[20] W.W. Tait. A realizability interpretation of the theory of species. In *Proceedings of Logic Colloquium, Lecture Notes in Mathematics, Vol. 435*, pages 240–251, 1975.

[21] S. Thatte. Type inference with partial types. In *Proc. Int'l Coll. on Automata, Languages and Programming (ICALP)*, pages 615–629, 1988. Lecture Notes in Computer Science, vol 317.

[22] S. van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102:135–163, 1992.

[23] S. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems.* PhD thesis, Katholieke Universiteit Nijmegen, 1993.

[24] Mithell Wand, Patrick O'Keefe, and Jens Palsberg. Strong normalization with non-structural subtyping. *Mathematical Structures in Computer Science*, 5(3):419–430, 1995.

[6] G. Ghelli. Termination of system F-bounded. Technical report, University of Pisa, August 1994. Available by anonymous ftp at `ftp://ftp.di.unipi.it/pub/Papers/ghelli/F-bounded.Term.ps` .

[7] J. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[8] J.M.E. Hyland and C.-H.L. Ong. Modified realizability topos and strong normalization proofs. In *Proc. Conf. Typed Lambda Calculus and Applications*, pages 179–194. Springer, 1993. Lecture Notes in Computer Science, Vol. 664.

[9] D. Kozen, J. Palsberg, and M.I. Schwartzbach. Efficient inference of partial types. *Journal of Computer and System Sciences*, 49(2):306–324, 1994.

[10] J.L. Krivine. *Lambda-calculus, Types and Models.* Ellis Horwood series in computers and their applications. Masson and Ellis Horwood, translation from French 1990 edition, 1993.

[11] D. Leivant. Polymorphic type inference. In *Proc. 10th ACM Symp. on Principles of Programming Languages*, pages 88–98. ACM, January 1983.

[12] D. McAllester, J. Kučan, and D.F. Otth. A proof of strong normalization for F_2, F_ω, and beyond. *Information and Computation*, 121(2):193–200, September 1995.

[13] J. Mitchell. Polymorphic type inference and containment. *Information and Control*, 76:211–249, 1988.

[14] J. Mitchell. Type inference with simple subtypes. *Journal of Functional Programming*, 1(3):245–285, July 1991.

[15] J.C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions. In *ACM Conference on LISP and Functional Programming*, pages 309–319. ACM, August 1986.

[16] C.-H.L. Ong and E. Ritter. A generic strong normalization argument: Application to the calculus of constructions. In *Proc. Computer Science Logic Workshop*, pages 261–279. Springer, 1993. Lecture Notes in Computer Science, Vol. 832.

[17] J. Palsberg, M. Wand, and P. O'Keefe. Type inference with non-structural subtyping. BRICS Technical Report RC-95-33, Department of Computer Science, University of Aarhus, April 1995.

[18] Jens Palsberg. Normal forms have partial types. *Information Processing Letters*, 45:1–3, 1993.

[19] W.W. Tait. Intensional interpretation of functionals of finite type. *Journal of Symbolic Logic*, 32:198–212, 1967.

union types and existential types as well as intersection types; a similar observation is in [12] where, rather than interpreting types as sets of terms in SN, types are interpreted as sets of values (with respect to a specified evaluation relation) of terms in SN. In [12] it is shown how this approach yields clean normalization proofs for many powerful extensions of system F_2, including several extensions with subtyping. Important examples are system F-bounded and Ghelli's system $F_{\leq}$; system $F_{\leq}$ was first shown to be strongly normalizing in Ghelli's thesis [5], which uses saturated sets. Termination of system F-bounded is also proven in [6] using similar methods. Recent generalizations of techniques for proving strong normalization can be found in [8], [16], [3].

Comparing the typing power of the non-structural subtyping system to other systems, it is noted in [24] that system F_2 and the system $\lambda(\rightarrow, \bot, \top, B)$ are incomparable with respect to typing power, since they both type some terms that have no type in the other system; this is observed to hold also for F_2 in comparison with Thatte's partial typing. The system $\lambda(\rightarrow, \bot, \top, B)$ is strictly more powerful than partial typing [24]. It is also worth noting that a bottom type (a subtype of every type) is not definable in system $F_{\leq}$, since, as is shown in [5], there are types with no common lower bound in system $F_{\leq}$. Palsberg [18] shows that every lambda term in normal form has a partial type in Thatte's system.

Type inference algorithms for partial types and non-structural subtyping can be found in [9], [17].

Acknowledgement

Thanks to an anonymous referee for helpful comments.

References

[1] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[2] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 117–309. Oxford University Press, 1992.

[3] J. H. Gallier. Proving properties of typed lambda terms using relaizability, covers, and sheaves. Technical Report IRCS95-24, University of Pennsylvania, September 1995.

[4] J.H. Gallier. On Girard's "Candidats de Reducibilité". In P. Odifreddi, editor, *Logic and Computer Science*, pages 123–203. Academic Press, 1990.

[5] G. Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, Universita di Pisa, Dipartimento di Informatica, March 1990.

□

The last part of our proof is to extend the proof of Proposition 2.4 with one new case of the induction, covering the subtyping rule via Lemma 3.2.

Proposition 3.3 *(Soundness of non-structural subtyping)*

$$\Gamma \vdash M : \sigma \Rightarrow \Gamma \models M : \sigma$$

PROOF The proof is by induction on the derivation of $\Gamma \vdash M : \sigma$. We consider only the new case where we use the subtyping rule

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \leq \tau}{\Gamma \vdash M : \tau}$$

By induction hypothesis we have

$$\Gamma \models M : \sigma$$

Now suppose that $\rho \models \Gamma$. We must show that $\rho \models M : \tau$. But, since $\Gamma \models M : \sigma$, we have $\rho \models M : \sigma$, *i.e.*, $[\![M]\!]\rho \in [\![\sigma]\!]$. By Lemma 3.2 we have $[\![\sigma]\!] \subseteq [\![\tau]\!]$, hence $[\![M]\!]\rho \in [\![\tau]\!]$, which shows that $\rho \models M : \tau$, as desired. □

Finally, we note that the extended version of Theorem 2.5 goes through as usual, using Proposition 3.3 in the standard way. This completes the proof of strong normalization for non-structural subtyping.

4 Other systems

It is not difficult to see that appropriate analogues of Lemma 3.2 will hold for many other subtyping systems than the one considered here. So long as the ordering on types respects the set-theoretic interpretation in saturated subsets of SN, the method appears to be robust. One example is the system of *intersection types* (the variant called $\lambda\cap^-$ in [2]) which, remarkably, can be shown to type exactly the set of strongly normalizing lambda terms, see [2], [22], [23]. For this system we may interpret intersection type expressions $\tau \sqcap \sigma$ by $[\![\tau \sqcap \sigma]\!] = [\![\tau]\!] \cap [\![\sigma]\!]$ and still have appropriate extensions of Lemma 2.3 and Lemma 3.2 going through. Strong normalization for this system can also be proved by reduction to strong normalization for the simply typed lambda calculus extended with pairing and projection, as shown in [11]. This method is much like a coercion interpretation. However, it appears that the method of saturated sets provides a single framework within which a very considerable number of strongly normalizing systems can be proven to be so, and in many cases the normalization result for an extension of a system for which the method has already provided a normalization proof will be obvious from the very structure of that proof.

More generally, it is immediate from Definition 2.2 that SAT is closed under arbitrary unions of elements of SAT, hence SAT is a complete lattice under set inclusion. This makes it an easy matter to extend the proof method to deal with subtyping systems with

PROOF The proof is by induction on the proof of $\sigma \leq \tau$.

1. *Case*
$$\sigma \equiv \bot \leq \tau$$
Then $[\![\sigma]\!] = \bigcap_{S \in SAT} S \subseteq [\![\tau]\!]$, since $[\![\tau]\!] \in SAT$, for arbitrary τ.

2. *Case*
$$\sigma \equiv b \leq b' \equiv \tau'$$
Here $[\![b]\!] = [\![b']\!] = SN$, and the inclusion holds.

3. *Case*
$$\sigma \leq \tau \equiv \top$$
Here $[\![\sigma]\!] \subseteq [\![\tau]\!] = SN$ holds, because every saturated set is included in SN and $[\![\sigma]\!]$ is saturated, for arbitrary σ.

4. *Case*
$$\frac{\tau_1 \leq \sigma_1 \quad \sigma_2 \leq \tau_2}{\sigma \equiv \sigma_1 \to \sigma_2 \leq \tau_1 \leq \tau_2 \equiv \tau}$$
Recall that
$$[\![\sigma]\!] = [\![\sigma_1 \to \sigma_2]\!] = [\![\sigma_1]\!] \to [\![\sigma_2]\!]$$
and that
$$[\![\tau]\!] = [\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \to [\![\tau_2]\!]$$
In other words, we must show that
$$[\![\sigma_1]\!] \to [\![\sigma_2]\!] \subseteq [\![\tau_1]\!] \to [\![\tau_2]\!]$$
So assume, for arbitrary term N, that $N \in [\![\sigma_1]\!] \to [\![\sigma_2]\!]$. By the definition of $A \to B$, this means that
$$\forall P \in [\![\sigma_1]\!].\, NP \in [\![\sigma_2]\!] \tag{1}$$
Suppose now, for arbitrary Q, that
$$Q \in [\![\tau_1]\!] \tag{2}$$
Then, since the induction hypothesis entails that $[\![\tau_1]\!] \subseteq [\![\sigma_1]\!]$, it follows from (2) that we have $Q \in [\![\sigma_1]\!]$. But then, by (1), we get $NQ \in [\![\sigma_2]\!]$, and since (induction hypothesis again) we have $[\![\sigma_2]\!] \subseteq [\![\tau_2]\!]$, it follows that $NQ \in [\![\tau_2]\!]$. We have shown
$$\forall Q \in [\![\tau_1]\!].\, NQ \in [\![\tau_2]\!]$$
in other words, $N \in [\![\tau_1]\!] \to [\![\tau_2]\!]$. In total, we have shown that $N \in [\![\sigma_1]\!] \to [\![\sigma_2]\!]$ implies $N \in [\![\tau_1]\!] \to [\![\tau_2]\!]$, proving the desired inclusion. This case completes the induction.

We recall that the proof is by induction on the derivation of $\Gamma \vdash M : \sigma$, using that $[\![\sigma]\!]$ is saturated (Lemma 2.3 (4).)

Theorem 2.5 *(Strong normalization for simply typed lambda calculus)*

If $\Gamma \vdash M : \sigma$ *, then* M *is strongly normalizing*

We recall that the proof is by Proposition 2.4, using the identity valuation ρ_0 with $\rho_0(x) = x$ and using that $[\![\sigma]\!] \subseteq SN$ for every $\sigma \in$ *Type* (this inclusion follows because $[\![\sigma]\!]$ is saturated and every saturated set is, by definition, included in SN.)

3 The proof

We now indicate how a proof of strong normalization for $\lambda(\rightarrow, \bot, \top, B)$ can be obtained by a simple twist of the proof method of saturated sets. In this section, τ, σ range over the types of $\lambda(\rightarrow, \bot, \top, B)$, and the relation $\Gamma \vdash M : \tau$ is the typing relation of that system also.

The first step in the proof for non-structural subtyping is to extend the interpretation of types. This is done in the following definition.

Definition 3.1 (Extended interpretation of types)
For every type σ of $\lambda(\rightarrow, \bot, \top, B)$ a set $[\![\sigma]\!]$ is defined by

$$\begin{array}{lcl} [\![\alpha]\!] & = & SN \\ [\![b]\!] & = & SN \\ [\![\top]\!] & = & SN \\ [\![\bot]\!] & = & \bigcap_{S \in SAT} S \\ [\![\sigma \rightarrow \tau]\!] & = & [\![\sigma]\!] \rightarrow [\![\tau]\!] \end{array}$$

□

We remark that the extension of the standard interpretation is rather natural. Type constants are treated as type variables, $\top$ is interpreted as the top element (under set inclusion) of SAT, and $\bot$ is interpreted as the bottom element of SAT. Note that this interpretation of $\bot$ agrees with the standard interpretation of the second order polymorphic type $\forall\alpha.\alpha$, which is the canonical bottom ("absurd", non-inhabited) type of that system; see [2], [7]. Note also that one has $\bigcap_{S \in SAT} S \neq \emptyset$ by condition (a) of Definition 2.2.

It is clear that Lemma 2.3 still goes through (only the fourth property needs to be extended, and it obviously follows by the same proof, since the first three properties of Lemma 2.3 are still true.)

The second extension we need to add to the standard proof is the following lemma, which shows that the non-structural subtype ordering is sound with respect to the interpretation of types, and this is really the single key property needed for the extension.

Lemma 3.2 *(Soundness of non-structural subtype ordering)*

If $\sigma \leq \tau$*, then* $[\![\sigma]\!] \subseteq [\![\tau]\!]$

1. A subset $X \subseteq SN$ is called *saturated* if
   (a) $\forall n \geq 0. \forall R_1 \dots R_n \in SN. (x\vec{R}) \in X$, where x is any term variable,
   (b) $\forall n \geq 0. \forall R_1 \dots R_n \in SN. \forall Q \in SN.$

$$P\{x := Q\}\vec{R} \in X \Rightarrow (\lambda x.P)Q\vec{R} \in X$$

2. $SAT = \{X \subseteq \Lambda \mid X \text{ is saturated}\}$

$\square$

Lemma 2.3 *(Types are saturated sets)*

1. $SN \in SAT$

2. $A, B \in SAT \Rightarrow A \rightarrow B \in SAT$

3. Let $\{A_i\}_{i \in I}$ *be a collection of members of* SAT*, then* $\bigcap_{i \in I} A_i \in SAT$

4. For all $\sigma \in$ *Type one has* $[\![\sigma]\!] \in SAT$

We note that the proof of the last claim of the previous lemma is by induction on the structure of $\sigma \in$ *Type*, using the other claims of the lemma in that induction proof. In the case of the simply typed lambda calculus, only the first two claims stated in Lemma 2.3 are needed for establishing the last claim. The third claim is used, in the standard method, in the proof of strong normalization for the second order polymorphic lambda calculus, see [2] and [7]. We shall use all three of them in our proof of strong normalization for non-structural subtyping to be given in the next section.

The core of the strong normalization proof consists in showing that the typing rules are *sound* with respect to the interpretation of types as subsets of SN. The argument has the flavour of a term model construction; specifically, it can be viewed as a soundness argument with respect to a *type inference model* in the sense of [13], where soundness concerns derivable typing judgements rather than an equational theory.

To state the soundness property we need a few more definitions. A *valuation* in Λ is a map $\rho : V \rightarrow \Lambda$ where V is the set of term variables. If ρ is a valuation in Λ we define

$$[\![M]\!]\rho = M\{x_1 := \rho(x_1), \dots, x_n := \rho(x_n)\}$$

where $x_1, \dots, x_n$ are the free variables in M. We say that ρ *satisfies* $M : \sigma$, notation $\rho \models M : \sigma$, if $[\![M]\!]\rho \in [\![\sigma]\!]$. If Γ is a set of type assumptions, then ρ *satisfies* Γ, notation $\rho \models \Gamma$, if $\rho \models x : \sigma$ for all $x : \sigma \in \Gamma$. A set of type assumptions Γ *satisfies* $M : \sigma$, notation $\Gamma \models M : \sigma$, if

$$\forall \rho.(\rho \models \Gamma \Rightarrow \rho \models M : \sigma)$$

Then one has

Proposition 2.4 *(Soundness for simply typed lambda calculus)*

$$\Gamma \vdash M : \sigma \Rightarrow \Gamma \models M : \sigma$$

2 The background

In this section we review the necessary background on the method of saturated sets for proving strong normalization for the simply typed lambda calculus and the second order polymorphic lambda calculus, see [2], [7], [10], [4] for recent expositions. The next section will then extend the method to deal with non-structural subtyping. The method appears to have been developed from the method of *computability* as found in early work by Tait [19] (for the simply typed lambda calculus) and extended by Girard (method of *canditdats de reducubilité* for the second order polymorphic lambda calculus), see [4] and [7] for excellent expositions with further references to earlier contributions. The specific method of saturated sets as used in the present paper is taken from the exposition in [2]; it is similar to methods employed by Tait [20] and by Mitchell [15].

We follow closely the presentation of the method as given in [2], and for ease of reference we give all necessary definitions and lemmas (though not explicit proofs, for which the reader is referred to [2].)

In this section the set *Type* means the set of type expressions of the simply typed lambda calculus, and τ, σ range over such types. Also, the typing relation $\Gamma \vdash M : \tau$ means the relation of the simply typed lambda calculus in this section.

The method, to recall, can be viewed as an ingenious strengthening of the induction hypothesis that all well typed terms are strongly normalizing. As is easily verified, the simple claim of strong normalization does not work as induction hypothesis, and the strengthening revolves around the fact that the set SN of all strongly normalizing terms enjoys a simple (yet subtle) closure condition. The notion of *saturation* captures this closure condition technically.

The first step of the proof method is to define an interpretation of types as subsets of the set SN of strongly normalizing lambda terms.

Definition 2.1 (Interpretation of types)

1. $SN = \{M \in \Lambda \mid M \text{ is strongly normalizing}\}$
2. Let $A, B \subseteq \Lambda$, then define $A \to B \subseteq \Lambda$ by
$$A \to B = \{M \in \Lambda \mid \forall N \in A.\ MN \in B\}$$
3. For every $\sigma \in$ *Type* a set of lambda terms $[\![\sigma]\!]$ is defined by
$$\begin{array}{lcl} [\![\alpha]\!] & = & SN \\ [\![\sigma \to \tau]\!] & = & [\![\sigma]\!] \to [\![\tau]\!] \end{array}$$

□

We can now define the notion of saturation.

Definition 2.2 (Saturated sets)

the subtype ordering. We let τ, σ range over types, b ranges over B and α ranges over a denumerable set of type variables. The type language is then given by

$$\tau ::= \alpha \mid b \mid \top \mid \bot \mid \tau \to \tau'$$

The subtype ordering $\leq$ is extended from the ordering $\sqsubseteq$ on base types together with axioms for $\top, \bot$ to function types $\sigma \to \tau$ by the usual contravariant/covariant rule, as given below. The subtype relation is the least relation closed under the following rules

$$\bot \leq \tau \qquad \tau \leq \top$$

$$\frac{b \sqsubseteq b'}{b \leq b'} \qquad \frac{\tau_1 \leq \sigma_1 \quad \sigma_2 \leq \tau_2}{\sigma_1 \to \sigma_2 \leq \tau_1 \to \tau_2}$$

This definition gives rise to the ability to compare types that are differently structured, such as $\bot \leq \tau \to \sigma$, hence the name *non-structural* subtyping. For comments on the motivation for such systems we refer the reader to [21] and [24]; see [14] for an introduction to structural subtyping systems based on the simply typed lambda calculus. The set Λ of lambda terms, ranged over by M, N, P, Q, is given by

$$M ::= x \mid c \mid \lambda x.M \mid MM'$$

where x ranges over a denumerable set of term variables, and c ranges over a set C of *base constants*, such as, *e.g.*, arithmetic operations; we assume the constants to be typed, by a function *TypeOf* such that *TypeOf*$(c) = \tau$ if and only if τ is the type of c. See [1] for a comprehensive survey of the lambda calculus. Below we give the typing rules for the simply typed lambda calculus extended with non-structural subtyping; for more details about typed lambda calculi we refer to [2].

$$[\text{VAR}] \quad \Gamma, x : \tau \vdash x : \tau \qquad [\text{CNST}] \quad \Gamma \vdash c : \mathit{TypeOf}(c)$$

$$[\text{ABS}] \quad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma} \qquad [\text{APP}] \quad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash MN : \sigma}$$

$$[\text{SUB}] \quad \frac{\Gamma \vdash M : \tau \quad \tau \leq \sigma}{\Gamma \vdash M : \sigma}$$

We call this system $\lambda(\to, \bot, \top, B)$. The simply typed lambda calculus is given by the rules [VAR], [ABS] and [APP], and its type language is obtained by removing all types containing type constants b and the constants $\top, \bot$. Thatte's original system of partial typing [21] is obtained by removing the type $\bot$ and its subtyping axiom from $\lambda(\to, \bot, \top, B)$.

The aim is to show that every well typed term of the system $\lambda(\to, \bot, \top, B)$ is strongly normalizing with respect to β-reduction, *i.e.*, there exists no infinite β-reduction sequence starting from a well typed term of $\lambda(\to, \bot, \top, B)$. See [1] for more details on reduction and normalization.

Strong Normalization for Non-structural Subtyping via Saturated Sets

Jakob Rehof [*]

Abstract

We show that the standard method of saturated sets for proving strong normalization of β-reduction in the simply typed and second order polymorphic lambda calculus incorporates non-structural subtyping systems in a natural way. This shows that strong normalization for non-structural subtyping proved by Wand, O'Keefe and Palsberg [24] via coercion interpretations can be obtained in a straight-forward extension of the standard method. The proof presented here is compared to other proofs of strong normalization for subtyping systems.

Keywords: Functional programming, lambda-calculus, programming calculi, programming languages, strong normalization, subtyping.

1 Introduction

The purpose of the following notice is to make it clear that the standard proof method of *saturated sets* [2], [7], [4], [10] for strong normalization of β-reduction in the simply typed and the second order polymorphic lambda calculus (system F_2) has the ability to deal directly with non-structural subtyping systems in a quite natural way. In [24] Wand, O'Keefe and Palsberg prove that strong normalization holds for the simply typed lambda calculus extended with a form of non-structural subtyping in the style of Thatte's *partial typing* [21]. Their proof is by reduction to strong normalization of the simply typed lambda calculus, using a coercion interpretation. In this notice we show that no new methods are necessary in order to deal with non-structural subtyping, and we indicate that the standard method is very robust with respect to many other subtyping systems.

We consider the simply typed lambda calculus [2] extended with a subtyping rule. We assume a given partially ordered set $(B, \sqsubseteq)$ of *base types.* In addition, we assume a top type $\top$ and a bottom type $\bot$; this is a distinctive feature of non-structural subtyping. The top type will be larger than any type and the bottom type smaller than any type in

*Correspondence: Jakob Rehof, DIKU, Universitetsparken 1, DK-2100 Copenhagen East, Denmark. Electronic mail: `rehof@diku.dk`. Phone: +45 35321408. Fax: +45 35321405.