symmetric proof-calculi we have investigated, including the $\overline{\lambda}\mu\widetilde{\mu}$-calculus of Curien and Herbelin [4, 7], and the dual calculus of Wadler [20]. For concreteness here we outline the treatment for λ^{sym}, chosen because it is syntactically most familiar and the issues with duality can be seen most clearly. We believe that the presentation here may also clarify some of the subtleties of Barbanera and Berardi's proof even for simply-typed λ^{sym}.

The technical contribution of this paper can be summarized as follows. The key to the symmetric candidates technique is to interpret types in certain families of *saturated* sets: these are sets which are, roughly speaking, closed under inverse β-reduction. Using a clever fixed-point computation Barbanera and Berardi [2] define a family of saturated sets appropriate for interpreting simply-typed terms. But when we wish to consider intersection types we face the following obstacle. In standard semantics of intersection types, the interpretation of an intersection type $(A \cap B)$ is simply the intersection of the interpretations of A and B. But in general *intersections of saturated sets are not saturated.* So it is not clear how to interpret a type of the form $(A \cap B)$.

The solution we adopt is the following. We take seriously the fact that saturated sets are fixed points of a monotonic operator. And as is well-known, a monotonic operator has *a complete lattice* of fixed points. So we take the interpretation $[\![(A \cap B)]\!]$ of an intersection type to be the meet $[\![A]\!] \curlywedge [\![B]\!]$ in this lattice.

A consequence of the fact that the interpretation of an intersection is not the intersection of interpretations is that the standard typing rule for intersection-introduction is not sound. So our type system has an intersection-elimination rule only. As it turns out this is not a problem since intersection-introduction is not needed for characterizing termination.

A natural question arises: in the absence of intersection-introduction, how do any terms ever receive a type which is an intersection? The answer is: by double-negation elimination! Referring to the type system of Definition 2, if x is declared with type $(A \cap B)^{\perp}$ in a typing of $c : \perp$, then $\lambda x.c$ will be typed with type $(A \cap B)$.

2 Intersection types for the λ^{sym}-calculus

The syntax of λ^{sym} expressions is given by the following:

$$t := x \mid \langle t_1, t_2 \rangle \mid \sigma_1(t), \mid \sigma_2(t), \mid \lambda x.c \mid (t_1 * t_2)$$

We depart from [2] in that we treat the operator $*$ is syntactically commutative, that is, we consider $(t_1 * t_2)$ and $(t_2 * t_1)$ to be the same term.

The reduction rules of the calculus are

$$(\lambda x.b * a) \longrightarrow b[x \leftarrow a] \qquad (\langle t_1, t_2 \rangle * \sigma_i(u)) \longrightarrow \langle t_i, u \rangle \qquad \lambda x.(b * x) \longrightarrow b \text{ if } x \text{ not free in } b$$

The system of [2] had an "η-reduction" rule as well as a reduction (labeled *Triv* there) which allowed replacing a term by one of its closed subterms under certain circumstances. We have omitted these reductions here for convenience (only): adding it would not violate any of our results and would complicate the presentation.

Definition 1. *The set* $\mathbb{T}$ *of* raw types *is generated from an infinite set TVar of type-variables as follows.*

$$\mathbb{T} ::= TVar \mid \mathbb{T} \wedge \mathbb{T} \mid \mathbb{T} \vee \mathbb{T} \mid \mathbb{T}^{\perp} \mid \mathbb{T} \cap \mathbb{T}$$

We consider raw types modulo the equations

$$A^{\perp\perp} = A \qquad (A \wedge B)^{\perp} = A^{\perp} \vee B^{\perp} \qquad (A \vee B)^{\perp} = A^{\perp} \wedge B^{\perp}$$

A *type* is either an equivalence class modulo these equations or the special type $\perp$. Note that by orienting the equations above left-to-right each type has a normal form, in which the $(\cdot)^{\perp}$ operator is applied only to type variables or intersections. It is then easy to see that each type other than $\perp$ is uniquely of one of the following forms (here τ is a type variable).

$$\tau \quad \tau^{\perp} \quad (A_1 \wedge \cdots \wedge A_n) \quad (A_1 \vee \cdots \vee A_n) \quad (A_1 \cap \cdots \cap A_n) \quad (A_1 \cap \cdots \cap A_n)^{\perp}$$

Definition 2 (Typing rules of the system $\mathcal{B}$). *The type assignment system $\mathcal{B}$ is given by the following typing rules.*

$$\frac{}{\Sigma,\ x:(T_1\cap\cdots\cap T_k)\ \vdash\ x:T_i}\ (ax)$$

$$\frac{\Sigma\ \vdash\ t_1:A_1 \qquad \Sigma\ \vdash\ t_2:A_2}{\Sigma\ \vdash\ \langle t_1,t_2\rangle:A_1\wedge A_2}\ (\wedge) \qquad \frac{\Sigma\ \vdash\ t:A_i}{\Sigma\ \vdash\ \sigma_i(t):A_1\vee A_2}\ (\vee)$$

$$\frac{\Sigma,\ x:A\ \vdash\ c:\bot}{\Sigma\ \vdash\ \lambda x.c:A^{\bot}}\ (\bot) \qquad \frac{\Sigma\ \vdash\ p:A \qquad \Sigma\ \vdash\ q:A^{\bot}}{\Sigma\ \vdash\ (p\ *\ q):\bot}\ (cut)$$

Theorem 3 (Main result). *A λ^{sym} term is terminating if and only if it is typable in $\mathcal{B}$.*

The direction "every terminating term is typable" follows the standard pattern from traditional λ-calculus. The nuance to be acknowledged is that, for the purpose of establishing this result, the standard intersection-introduction typing rule is not needed. This is crucial for our approach since, as noted in the introduction, this rule seems to be an obstacle to the other direction of the result, "every typable term is terminating."

The remainder of the paper is an outline of the proof that every typable term is terminating, with an emphasis on those aspects of the argument requiring novel treatment beyond the proof for traditional λ-calculus.

Definition 4. *A* pair *is given by two sets of terms X and Y, each of which is non-empty.*
The pair $\{X_1,X_2\}$ is stable *if for every $r\in X_1$ and every $e\in E$, the command $(r\ *\ e)$ is terminating.*

For example, the pair $\{Vars,Vars\}$ is stable. Since the sets in a pair are non-empty, any stable pair consists of terminating expressions. We write pairs as $\{X,Y\}$ but we stress that this is a multiset, that is, it may be that X and Y are the same set.

The following technical condition will be crucial to the use of pairs to interpret types (it is this condition which makes the Type Soundness Theorem go through, specifically the cases of typing a λ expression).

Definition 5. *A pair $\{X_0,X_1\}$ is* saturated *if for each i, whenever $\lambda x.c$ satisfies: $\forall e\in X_i, c[x\leftarrow e]$ is terminating, then $\lambda x.c\in X_{1-i}$.*

We can always expand a pair to be saturated. It is more delicate to expand a stable pair to be saturated and remain stable. The development below achieves this.

Definition 6. *An expression is* simple *if it is not a 'l-abstraction; a set X is simple if each term in X is simple.*

Definition 7. *Define the map $\Phi:2^{\Lambda}\to 2^{\Lambda}$ by*

$$\begin{aligned}\Phi(X) = &\{e\mid e \textit{ is of the form } \lambda x.c \textit{ and } \forall r\in X, c[x\leftarrow r] \textit{ is terminating}\}\\ &\cup\{e\mid e \textit{ is simple and } \forall r\in X,(r\ *\ e) \textit{ is terminating}\}\end{aligned}$$

Note that if $X\neq\emptyset$ then $\Phi(X)$ is a set of terminating terms. Also, if $X\subseteq SN$ then all variables are in $\Phi(X)$.

It is easy to see that Φ is antimonotone. So the map $(\Phi\circ\Phi)=\Phi^2$ is monotone. By the Knaster-Tarski fixed point theorem [11, 18] Φ^2 has a complete lattice of fixed points, ordered by set inclusion.

Definition 8. *A pair $\{X,Y\}$ is a* mutual fixed point *for Φ if $\Phi(X)=Y$ and $\Phi(Y)=X$.*

It is easy to see that if $\{X,Y\}$ is a mutual fixed point for Φ then X is a fixed point for Φ^2.

Lemma 9. *Suppose $\{X,Y\}$ is a mutual fixed point for Φ. Then $\{X,Y\}$ is stable and saturated.*

So now our task is to show how to make mutual fixed points which have the right structure for interpreting types. The strategy for defining saturated pairs for types is slightly different depending on whether the type to be interpreted is (i) an arrow-type or its dual or (ii) an intersection or its dual. In the former case we need to establish the that the operator Φ^2 is inflationary on strongly normalizing simple sets.

Lemma 10. *If X is a simple set of terminating terms then $X \subseteq \Phi^2(X)$.*

As is well-known, when $\mathcal{G}$ is a monotone operator on a complete lattice of sets and X satisfies $X \subseteq \mathcal{G}(X)$ then $\mathcal{G}$ has a fixed point containing X. This, in light of Lemma 10, justifies the following definition.

Definition 11. *If X is a simple set of terminating terms let $X^{\uparrow}$ be the least fixed point of Φ^2 with the property that $X \subseteq X^{\uparrow}$.*

We note the following facts. If X is a simple set of terminating terms then $\{X^{\uparrow}, \Phi(X^{\uparrow})\}$ is a mutual fixed point of Φ, with $X \subseteq X^{\uparrow}$. If $\{X,Y\}$ is a stable pair and Y is a set of simple terms then $Y \subseteq \Phi(X^{\uparrow})$.

In Definition 13 we will use the above construction to interpret types which are not intersections (or their duals). When the types we want to interpret are intersections, or types of the form $(T_1 \cap \cdots \cap T_k)^{\circ}$ the above construction does not work. The essential problem is that the intersection of saturated pairs does not in general yield a saturated pair. This means that the interpretation of an intersection type $(A \cap B)$ will not be the intersection of the interpretations of A and B. But the collection of fixed points of Φ^2 carries its own lattice structure under inclusion, and this is all we require to interpret intersection types.

Definition 12. *Let Fix_{Φ^2} be the set of fixed points of the operator Φ^2. If $R_1, \ldots, R_k$ are fixed points of Φ^2, let $(R_1 \curlywedge \ldots \curlywedge R_k)$ denote the meet of these elements in the lattice Fix_{Φ^2}.*

The set of objects of the lattice Fix_{Φ^2} is a subset of the set 2^{Λ}. Since this lattice is ordered by set inclusion, we have $(R_1 \curlywedge \ldots \curlywedge R_k) \subseteq R_i$ for each i. Since $\cap$ is the greatest lower bound operator in 2^{Λ}, $(R_1 \curlywedge \ldots \curlywedge R_k) \subseteq (R_1 \cap \cdots \cap R_k)$.

We stress that $(R_1 \curlywedge \ldots \curlywedge R_k)$ is a fixed point of Φ^2 and so the pair $\{(R_1 \curlywedge \ldots \curlywedge R_k),\ \Phi(R_1 \curlywedge \ldots \curlywedge R_k)\}$is a mutual fixed point of Φ.

Definition 13 (Interpretation of types). *For each type T we define the set $[\![T]\!]$.*

1. *When T is $\perp$ then $[\![T]\!]$ is the set of terminating terms.*
2. *When T is a type variable we set R to be the set of term variables, then construct the pair $(R^{\uparrow}, \Phi(R^{\uparrow})$. We then take $[\![T]\!]$ to be $R^{\uparrow}$ and $[\![T^{\perp}]\!]$ to be $\Phi(R^{\uparrow})$.*
3. *Suppose T is $(A_1 \wedge A_2)$. Set R to be $\{\langle t_1, t_2\rangle \mid t_i \in [\![A_i]\!],\ i = 1,2\}$. We then take $[\![T]\!]$ to be $(R^{\uparrow})$ and $[\![T^{\perp}]\!] = [\![{A_1}^{\perp} \vee {A_2}^{\perp}]\!]$ to be $\Phi(R^{\uparrow})$.*
4. *When T is $(A_1 \cap A_2 \cdots \cap A_n)$, $\quad n \geq 2$, we take $[\![T]\!]$ to be $([\![A_1]\!] \curlywedge \ldots \curlywedge [\![A_n]\!])$ and then take $[\![T^{\perp}]\!]$ to be $\Phi([\![T]\!])$.*

Note that the interpretation $[\![A_1 \vee A_2]\!]$ of a disjunction-type is determined in part 3 above since any type $B_1 \vee B_2$ is the dual of ${B_1}^{\perp} \wedge {B_2}^{\perp}$. Also note that by definition, for each type T the pair $([\![T]\!], [\![T^{\perp}]\!])$ is a mutual fixed point of Φ and Φ and so constitutes a stable saturated pair.

Lemma 14.

1. *$[\![T]\!]$ is a set of terminating terms.*
2. *$[\![A_1 \wedge A_2]\!] \supseteq \{\langle t_1, t_2\rangle \mid t_i \in [\![A_i]\!],\ i = 1,2\}$.*
3. *$[\![A_1 \vee A_2]\!] \supseteq \{\sigma_1(p) \mid p \in [\![A_1]\!]\} \cup \{\sigma_2(p) \mid p \in [\![A_2]\!]\}$.*
4. *$(\lambda x.c) \in [\![A]\!]$ if for all $e \in [\![A^{\perp}]\!]$ we have $c[x \leftarrow e]$ terminates.*
5. *$[\![(A_1 \cap \cdots \cap A_k)]\!] \subseteq ([\![A_1]\!] \cap \cdots \cap [\![A_k]\!])$.*

Since each $[\![T]\!]$ consists of terminating expressions the following theorem implies that all typable expressions are terminating.

Theorem 15 (Type Soundness). *If expression t is typable with type T then $t \in [\![T]\!]$.*

The proof is by induction over typing derivations. All the hard work has been done in Lemma 14; the proof is organized by cases according to the last inference in the derivation, and each case follows readily from the clauses of the Lemma.

References

1. Z. M. Ariola and H. Herbelin. Minimal classical logic and control operators. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, volume 2719 of *LNCS*, pages 871–885. Springer-Verlag, 2003.
2. F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Information and Computation*, 125(2):103–117, 1996.
3. G. M. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In *Proc. of Symp. on Mathematical Foundations of Computer Science.*, volume 1450 of *LNCS*, pages 336–345. Springer-Verlag, 1998.
4. P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. of the 5th ACM SIGPLAN Int. Conference on Functional Programming, ICFP'00*, Montreal, Canada, 2000. ACM Press.
5. R. David and K. Nour. Arithmetical proofs of strong normalization results for the symmetric $\overline{\lambda}\mu$-calculus. In *Typed Lambda Calculus and application, TLCA 2005*, volume 3461 of *LNCS*, pages 162–178. Springer-Verlag, 2005.
6. Ph. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *Logic Programming and Artificial Reasoning, LPAR'94*, volume 822 of *LNCS*, pages 31–43. Springer-Verlag, 1994.
7. D. Dougherty, S. Ghilezan, and P. Lescanne. Characterizing strong normalization in the Curien-Herbelin symmetric lambda calculus: extending the Coppo-Dezani heritage. In S. Berardi and U de' Liquoro, editors, *Theoretical Computer Science*, volume Festschrift Coppo, Dezani, Ronchi. To appear.
8. T. Griffin. A formulae-as-types notion of control. In *Proc. of the 19th Annual ACM Symp. on Principles Of Programming Languages, (POPL'90)*, pages 47–58, San Fransisco (Ca., USA), 1990. ACM Press.
9. H. Herbelin. *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes*. Thèse, U. Paris 7, Janvier 1995.
10. W. A. Howard. The formulas-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490, New York, 1980. Academic Press.
11. B. Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Societé Polonaise de Mathematique*, 6:133–134, 1928.
12. S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In Bernhard Gramlich and Salvador Lucas, editors, *ENTCS*, volume 86. Elsevier, 2003.
13. Ch. R. Murthy. Classical proofs as programs: How, what, and why. In J. Paul Myers Jr. and Michael J. O'Donnell, editors, *Constructivity in Computer Science*, volume 613 of *LNCS*, pages 71–88. Springer, 1991.
14. C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *POPL 24*, pages 215–227, 1997.
15. M. Parigot. An algorithmic interpretation of classical natural deduction. In *Proc. of Int. Conf. on Logic Programming and Automated Reasoning, LPAR'92*, volume 624 of *LNCS*, pages 190–201. Springer-Verlag, 1992.
16. M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *The J. of Symbolic Logic*, 62(4):1461–1479, December 1997.
17. E. Polonovski. Strong normalization of $\lambda\mu\tilde{\mu}$-calculus with explicit substitutions. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004*, volume 2987 of *LNCS*, pages 423–437. Springer-Verlag, 2004.
18. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
19. C. Urban and G. M. Bierman. Strong normalisation of cut-elimination in classical logic. In *Typed Lambda Calculus and Applications, TLCA'99*, volume 1581 of *LNCS*, pages 365–380. Springer-Verlag, 1999.
20. Ph. Wadler. Call-by-value is dual to call-by-name. In *Proc. of the 8th ACM SIGPLAN Int. Conference on Functional Programming, ICFP'03*, pages 189–201, 2003.