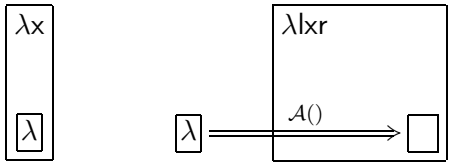
Deriving Strong Normalisation

Stéphane Lengrand

School of Computer Science, University of St Andrews, United Kingdom.
PPS, Université Paris 7, France. lengrand@pps.jussieu.fr
WWW home page: http://www.pps.jussieu.fr/lengrand/

We present a somewhat general technique to derive the strong normalisation of some specific terms of various calculi with explicit substitutions from the strong normalisation of some (well-chosen) λ-terms. One main application of the method is proving the *Preservation of Strong Normalisation* (PSN) of various explicit substitution calculi and the strong normalisation of the Cut-elimination in various sequent calculi.

- The notion of PSN concerns syntactic extensions of λ-calculus with their own reduction systems and states that if a λ-term is strongly normalising for the β-reduction, then it is still strongly normalising when considered as a term of the extended calculus undergoing its corresponding reduction system. Such extensions of λ-calculus are explicit reduction calculi like $\lambda\mathsf{x}$ [BR95], the reduction system of which reduces β-redexs and evaluate the explicit substitutions.
  The definition of the PSN property can be slightly generalised for calculi in which λ-calculus can be *embedded* (by a one-to-one translation, say $\mathcal{A}$), and the embedding need not be the identity/inclusion. In that case PSN states that if a λ-term is strongly normalising, then its encoding is also strongly normalising. This is the case for the explicit substitution calculus $\lambda\mathsf{lxr}$ introduced in [KL04] which requires terms to be linear and hence is not a syntactic extension of λ-calculus.



- Another application is deriving the strong normalisation of Cut-elimination in intuitionistic sequent calculus. When using a term-assignment system to denote proofs of sequent calculus, the Cut-rule is actually the typing rule of a term constructor that very much looks like an explicit substitution, so that Cut-elimination consists in fully propagating this "explicit substitution" operator.

1 The idea of simulation and its problems

The base idea to prove that a term M of a calculus with explicit substitutions is SN is to interpret it as a strongly normalising λ-term $\mathsf{L}(M)$ and then simulate its reductions by β-reductions of its interpretation. Hence, after a certain number of steps, only reductions can take place that no longer modify the interpretation. Then it would suffice to show that the reductions that do not necessarily induce at least one β-reduction terminate, so that no infinite reduction sequence can start from M, as illustrated in Figure 1.

For PSN, if $M = \mathcal{A}(t)$ where t is the λ-term supposed to be SN_β, then we would take $\mathsf{L}(M) = t$. For sequent calculus, it would be a typed (and hence strongly normalising) λ-term that denotes a proof in natural deduction of the same sequent (using Curry-Howard

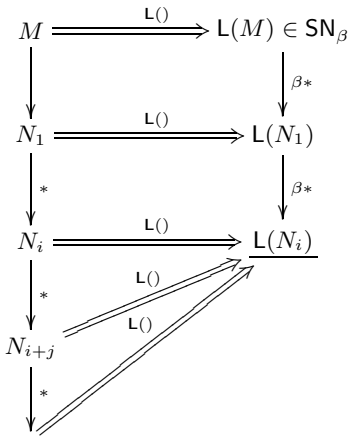


Figure 1. Deriving strong normalisation by simulation

correspondence), so that the idea of simulating Cut-elimination by β-reductions is closely related to Prawitz's correspondence [Pra65].

There is one problem in doing so: an encoding into λ-calculus that allows the simulation needs to interpret explicit substitutions by implicit substitutions like $t\{x = u\}$. But should x not be free in t, all reduction steps taking place within the term of which u is the encoding would not induce any β-reduction in $t\{x = u\}$.

Therefore, the sub-system consisting of all the reductions that are not necessarily simulated by at least one β-reduction is too big to be proven terminating (and very often it is not).

2 Refining the simulation by using Church-Klop's λI-calculus

Our contribution is to overcome this problem by encoding a calculus with explicit substitutions in Church-Klop's λI-calculus [Klo80] instead of λ-calculus. On the one hand, λI extends the syntax of λ-calculus with a "memory operator" so that, instead of being thrown away, a term N can be retained and carried along in a construct $[-, N]$. With this operator, those bodies of substitutions are encoded that would otherwise disappear, as explained above. On the other hand, λI restricts λ-abstractions to variables that have at least one free occurrence, so that β-reduction never erases its argument.

Doing so requires the encoding in λI to be non-deterministic, i.e. we define a relation $\mathcal{H}$ between the calculus and λI, and the reason for this is that, since the reductions in λI are non-erasing reductions, we need to add this memory operator at random places in the encoding, using such a rule:

$$\frac{M \mathcal{H} T}{M \mathcal{H} [T, U]} U \in \lambda I$$

For instance, $\lambda x.x \mathcal{H} \lambda x.[x, x]$ but also $\lambda x.x \mathcal{H} [\lambda x.x, \lambda z.z]$, so that both $\lambda x.[x, x]$ and $[\lambda x.x, \lambda z.z]$ (and also $\lambda x.x$) are encodings of $\lambda x.x$.

Another problem that arises is that explicit substitutions are pushed down in the terms of the calculus, whereas the memory operator cannot be pushed down in λI, making the simulation difficult. Hence, we use a subtle side-condition for the encoding of an explicit

substitution $\langle N/x\rangle M$:

$$\frac{M \mathcal{H} T \quad N \mathcal{H} U}{\langle N/x\rangle M \mathcal{H} T\{x = U\}}\ x \in FV(T) \text{ or } N \in \mathsf{SN}$$

1. Always requiring $x \in FV(T)$ would be too strong, because the substitution $\langle N/x\rangle$ can be propagated into the sub-terms of M whereas x has no reason to be free in the corresponding sub-terms of T, so the simulation theorem would not hold.
2. On the contrary, no side-condition would make the encoding too relaxed because U could disappear and the reduction steps that could take place within N would be simulated by zero reduction steps so we would have the same problem as with λ-calculus.
3. The side-condition $x \in FV(T)$ or $N \in \mathsf{SN}$ is the adequate balance:
   - First, when we reduce N or reduce T, the side-condition is still preserved (because reductions in λI are non-erasing).
   - If $N \notin \mathsf{SN}$, then $x \in FV(T)$ is required and for the same reason as in point 1 we cannot simulate the propagation of $\langle N/x\rangle$. So we need to prove that from a term which is not strongly normalising we can find an infinite reduction sequence that never propagates such a substitution, and this can be done because instead of propagating $\langle N/x\rangle$, we can leave it where it is and infinitely reduce N instead.
   - If on the contrary $N \in \mathsf{SN}$, then we can simulate its propagation because we never care whether $x \in FV(\ldots)$, but for the same reason as in point 2, reduction steps within N might be simulated by zero reduction steps, but because $N \in \mathsf{SN}$ this can only happen finitely many times (we call those reductions *safe reductions*).

Eventually, the reduction relation of the explicit substitution calculus is split into two parts Y and Z that satisfy the following simulation theorem:
If $M \mathcal{H} T$ and $M \longrightarrow_Y N$, then there is a U such that $N \mathcal{H} U$ and $T \longrightarrow^+ U$
If $M \mathcal{H} T$ and $M \longrightarrow_Z N$, then there is a U such that $N \mathcal{H} U$ and $T \longrightarrow^* U$

Now it must be proven that every term M can be encoded into a strongly normalising term of λI. This depends on the calculus that is being treated, but the following method generally works:

- Encode the term M as a strongly normalising λ-term t, such that no sub-term is lost, i.e. *not* using implicit substitutions. For PSN, the original λ-term would do, because it is strongly normalising by hypothesis; for a proof-term of sequent calculus, t would be a λ-term typed in an appropriate typing system, the typing tree of which is derived from the proof-tree of the sequent (we would get $t \in \mathsf{SN}_\beta$ using a theorem stating that typed terms are SN_β).
- Then encode t in λI with the following encoding:

$$\boxed{\begin{array}{lll} \mathsf{i}(x) & = x & \\ \mathsf{i}(\lambda x.t) & = \lambda x.\mathsf{i}(t) & x \in FV(\mathsf{i}(t)) \\ \mathsf{i}(\lambda x.t) & = \lambda x.[\mathsf{i}(t), x] & x \notin FV(\mathsf{i}(t)) \\ \mathsf{i}(t\ u) & = \mathsf{i}(t)\ \mathsf{i}(u) & \end{array}}$$

  In [KL04] we prove that if a λ-term t is strongly normalising for β-reductions, then $\mathsf{i}(t)$ is weakly normalising in λI. The proof simply consists in simulating an adequate reduction sequence that starts from t and ends with a normal form, the encoding of which is a normal form of λI. What makes this simulation work is the fact that the reduction sequence is provided by a perpetual strategy. Also, weak normalisation implies strong normalisation in λI [Ned73], so $\mathsf{i}(t)$ is strongly normalising.

- Then prove that $\mathsf{i}(t)$ reduces to one of the non-deterministic encodings of M in λI, that is, that there is a term T such that $M \mathcal{H} T$ and $\mathsf{i}(t) \longrightarrow^* T$.

The technique is summarised in Figure 2.

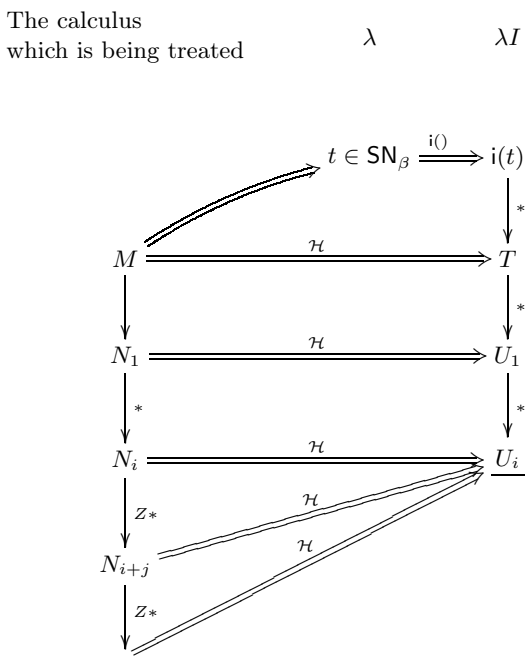


Figure 2. The general technique to prove that $M \in \mathsf{SN}$

Finally, it remains to be proven that the relation Z that consists of the reductions that are not simulated in at least one step is now small enough to be terminating.

3 Conclusion

This technique works for $\lambda\mathsf{x}$, giving a new proof of PSN (already shown in [BR95]), as well as for proving PSN of the explicit substitution calculus λ_{lxr} of [KL04], and for various sequent calculi that range from propositional logic to a logic as expressive as the Calculus of Constructions, and we believe that it can be applied to many other calculi.

References

[BR95] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95 – Computer Science in the Netherlands*, pages 62–72, Koninklijke Jaarbeurs, Utrecht, November 1995.

[KL04] D. Kesner and Stéphane Lengrand. Broadening the horizon of the explicit substitution paradigm via a logical model. *submitted to CSL' 2004.*

[Klo80] Jan-Willem Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, 1980. PhD Thesis.

[Ned73] Rob Nederpelt. *Strong Normalization in a Typed Lambda Calculus with Lambda Structured Types*. PhD thesis, Eindhoven University of Technology, 1973.

[Pra65] D. Prawitz. Natural deduction. a proof-theoretical study. In *Acta Universitatis Stockholmiensis*, volume 3. Almqvist & Wiksell, 1965.