[16] G. S. Smith. Principal type schemes for functional programs with overloading and subtyping. *Science of Computer Programming*, 23:197–226, 1994.

[17] J. Tiuryn. Subtype inequalities. In *Proc. 7th Annual IEEE Symp. on Logic in Computer Science (LICS), Santa Cruz, California*, pages 308–315. IEEE, IEEE Computer Society Press, June 1992.

[18] V. Trifonov and S. Smith. Subtyping constrained types (to appear). In *Static Analysis Symposium, Aachen, Germany*, September 1996.

[19] D. Turner. Miranda: A non-strict functional language with polymorphic types. In *Proc. Functional Programming Languages and Computer Architecture*, pages 1–16, Nancy, France, 1985. Springer. Lecture Notes in Computer Science, Vol. 201.

[20] M. Wand. A simple algorithm and proof for type inference. *Fundamenta Informaticae*, X:115–122, 1987.

## References


[1] A.V. Aho, R. Garey, and J.D. Ullman. The transitive reduction of a directed graph. *SIAM Journal of Computing*, 1(2):131–137, June 1972.

[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] Y. Fuh and P. Mishra. Polymorphic subtype inference: Closing the theory-practice gap. In *Proc. Int'l J't Conf. on Theory and Practice of Software Development*, pages 167–183, Barcelona, Spain, March 1989. Springer-Verlag.

[4] Y. Fuh and P. Mishra. Type inference with subtypes. *Theoretical Computer Science (TCS)*, 73:155–175, 1990.

[5] M. Hoang and J.C. Mitchell. Lower bounds on type inference with subtypes. In *Proc. 22nd Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 176–185. ACM Press, 1995.

[6] P. Hudak and J. Fasel. A gentle introduction to Hhaskell. *Sigplan Notices*, 27(5):Section T, 1992.

[7] Stefan Kaes. Type inference in the presence of overloading, subtyping and recursive types. In *Proc. ACM Conf. on LISP and Functional Programming (LFP), San Francisco, California*, pages 193–204. ACM, ACM Press, June 1992. also in LISP Pointers, Vol. V, Number 1, January-March 1992.

[8] P. Kanellakis, H. Mairson, and J. Mitchell. Unification and ML type reconstruction. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic — Essays in Honor of Alan Robinson.* MIT Press, 1991.

[9] P. Lincoln and J. Mitchell. Algorithmic aspects of type inference with subtypes. In *Proc. 19th Annual ACM SIGPLAN–SIGACT Symposium on Principles of Programmin Languages (POPL), Albuquerque, New Mexico*, pages 293–304. ACM Press, January 1992.

[10] R. Milner, M. Tofte., and R. Harper. *The Definition of Standard ML.* MIT Press, 1990.

[11] J. Mitchell. Coercion and type inference (summary). In *Proc. 11th ACM Symp. on Principles of Programming Languages (POPL)*, pages 175–185, 1984.

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

[13] F. Pottier. Simplifying subtyping constraints. In *International Conference on Finctional Programming*, pages 122–133. ACM, ACM Press, May 1996.

[14] Vaughan Pratt and Jerzy Tiuryn. Satisfiability of inequalities in a poset. *Studia Logica, Helene Rasiowa memorial issue (to appear)*, 1996.

[15] Dag Prawitz. *Natural deduction.* Almquist & Wiksell, Uppsala 1965.

$$[const] \quad C \vdash_P b \leq b' \text{ , provided } b \leq_P b'$$

$$[ref] \quad C \vdash_P \tau \leq \tau$$

$$[hyp] \quad C \cup \{\tau \leq \tau'\} \vdash_P \tau \leq \tau'$$

$$[trans] \quad \frac{C \vdash_P \tau \leq \tau' \quad C \vdash_P \tau' \leq \tau''}{C \vdash_P \tau \leq \tau''}$$

$$[arrow] \quad \frac{C \vdash_P \tau_1' \leq \tau_1 \quad C \vdash_P \tau_2 \leq \tau_2'}{C \vdash_P \tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'}$$

Figure 7: Subtype logic

$$[var] \quad C, \Gamma \cup \{x : \tau\} \vdash x : \tau$$

$$[base] \quad C, \Gamma \vdash c : \tau \text{ , where TypeOf}(c) = \tau$$

$$[abs] \quad \frac{C, \Gamma \cup \{x : \tau\} \vdash M : \tau'}{C, \Gamma \vdash \lambda x.M : \tau \to \tau'}$$

$$[app] \quad \frac{C, \Gamma \vdash M : \tau \to \tau' \quad C, \Gamma \vdash N : \tau}{C, \Gamma \vdash M\,N : \tau'}$$

$$[sub] \quad \frac{C, \Gamma \vdash M : \tau \quad C \vdash_P \tau \leq \tau'}{C, \Gamma \vdash M : \tau'}$$

Figure 8: Subtype system

computationally *efficient* transformations which can do this. Finally, the lower bound construction may well yield lower bounds for other, non-atomic, subtyping systems.

# 7 Related work

Most closely related to the present work is that of Hoang and Mitchell [5] and that of Fuh and Mishra [3]. In [5], the authors prove a linear lower bound for principal typings. However, this is using a generic instance relation, and lower bounds for this relation are expectedly harder to obtain than our bound, since less information is available about instance related typings using generic instance. Also, the techniques used by Kanellakis, Mairson and Mitchell in [8] to analyze type size for simply typed lambda calculus and ML are related to our technique for the lower bound proof in Section 5, but the main part of that proof relies on new techniques taylored for subtyping. The fundamental work by Fuh and Mishra [3] on subtype simplification provided important background in the form of their $S$- and $G$-transformations. Fuh and Mishra operate with a notion of minimal typings, but their notion is defined in terms of their transformations, and, as shown by our incompletenss results, this notion is distinct from ours. As far as we know, our notion of minimality is the first purely logical notion, which relates all typings in a given equivalence class, independently of particular transformations. Recent work by Pottier [13] and by Trifonov and Smith [18] introduce more powerful notions of simplification based on stronger entailment relations in the subtype logic, some of which are not known to be decidable.


# Acknowledgements

I am indebted to Fritz Henglein for introducing me to many of the problems studied in this paper and for many helpful discussions. This paper evidently owes a lot to the "2-crowns" which were used by Pratt and Tiuryn [14], [17] in studying the complexity of satisfiability of inequalities. Thanks are due to Vaughan Pratt for helpful discussions on details of [14].


# A Type system

The subtyping system shown in Figure 8 is a logic for deriving judgements of the form

$$C, \Gamma \vdash_P M : \tau$$

where $C$ is a *constraint set*, consisting of subtyping hypotheses of the form $\tau \leq \tau'$ $\Gamma$ is a set of *type assumptions* of the form $x : \tau$, where it is assumed that no variable $x$ occurs twice in $\Gamma$, $M$ is a lambda term, and $\tau$ a type.

The system uses, in the rule [*sub*] for coercion, a standard logic $\vdash_P$, shown in Figure 7, for deriving subtyping relations of the form $\tau \leq \tau'$ between types, given assumptions $C$ of atomic subtyping relations of the form $A \leq A'$. The rules are parametric in $P$, which is a given finite poset of *base types* (type constants), containing relations such as, e.g., `int` $\leq$ `real`. The rule [*base*] assumes a function TypeOf assigning types to term constants.

A judgement $C, \Gamma \vdash_P M : \tau$ is a *derivable atomic subtyping judgement* iff it is derivable in the proof system of Figure 8 *and* $C$ *is atomic*, i.e., $C$ contains only subtyping hypotheses of the form $A \leq A'$ relating atomic types.

For more details and standard properties of atomic subtyping we refer to [12].

It is tedious but not difficult to see that all the relevant effects on coercions which is exploited in our construction above are also present under this encoding, using the encoded pair-type $(\tau_1 \to \tau_2 \to \sigma) \to \sigma$ (for arbitrary type $\sigma$) to encode the type $\tau_1 * \tau_2$. In particular, it is still possible to eliminate all internal variables using $G$- and $S$-simplification on the constraint sets extracted by the standard procedure, and Theorem 4.4 can therefore be used as before. As a result, the construction can be carried out for an arbitrary poset of ground types, because the encodings eliminate reference to constants. We therefore have

**Theorem 5.2** *For any poset $P$ of base types it holds for arbitrarily large $n$, that there exist closed expressions of length $n$ such that any principal typing for the expressions has a coercion set containing $2^{\Omega(n)}$ distinct type variables and a type containing $2^{\Omega(n)}$ distinct type variables.*

*Proof*: Take the series of terms $\mathbf{Q}_n$ (using encodings as shown, to eliminate assumptions on $P$); clearly, the size of $\mathbf{Q}_n$ is of the order of $n$, and the main lemma (Lemma 5.1 ) shows that $\mathbf{Q}_n$ has a minimal principal typing $t$ with coercion set containing more than $2^n$ distinct variables and a type containing more than $2^n$ distinct variables. Any other principal typing $t'$ satisfies $t \approx t'$, and hence $t'$ must have at least as many distinct type variables in both coercion set and type as $t$, because $t \lesssim t'$ by the definition of minimality. □

The theorem immediately implies that the *dag-size* of coercion sets as well as of types in principal typings are of the order $2^{\Omega(n)}$ in the worst case. Since it follows from standard type inference algorithms such as those of [12] and [3] that a principal atomic typing can be obtained by extracting a set $C$ of (possibly non-atomic) coercions of size linear in the size of the term followed by a match-step which expands and decomposes $C$ to atomic constraints under at most an exponential blow-up, we have

**Corollary 5.3** *The dag-size of coercion sets as well as of types in atomic principal typings is of the order $2^{\Theta(n)}$ in the worst case.*

# 6 Conclusion and further work

We have defined a notion of minimal typings for simply typed lambda calculus extended with subtyping. The notion of minimality is purely logical, defined in terms of the type system and the notion of instance of Fuh and Mishra [3]. We have shown that every judgement has a unique minimal representative in every equivalence class with respect to instantiation. Since our notion of minimality is defined independently from any particular notion of simplification, we could meaningfully investigate completeness and incompleteness properties of known simplifications. We found that the typing transformations discussed by Fuh and Mishra are incomplete, but we have also shown that in the special case where all variables in a typing are observable, $S$-simplification is complete for minimization. This result, together with the characterization of minimal typings, allowed us to prove a tight exponential lower bound for the dag-size of coercion sets as well as of types in most general typings.

It is possible to envisage more powerful typing transformations than such that are sound with respect to the notion of instance studied here. However, the instance relation studied here is natural and powerful enough to be an interesting object of study, and our results and techniques may be of value for more powerful frameworks also. In particular, we beleive that the exponential lower bound construction should provide an interesting test for more powerful transformations. In fact, one may well ask if there exists *any* semantically sound transformations strong enough to brake the worst case exponential barrier. Furthermore, we may ask (and doubt) whether there are any

### 5.3 Encoding the construction in pure lambda calculus

We now show that the conditional construct, the pairing construct and the projection functions can be eliminated from the construction.

First consider the conditional. The only property of the conditional used in the construction is that it requires the types of both of its branches, say $M_1$ and $M_2$, to be coerced to a common supertype. This can be effected without the conditional by placing $M_1$ and $M_2$ in the context

$$\lambda x.\mathbf{K}(x\,M_1)(x\,M_2)$$

which requires the types of $M_1$ and $M_2$ to be coerced to the domain type of $x$. This eliminates the need for the conditional.

As for the pairing construction with projections, we might be tempted to use the standard lambda-calculus encodings, but this will not work here, since the simply typed lambda-calculus has no type-correct equivalent of pairing with projection. This follows, via the Curry-Howard isomorphism, from the fact that one cannot define logical conjunction as a derived notion from implication in minimal logic (see [15]) However, our construction does not need the full power of pairing and projections, since the projections are only used to permute a pair, in order to force the type of $\langle M, N\rangle$ to be equal to the type of $\langle N, M\rangle$. At the same time, the technique used to achieve the exponential blow-up requires us to type an abstraction $\lambda z.\ldots$, which expects a pair $z$ and copies it into new pairs that are double the size of the original pair [5] However, instead of writing expressions such as

$$\mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle z, \langle \mathtt{2nd}\, z, \mathtt{1st}\, z\rangle\rangle\ \mathit{else}\ \langle\langle \mathtt{2nd}\, z, \mathtt{1st}\, z\rangle, z\rangle$$

we can obtain a similar effect on typings by writing the term

$$\mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle\!\langle\, \langle\!\langle\, z, s\,\rangle\!\rangle, \langle\!\langle\, z, t\,\rangle\!\rangle\,\rangle\!\rangle\ \mathit{else}\ \langle\!\langle\, \langle\!\langle\, s, z\,\rangle\!\rangle, \langle\!\langle\, t, z\,\rangle\!\rangle\,\rangle\!\rangle$$

where $s$ and $t$ are free variables and where $\langle\!\langle\, M, N\,\rangle\!\rangle = \lambda p.(p\,M)\,N$ is just the standard encoding of pairing.

Summing up, we use the definitions

$$\begin{array}{lcl}
\mathit{eqty}(M,N) & = & \lambda x.\mathbf{K}(xM)(xN) \\
\langle\!\langle\, M, N\,\rangle\!\rangle & = & \lambda p.(pM)N \\
\mathbf{P}_{f,s,t} & = & \lambda z.\mathbf{K} \\
 & & \mathit{eqty}(\langle\!\langle\, \langle\!\langle\, z, s\,\rangle\!\rangle, \langle\!\langle\, z, t\,\rangle\!\rangle\,\rangle\!\rangle, \langle\!\langle\, \langle\!\langle\, s, z\,\rangle\!\rangle, \langle\!\langle\, t, z\,\rangle\!\rangle\,\rangle\!\rangle) \\
 & & (f\ z) \\
\mathbf{P}^{n+1}N & = & \mathbf{P}_{f_{n+1},s_{n+1},t_{n+1}}(\mathbf{P}^n N) \\
\mathbf{Q}_n & = & \lambda f_{[n]}.\lambda s_{[n]}.\lambda t_{[n]}.\lambda x.\lambda y.\mathbf{P}^n \mathit{eqty}(x,y)
\end{array}$$

[5] This is in fact where the *standard* encodings of pairing and projection will fail to type if used to encode the present construction directly; the problem is that we cannot find a common (standard, encoded) type for all of the occurrences of $z$ in the expressions $\mathbf{P}_f$, even using subtyping.

```
{b0<b3,  b0<b4,   b1<b4,
b1<b3,   b3<b8,   b3<b7,   b3<b9,   b3<b6,   b4<b9,   b4<b6,   b4<b8,
b4<b7,   b6<b15,  b6<b13,  b6<b17,  b6<b11,  b7<b16,  b7<b14,  b7<b18,
b7<b12,  b8<b17,  b8<b11,  b8<b15,  b8<b13,  b9<b18,  b9<b12,  b9<b16,
b9<b14,  b11<b28, b11<b24, b11<b32, b11<b20, b12<b29, b12<b25, b12<b33,
b12<b21, b13<b30, b13<b26, b13<b34, b13<b22, b14<b31, b14<b27, b14<b35,
b14<b23, b15<b32, b15<b20, b15<b28, b15<b24, b16<b33, b16<b21, b16<b29,
b16<b25, b17<b34, b17<b22, b17<b30, b17<b26, b18<b35, b18<b23, b18<b31,
b18<b27, b20<b53, b20<b45, b20<b61, b20<b37, b21<b54, b21<b46, b21<b62,
b21<b38, b22<b55, b22<b47, b22<b63, b22<b39, b23<b56, b23<b48, b23<b64,
b23<b40, b24<b57, b24<b49, b24<b65, b24<b41, b25<b58, b25<b50, b25<b66,
b25<b42, b26<b59, b26<b51, b26<b67, b26<b43, b27<b60, b27<b52, b27<b68,
b27<b44, b28<b61, b28<b37, b28<b53, b28<b45, b29<b62, b29<b38, b29<b54,
b29<b46, b30<b63, b30<b39, b30<b55, b30<b47, b31<b64, b31<b40, b31<b56,
b31<b48, b32<b65, b32<b41, b32<b57, b32<b49, b33<b66, b33<b42, b33<b58,
b33<b50, b34<b67, b34<b43, b34<b59, b34<b51, b35<b68, b35<b44, b35<b60,
b35<b52, b37<b102, b37<b86, b37<b118, b37<b70, b38<b103, b38<b87, b38<b119,
b38<b71, b39<b104, b39<b88, b39<b120, b39<b72, b40<b105, b40<b89, b40<b121,
b40<b73, b41<b106, b41<b90, b41<b122, b41<b74, b42<b107, b42<b91, b42<b123,
b42<b75, b43<b108, b43<b92, b43<b124, b43<b76, b44<b109, b44<b93, b44<b125,
b44<b77, b45<b110, b45<b94, b45<b126, b45<b78, b46<b111, b46<b95, b46<b127,
b46<b79, b47<b112, b47<b96, b47<b128, b47<b80, b48<b113, b48<b97, b48<b129,
b48<b81, b49<b114, b49<b98, b49<b130, b49<b82, b50<b115, b50<b99, b50<b131,
b50<b83, b51<b116, b51<b100, b51<b132, b51<b84, b52<b117, b52<b101,b52<b133,
b52<b85, b53<b118, b53<b70, b53<b102, b53<b86, b54<b119, b54<b71, b54<b103,
b54<b87, b55<b120, b55<b72, b55<b104, b55<b88, b56<b121, b56<b73, b56<b105,
b56<b89, b57<b122, b57<b74, b57<b106, b57<b90, b58<b123, b58<b75, b58<b107,
b58<b91, b59<b124, b59<b76, b59<b108, b59<b92, b60<b125, b60<b77, b60<b109,
b60<b93, b61<b126, b61<b78, b61<b110, b61<b94, b62<b127, b62<b79, b62<b111,
b62<b95, b63<b128, b63<b80, b63<b112, b63<b96, b64<b129, b64<b81, b64<b113,
b64<b97, b65<b130, b65<b82, b65<b114, b65<b98, b66<b131, b66<b83, b66<b115,
b66<b99, b67<b132, b67<b84, b67<b116, b67<b100, b68<b133, b68<b85, b68<b117,
b68<b101},

{f5:((((((b68*b67)*(b66*b65))*((b64*b63)*(b62*b61)))*(((b60*b59)*(b58*b57))*
    ((b56*b55)*(b54*b53))))*((((b52*b51)*(b50*b49))*((b48*b47)*(b46*b45)))*
    (((b44*b43)*(b42*b41))*((b40*b39)*(b38*b37)))))->b36),
f4: (((((b35*b34)*(b33*b32))*((b31*b30)*(b29*b28)))*(((b27*b26)*(b25*b24))*
    ((b23*b22)*(b21*b20))))->b19),
f3: ((((b18*b17)*(b16*b15))*((b14*b13)*(b12*b11)))->b10),
f2: (((b9*b8)*(b7*b6))->b5),f1:((b4*b3)->b2),
y : b0,
x : b1}|-

((\z.((((\x1.\x2.x1) if true then <z,<(snd z),(fst z)>> else <<(snd z)
,(fst z)>,z>)) ((f5) z))) ((\z.((((\x1.\x2.x1) if true then <z,<(snd z
),(fst z)>> else <<(snd z),(fst z)>,z>)) ((f4) z))) ((\z.((((\x1.\x2.x
1) if true then <z,<(snd z),(fst z)>> else <<(snd z),(fst z)>,z>)) ((f
3) z))) ((\z.((((\x1.\x2.x1) if true then <z,<(snd z),(fst z)>> else <
<(snd z),(fst z)>,z>)) ((f2) z))) ((\z.((((\x1.\x2.x1) if true then <z
,<(snd z),(fst z)>> else <<(snd z),(fst z)>,z>)) ((f1) z))) if true th
en <x,y> else <y,x>))))):

((((((b133*b132)*(b131*b130))*((b129*b128)*(b127*b126)))*(((b125*b124)
*(b123*b122))*((b121*b120)*(b119*b118))))*((((b117*b116)*(b115*b114))*
((b113*b112)*(b111*b110)))*(((b109*b108)*(b107*b106))*((b105*b104)*
(b103*b102)))))*(((((b101*b100)*(b99*b98))*((b97*b96)*(b95*b94)))*
(((b93*b92)*(b91*b90))*((b89*b88)*(b87*b86))))*((((b85*b84)*(b83*b82))*
((b81*b80)*(b79*b78)))*(((b77*b76)*(b75*b74))*((b73*b72)*(b71*b70))))))
```

Figure 6: Minimal typing for $\mathbf{P}^5 cond_{xy}$

implemented $G$- and $S$-simplification and used the system to generate the minimal typings for the first few terms $\mathbf{Q}_n$. Figure 6 shows the output for $n = 5$ (showing actually the term $\mathbf{P}^5 cond_{xy}$), and the reader may discern an exponential number of 2-crowns in the coercion set.

$T^m_{n+1}$, and hence we need apply no coercion to $z$ or $f_{n+1}$. Using elimination of internal variables by $G$-simplification, it is then straight-forward to see that a principal completion of $\mathbf{Q}_{n+1}$ is obtained by inserting coercions as follows (assuming suitable coercions inside $(\mathbf{P}^n cond_{x,y})$)

$$
\begin{array}{l}
\lambda f_{n+1} : T^m_{n+1} \to \gamma.\lambda f_{[n]} : [\sigma'_n].\lambda x : \alpha.\lambda y : \beta. \\
(\lambda z : T^m_{n+1}.\mathbf{K} \\
\quad (\mathit{if}\ \mathtt{true}\ \mathit{then} \\
\quad \langle \uparrow^{\theta_1 * \theta_2}_{T_1 * T_2} z, \langle \mathtt{2nd}\ \uparrow^{T_1 * \theta_3}_{T_1 * T_2} z, \mathtt{1st}\ \uparrow^{\theta_4 * T_2}_{T_1 * T_2} z \rangle \rangle \\
\quad \mathit{else} \\
\quad \langle \langle \mathtt{2nd}\ \uparrow^{T_1 * \theta_1}_{T_1 * T_2} z, \mathtt{1st}\ \uparrow^{\theta_2 * T_2}_{T_1 * T_2} z \rangle, \uparrow^{\theta_3 * \theta_4}_{T_1 * T_2} z \rangle) \\
\quad (f_{n+1}\ z)) \\
\quad (\mathbf{P}^n cond_{x,y})
\end{array}
$$

Here $T_1$ and $T_2$ are such that $T_1 * T_2 = T^m_{n+1}$, so $T_1$ and $T_2$ match each other and each has $2^n$ distinct variables, and $\lambda f_{[n]} : [\sigma'_n]$ abbreviates the list of typed parameters $\lambda f_i : \sigma_i \to v_i$, $i = 1 \ldots n$. The types $\theta_i$ are renamings, using fresh variables, of $T_1$ (or, equivalently, of $T_2$.) This shows that a principal typing for $\mathbf{Q}_{n+1}$ can be obtained by adding to $C_n$ the two 2-crowns $C_1$, $C_2$ of coercions shown in the completion above:

$$
\begin{array}{rcl}
C_1 & = & \{T_1 \le \theta_1, T_1 \le \theta_2, T_2 \le \theta_2, T_2 \le \theta_1\} \\
C_2 & = & \{T_1 \le \theta_4, T_1 \le \theta_3, T_2 \le \theta_3, T_2 \le \theta_4\}
\end{array}
$$

Now, these crowns are not atomic, since the types in them are all of the shape of $T^m_n$, hence to add them to $C_n$ we need to decompose them. It is easy to see that each $C_i$ decomposes into a set $C'_i$ of $2^n$ atomic 2-crowns (since $T^m_n$ has $2^n$ leaves) resulting in a total of $2 \cdot 2^n = 2^{n+1}$ new crowns; since the $\theta_i$ have fresh variables, each $C_i$ contains $2 \cdot 2^n = 2^{n+1}$ new, distinct variables, and since, by induction, $C_n$ already has at least $2^{n+1}$ distinct variables, it follows that the number of distinct variables in $C_{n+1} = C_n \cup C'_1 \cup C'_2$ is at least $2^{n+1} + 2^{n+1} = 2^{n+2}$.

The type of $\mathbf{Q}_{n+1}$ under the principal typing shown has the form

$$
\begin{array}{l}
(T^m_{n+1} \to \gamma) \to \sigma'_1 \to \ldots \to \sigma'_n \to \\
\alpha \to \beta \to ((\theta_1 * \theta_2) * (\theta_3 * \theta_4))
\end{array}
$$

which can be renamed to $\tau^{[n+1]}$. Since, by induction, all variables in $C_n$ are observable, and since all new variables in $C_{n+1}$ are in the $\theta_i$, it follows that all variables in $C_{n+1}$ are observable in the typing of $\mathbf{Q}_{n+1}$.

It remains to show that the typing shown for $\mathbf{Q}_{n+1}$ is *minimal*. By induction hypothesis, $C_n$ is minimal as part of a minimal typing of $\mathbf{Q}_n$, hence, in particular, it is $S$-simplified. By the shape of 2-crowns, it is easy to verify that adding the crowns of $C'_1$ and $C'_2$ preserves this property, so $C_{n+1}$ is $S$-simplified also. It then follows from Theorem 4.4 that the typing shown for $\mathbf{Q}_{n+1}$ is the minimal one. This completes the proof of the lemma. □

A critical part of the construction given above is that the principal typings of the terms $\mathbf{Q}_n$ have constraints sets with all variables observable. This is what allows us to use Theorem 4.4 in order to conclude, in the proof of the preceeding lemma, that typings are *minimal*. The rôle of the application $(f\ z)$ in the terms $\mathbf{P}_f$ is to make the type of the argument $z$ observable in the typing of $\mathbf{Q}_n$; this greatly simplifies the correctness argument for the construction, since it makes it easier to apply Theorem 4.4.

The terms $\mathbf{Q}_n$ are simple enough to enable $S$-simplification and *G-simplification* to do the job of transforming a typing extracted by the *standard procedure* into the minimal one. We have

To prepare for a proof that the $\mathbf{Q}_n$ behave as claimed, let $\alpha$ and $\beta$ be two distinct variables and let $u_0, u_1, \ldots, u_k, \ldots$ and $v_0, v_1, \ldots, v_k, \ldots$ be two enumerations of infinitely many distinct type variables (so, all the $v_i$ are different from all the $u_j$ and all of these are distinct from $\alpha$ and $\beta$.) Let $T_n^m$ denote the pair-type constructed as the fully balanced binary tree of height $n$ with $2^n$ leaf nodes, with internal nodes labeled by $*$ and leaf nodes labeled by variables $u_m, u_{m+1}, \ldots, u_{m+2^n-1}$, from left to right. So, for instance, $T_0^0$ is just the variable $u_0$, $T_1^0$ is the type $u_0 * u_1$, $T_2^0$ is the type $(u_0 * u_1) * (u_2 * u_3)$, etc. Let us say that a type $\tau$ *has the shape of* $T_n^m$ if $\tau$ is built from $*$ and variables only and, moreover, $\tau$ matches $T_n^m$; so, in this case, $\tau$ differs from $T_n^m$ only by having possibly other variables than $T_n^m$ at the leaves.

Define the type $\tau^{[n]}$ for $n \geq 0$ by setting

$$\begin{array}{lcl} \tau^{[0]} & = & \alpha \to (\beta \to T_1^0), \\ \tau^{[n]} & = & (\sigma_n \to v_n) \to \ldots (\sigma_1 \to v_1) \to \\ & & \alpha \to \beta \to T_{n+1}^{k+1} \end{array}$$

where $\sigma_i$ is a renaming of $T_i^0$ for $i = 1 \ldots n$, using fresh variables which occur nowhere else in the type. We assume that the $\sigma_i$ use the variables $v_1, v_2, \ldots, v_k$.

We show by induction on $n \geq 0$ that

**Lemma 5.1** (Main Lemma) *The minimal principal typing of* $\mathbf{Q}_n$ *has the form* $C_n, \emptyset \vdash_P \mathbf{Q}_n : \tau^{[n]}$ *where all variables in* $C_n$ *are observable, and* $C_n$ *contains at least* $2^{n+1}$ *distinct variables.*

*Proof*: To prove the lemma, consider the term $\mathbf{Q}_0$ for the base case. It is easy to see that a principal typing of $\mathbf{Q}_0$ has the form $C_0, \emptyset \vdash_P \lambda x.\lambda y.cond_{x,y} : \alpha \to (\beta \to u_0 * u_1)$ with $C_0 = \{\alpha \leq u_0, \alpha \leq u_1, \beta \leq u_0, \beta \leq u_1\}$. This typing is derived by the *standard procedure* described earlier, where $G$-simplification has been performed to eliminate internal variables. The resulting typing derivation is given by the completion:

$$\begin{array}{l} \lambda x : \alpha.\lambda y : \beta.\mathit{if}\ \mathtt{true} \\ \mathit{then}\ \langle \uparrow_\alpha^{u_0} x, \uparrow_\beta^{u_1} y \rangle \\ \mathit{else}\ \langle \uparrow_\beta^{u_0} y, \uparrow_\alpha^{u_1} x \rangle \end{array}$$

All variables in $C_0$ are observable, and, moreover, $C_0$ is a 2-crown, from which it easily follows that the typing shown is $S$-simplified. It then follows from Theorem 4.4 that the typing is the *minimal* principal typing for $\mathbf{Q}_0$, and this typing satisfies the claim $(*)$ for the case $n = 0$.

For the inductive case, consider $\mathbf{Q}_{n+1} =$

$$\begin{array}{l} \lambda f_{n+1}.\lambda f_{[n]}.\lambda x.\lambda y.(\lambda z.\mathbf{K} \\ \quad (\mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle z, \langle \mathtt{2nd}\ z, \mathtt{1st}\ z \rangle \rangle \\ \quad \mathit{else}\ \langle \langle \mathtt{2nd}\ z, \mathtt{1st}\ z \rangle, z \rangle) \\ \quad (f_{n+1} z)) \\ \quad (\mathbf{P}^n\, cond_{x,y}) \end{array}$$

By induction hypothesis for $\mathbf{Q}_n$, $(\mathbf{P}^n\, cond_{x,y})$ has a minimal principal typing with type $T_{n+1}^m$ (for some $m$) and coercion set $C_n$ with at least $2^{n+1}$ distinct variables, assuming the appropriate types $\sigma_i \to v_i$ for the $f_i$ $(i = 1 \ldots n)$, $\alpha$ for $x$ and $\beta$ for $y$. Now imagine that we have inserted coercions at the leaves in the remaining part of $\mathbf{Q}_{n+1}$ (cf. the *standard procedure*.) Then, due to the application $(f_{n+1} z)$, the type assumed for $f_{n+1}$ must have the form $\tau_1 * \tau_2 \to \gamma$ where $\tau_1 * \tau_2$ is a renaming of $T_{n+1}^m$ (using fresh variables, and $\gamma$ fresh), because the type of $z$ must be $T_{n+1}^m$ (because $(\mathbf{P}^n\, cond_{x,y})$ is applied to $\lambda z. \ldots$) However, since every variable in $\tau_1 * \tau_2$ occurs only positively in the type of the entire expression $\mathbf{Q}_{n+1}$, it easily follows by $S$-simplification that $\tau_1 * \tau_2$ can be identified with

using the constraints of Figure 5, represents a minimal, principal typing of $M$:

$$\begin{array}{rcl} M' & = & \lambda x : \alpha * \beta.\mathit{if}\ \mathit{true} \\ & & \mathit{then}\ \langle \uparrow^{\alpha_1 * \beta_1}_{\alpha * \beta} x, \langle \mathtt{2nd}\ \uparrow^{\alpha * \beta'_2}_{\alpha * \beta} x, \mathtt{1st}\ \uparrow^{\alpha'_3 * \beta}_{\alpha * \beta} x \rangle \rangle \\ & & \mathit{else}\ \langle \langle \mathtt{2nd}\ \uparrow^{\alpha * \alpha_1}_{\alpha * \beta} x, \mathtt{1st}\ \uparrow^{\beta_1 * \beta}_{\alpha * \beta} x \rangle, \uparrow^{\beta'_2 * \alpha'_3}_{\alpha * \beta} x \rangle \end{array}$$

Now, in analogy with the constraint set for $cond_{x,y}$ shown in Figure 4, we see that if a type of the form $\tau_1 * \tau_2$ must be assumed for the variable $x$ in $M$ (instead of $\alpha * \beta$), with $\tau_1, \tau_2$ recursively built by pairing of atoms, then the constraint set shown in Figure 5 with $\tau_1$ substituted for $\alpha$ and $\tau_2$ substituted for $\beta$ will be valid for the principal typing of $M$, and, moreover, in that coercion set each crown of Figure 5 will expand to $n_i$ new 2-crowns of observable atomic types once a matching substitution has been used and decomposition into atomic constraints has been performed, where $n_i$ measures the number of atoms occurring in $\tau_i$, $i \in \{1, 2\}$. If $\tau_1$ and $\tau_2$ are built by pairing from distinct variables, each with just a single occurrence in the type $\tau_1 * \tau_2$, the expanded 2-crowns will be $S$-simplified, and so it follows by the usual argument from $S$-normal form and Theorem 4.4 that the resulting, expanded coercion set will be valid for the *minimal* principal typing of $M$ in a context in which the type $\tau_1 * \tau_2$ must be assumed for the variable $x$ in $M$.

## 5.2 The construction

We proceed to give the construction. Let $cond_{x,y}$ denote the expression with two free variables $x$ and $y$, given by

$$cond_{x,y} = \mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle x, y \rangle\ \mathit{else}\ \langle y, x \rangle$$

For a term variable $f$, define the expression $\mathbf{P}_f$ with free variable $f$ as follows:

$$\begin{array}{rcl} \mathbf{P}_f & = & \lambda z.\mathbf{K} \\ & & (\mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle z, \langle \mathtt{2nd}\ z, \mathtt{1st}\ z \rangle \rangle \\ & & \mathit{else}\ \langle \langle \mathtt{2nd}\ z, \mathtt{1st}\ z \rangle, z \rangle) \\ & & (f\ z) \end{array}$$

where $\mathbf{K}$ is the combinator $\lambda x.\lambda y.x$. Let $f_1, f_2, \ldots$ be an enumeration of infinitely many distinct term variables, and let $N$ be any expression; then define, for $n \geq 0$, the expression $\mathbf{P}^n N$ recursively by setting

$$\begin{array}{rcl} \mathbf{P}^0 N & = & N, \\ \mathbf{P}^{n+1} N & = & \mathbf{P}_{f_{n+1}}(\mathbf{P}^n N) \end{array}$$

Write $\lambda f_{[n]}.M = \lambda f_n \ldots \lambda f_1.M$ for $n \geq 1$ and $\lambda f_{[0]}.M = M$. Now we can define the series of terms $\mathbf{Q}_n$ for $n \geq 0$ by setting

$$\mathbf{Q}_n = \lambda f_{[n]}.\lambda x.\lambda y.\mathbf{P}^n cond_{x,y}$$

So, for instance, we have

$$\mathbf{Q}_0 = \lambda x.\lambda y.cond_{x,y}$$

and

$$\begin{array}{rcl} \mathbf{Q}_1 & = & \lambda f_1.\lambda x.\lambda y.(\lambda z.\mathbf{K} \\ & & (\mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle z, \langle \mathtt{2nd}\ z, \mathtt{1st}\ z \rangle \rangle \\ & & \mathit{else}\ \langle \langle \mathtt{2nd}\ z, \mathtt{1st}\ z \rangle, z \rangle) \\ & & (f_1 z)) \\ & & cond_{x,y} \end{array}$$

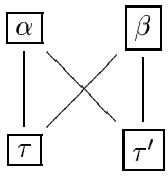



Figure 3: Minimal constraint set for $cond_{x,y}$

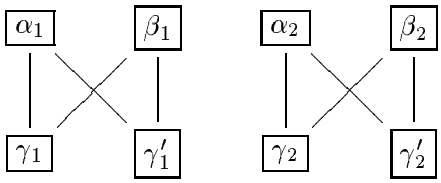



Figure 4: Constraint set of Figure 3 with $\tau = \gamma_1 * \gamma_2$, $\tau' = \gamma'_1 * \gamma'_2$ after matching expansion and decomposition

two 2-crowns shown in Figure 4 under the matching substitution which maps $\alpha$ to $\alpha_1 * \alpha_2$ and $\beta$ to $\beta_1 * \beta_2$ followed by decomposition into atomic constraints.

Now consider the term $M$ defined by

$$M = \lambda x.\mathit{if}\ \mathit{true}\ \mathit{then}\ \langle x, \langle \mathtt{2nd}\, x, \mathtt{1st}\, x\rangle\rangle\ \mathit{else}\ \langle\langle \mathtt{2nd}\, x, \mathtt{1st}\, x\rangle, x\rangle$$

It is easy to verify that the following completion represents a principal typing of $M$:

$$\begin{array}{l}\lambda x : \alpha * \beta.\mathit{if}\ \mathit{true}\\ \mathit{then}\ \langle \uparrow^{\alpha_1*\beta_1}_{\alpha*\beta} x, \langle \uparrow^{\beta'_2}_{\beta_2} (\mathtt{2nd}\ \uparrow^{\alpha_2*\beta_2}_{\alpha*\beta} x), \uparrow^{\alpha'_3}_{\alpha_3} (\mathtt{1st}\ \uparrow^{\alpha_3*\beta_3}_{\alpha*\beta} x)\rangle\rangle\\ \mathit{else}\ \langle\langle \uparrow^{\alpha_1}_{\alpha'_1} (\mathtt{2nd}\ \uparrow^{\alpha_4*\alpha'_1}_{\alpha*\beta} x), \uparrow^{\beta_1}_{\alpha_5} (\mathtt{1st}\ \uparrow^{\alpha_5*\beta_4}_{\alpha*\beta} x)\rangle, \uparrow^{\beta'_2*\alpha'_3}_{\alpha*\beta} x\rangle\end{array}$$

at the type $\alpha * \beta \to (\alpha_1 * \beta_1) * (\beta'_2 * \alpha'_3)$. The coercion set corresponding to the coercions shown in the term $G$-simplifies into the double crown shown in Figure 5, using substitutions which map $\alpha_2, \alpha_3, \alpha_4$ and $\alpha_5$ to $\alpha$, $\alpha'_1$ to $\alpha_1$ and $\beta_2, \beta_3, \beta_4$ to $\beta$. Verification of this is easy and left for the reader. Furthermore, it is immediate that the constraint set shown in Figure 5 is $S$-simplified, and since everything is observable, it follows from Theorem 4.4 that the completion $M'$ shown below,

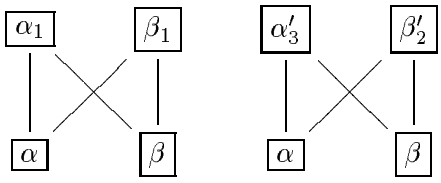



Figure 5: Constraint set in principal typing of $M$ after $G$-simplification

(1) through (3) have been performed, we can then apply transformations such as $G$-minimization and $S$-simplification to simplify the constraint set, without loosing principality. We shall sometimes indicate the form of a typing derivation by *completions* which are terms with explicit subtyping coercions and type assumptions for bound variables. A coercion from $\tau$ to $\tau'$ applied to a term $M$ will be written as $\uparrow_{\tau}^{\tau'} M$, so, for instance, the completion $\lambda x : \alpha.\, \uparrow_{\alpha}^{\beta} x$ encodes the typing judgement $\{\alpha \leq \beta\}, \emptyset \vdash_P \lambda x.x : \alpha \to \beta$ as well as a derivation of that judgement.

We know (see, e.g., [8]) that, in the simply typed lambda calculus without subtyping, we can create an exponential explosion in the *textual size* of the types of terms. The way to do this is quite obvious, since we can simply copy the argument of a function to the output, by using, say, the combinator $\mathbf{C} = \lambda z.\langle z, z\rangle$; if $\mathbf{C}^n M$ is the $n$-fold application of $\mathbf{C}$ to argument $M$, then (for $n \geq 1$) the type of $\mathbf{C}^n M$ will be the product type $\tau * \ldots * \tau$ with $2^n$ occurrences of $\tau$, assuming that $M$ has type $\tau$. We see, though, that while textual type size can thus become exponentially dependent upon the size of terms, *dag* size is still linear, because the type consists of repetitions of the same type, so sharing introduces exponential succinctness in the dag-representation of the type. The proof that every principal typing in simply typed lambda calculus produces types of size linearly dependent on the size of terms is just the standard unification based implementation of simply typed lambda calculus, reflecting the fact that all type constraints are equational (see [20].) With subtyping, however, the matter gets more complicated, because subtyping is not an equational system, being based, as it is, on *in*equality constraints. In fact, it is easy (and left for the reader) to see that the trick used above for simple types will not yield anything new in the presence of subtyping, since $G$- and $S$-simplification will eliminate all variables from the constraint set, so that we get the principal typing $\emptyset, \emptyset \vdash_P \mathbf{C}^n M : \tau * \ldots * \tau$ with subtyping, assuming that $\tau$ is a principal type for $M$. Hence, we need some new ideas to prove exponential dag-size for subtyping.

In order to illustrate some of the principles that will be appealed to in the construction to be given later, we shall now consider some expressions and their principal typings. Similar expressions will be part of the construction to be given later. The first expression we shall consider is the expression $cond_{x,y}$ with two free variables $x$ and $y$ and defined by

$$cond_{x,y} = \mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle x, y\rangle\ \mathit{else}\ \langle y, x\rangle$$

Suppose this occurs in a context where the type $\tau$ has been assumed for $x$ and the type $\tau'$ has been assumed for $y$. In case $\tau$ and $\tau'$ are atoms, it is easy to see (using the *standard procedure*) that the following completion represents a principal typing for $cond_{x,y}$:

$$\mathit{if}\ \mathtt{true}\ \mathit{then}\ \langle \uparrow_{\tau}^{\alpha} x, \uparrow_{\tau'}^{\beta} y\rangle\ \mathit{else}\ \langle \uparrow_{\tau'}^{\alpha} y, \uparrow_{\tau}^{\beta} x\rangle$$

at type $\alpha * \beta$. This yields the atomic coercion set in Figure 3 (we show *observable variables* inside a box, as usual.) An interesting property of this constraint set is that it constitutes a *2-crown* of variables (recall the notion of 2-crowns from Section 4.1.) Now, 2-crowns are interesting, because any constraint set consisting of 2-crowns built from distinct observable types is easily seen to be $S$-simplified, and hence, by Theorem 4.4, any such set is minimal. In particular, the constraint set of Figure 3 is minimal with respect to the term $cond_{x,y}$ by this reasoning.

Suppose now that either of $\tau$ or $\tau'$ is not an atom but a type built from pairing and atoms. Then the constraint set in Figure 3 must be expanded into a matching set, where both $\tau$ and $\tau'$ match both $\alpha$ and $\beta$ (hence, in particular, $\tau$ and $\tau'$ must match each other.) This expansion composed with decomposition into atomic constraints will result in a number of 2-crowns of observable atoms, where the number of crowns equals the number of components in the pair-type $\tau$ (or, equivalently, in $\tau'$.) For instance, if $\tau = \gamma_1 * \gamma_2$ and $\tau' = \gamma_1' * \gamma_2'$, then the set of Figure 3 will expand into the

For $\lambda^{\rightarrow}$ we know (see [8]) that while *textual* type size can be exponential, *dag-size* is at most linear. The basic property responsible for this is that $\lambda^{\rightarrow}$ is a purely equational type system, in the sense of [20], so that enough sharing becomes possible. ML is not purely equational, and with its universal quantifier and its succinct `let`-expressions, we get doubly exponential textual size of types and exponential dag-size, in the worst case (see [8] with further references.) Subtyping is a system based on *in*equalities, and, as shown below, this leads to the impossibility of sharing, resulting in exponential dag-size of typings (coercion sets as well as of types) due to the fact that exponentially many distinct variables may have to be present in a principal typing. Among other things, this result shows an intrinsic limit as to how much type-simplification techniques can possibly achieve for atomic subtyping, at least with instance relations not stronger than $\prec$.

The exponential lower bound proof has two core parts. One is the construction of a series of terms $\mathbf{Q}_n$, with the intention that, for all $n \geq 0$, any principal typing of $\mathbf{Q}_n$ has the form $C_n, \emptyset \vdash_P \mathbf{Q}_n : \tau_n$, where $C_n$ contains more than $2^n$ distinct type variables and $\tau_n$ contains more than $2^n$ distinct type variables. The second main ingredient in the proof is Theorem 4.4 and the characterization of minimal typings of Section 3, which are employed in order to prove that the intended property in fact holds for the $\mathbf{Q}_n$, viz. that *any* principal typing of the terms $\mathbf{Q}_n$ must have a number of variables in both coercion set and type which is exponentially dependent on the size of the terms.

### 5.1 Preliminaries to the construction

For the purpose of the following development we shall first assume that we have a conditional construct *if* ... *then* ... *else* ... with the typing rule

$$[\mathit{if}] \quad \frac{C, \Gamma \vdash_P M : \mathtt{bool} \quad C, \Gamma \vdash_P N : \tau \quad C, \Gamma \vdash_P Q : \tau}{C, \Gamma \vdash_P \mathit{if}\ M\ \mathit{then}\ N\ \mathit{else}\ Q : \tau}$$

Moreover, we shall assume pairs $\langle M, N \rangle$ and pair-types $\tau * \tau'$ with the usual typing rule

$$[\mathit{pair}] \quad \frac{C, \Gamma \vdash_P M_1 : \tau_1 \quad C, \Gamma \vdash_P M_2 : \tau_2}{C, \Gamma \vdash_P \langle M_1, M_2 \rangle : \tau_1 * \tau_2}$$

together with projections `1st` and `2nd` using the standard typing rules

$$[\mathit{proj1}] \quad \frac{C, \Gamma \vdash_P M : \tau_1 * \tau_2}{C, \Gamma \vdash_P (\mathtt{1st}\ M) : \tau_1} \qquad [\mathit{proj2}] \quad \frac{C, \Gamma \vdash_P M : \tau_1 * \tau_2}{C, \Gamma \vdash_P (\mathtt{2nd}\ M) : \tau_2}$$

The subtype ordering is lifted co-variantly to pairs in the usual way, such that $\tau_1 * \tau_2 \leq \tau_1' * \tau_2'$ holds if and only if $\tau_1 \leq \tau_1'$ and $\tau_2 \leq \tau_2'$ both hold.

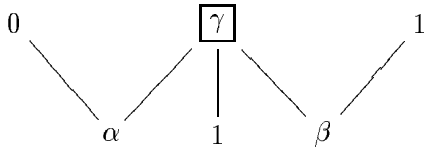
These additional constructs are introduced here only because it is easier to understand the essence of the constructions to be given below using these constructs. Once we have completed the constructions, we shall show how to encode them in the "pure" lambda calculus with no additional constructs.

In the sequel, we shall often carefully consider the form of principal typings of terms. We use the fact, shown in [12], that a principal typing can always be obtained by the following *standard procedure*: first (1) extract subtyping constraints only at the *leaves* of the term (i.e., coercions are applied to variables and constants only), and then (2) perform a *match-step* in which a most general matching substitution (see [12] for details) is applied to the extracted constraint set, and finally (3) *decompose* the matching constraints into atomic constraints (any matching set can be decomposed, and if the match-step fails, then the term has no typing with atomic subtyping at all.) Once steps

### 4.3 Incompleteness of the Fuh-Mishra transformations

Proposition 4.3 naturally leads to the question whether *G-minimization* together with $S$-simplification is complete for minimization, i.e., is it the case that a typing, which is both $G$-minimized and $S$-simplified, is also minimal in the sense of Definition 3.2? However, as the following example shows, this is *not* the case in *any* non-trivial partial order (we consider a poset trivial here, if it is discrete.)

**Example 4.5** (Incompleteness of $G$-minimization and $S$-simplification)
*Let $P$ be any non-trivial poset. Then there are two distinct elements $0, 1 \in P$ with $0 < 1$. Let $C$ be the constraint set over $P$:*




*Here $\gamma$ is observable and $\alpha, \beta$ are internal. Now, it is easy (and left for the reader) to verify that $C$ is indeed $G$-*minimal*, since if $S$ is any substitution on the internal variables of $C$, i.e., with $Supp(S) \subseteq \{\alpha, \beta\}$, and $S$ is not the identity on $FTV(C)$, then $S$ cannot satisfy $C \vdash_P S(C)$; moreover, it is easy to verify that $C$ is also $S$-simplified, since for no $A \neq \gamma$ do we have $\gamma \leq_S A$. However, with $S_{min} = \{\alpha \mapsto 0, \beta \mapsto 0, \gamma \mapsto 1\}$ we do have [4] $C \vdash_P S_{min}(C)$, because $S_{min}(C) = \{0, 1\}$ Hence, any typing $t$ of the form $t = C, \emptyset \vdash_P M : \tau$ with $FTV(\tau) = \{\gamma\}$ and $\gamma$ occurring positively (and not negatively) in $\tau$, cannot be fully substituted and hence it cannot be minimal, since $S_{min}(t) \prec t$.* □

One lesson we can learn from Example 4.5 is that in minimization one cannot treat optimization of internals and observables seperately without loosing completeness. It may be instructive to see that we can easily realize, in a simple program, the situation described by the example. Assume we have typed functional constants `succ` $: \texttt{int} \to \texttt{int}$ and `sqrt` $: \texttt{real} \to \texttt{real}$. Let $M$ be the term of the following typing (the application of subtyping coercions are shown explicitly; a coercion from $\tau$ to $\tau'$ applied to a term $M$ is written as $\uparrow_\tau^{\tau'} M$)

$$\begin{array}{lll} \lambda f^{\gamma \to \delta}. \ \texttt{\#4} & \langle \lambda x^{\alpha}.\langle f\ (\uparrow_\alpha^\gamma x), \texttt{succ}\ & (\uparrow_\alpha^{\texttt{int}} x)\rangle, \\ & \lambda y^{\beta}.\langle f\ (\uparrow_\beta^\gamma y), \texttt{sqrt}\ & (\uparrow_\beta^{\texttt{real}} y)\rangle, \\ & & f\ (\uparrow_{\texttt{int}}^\gamma 1), \\ & & 1\rangle \end{array}$$

Then we have $C, \emptyset \vdash_P M : (\gamma \to \delta) \to \texttt{int}$ with $C$ as in Example 4.5, taking $0 = \texttt{int}$ and $1 = \texttt{real}$ .

## 5 The size of principal typings

We prove a tight worst case exponential lower bound for the dag-size of coercion sets as well as of types in principal typings. To the best of our knowledge, the best lower bound previously proven is the linear lower bound for a generic instance relation shown in [5]. The basic idea of enforcing a blow-up in textual type size described for simply typed lambda calculus without subtyping ($\lambda^{\to}$) in [8] is present in our proof. However, the matter is much more complicated in subtyping, and our proof uses new techniques; moreover, the results of Section 3 as well as Theorem 4.4 are important ingredients.

[4] We could also take $\beta \mapsto 1$.

By the Substitution Lemma, we then have $S(C) \vdash_P S(A_1) \leq S(\alpha)$, and so, by (1), we also have

$$C \vdash_P S(A_1) \leq S(\alpha) \tag{35}$$

If $A_1 = S(A_1)$, then by (35) it would follow that $C \vdash_P A_1 \leq S(\alpha)$, hence $A_1 \in{\downarrow_C}(S(\alpha))$ in contradiction with (33). Therefore we must have

$$A_1 \neq S(A_1) \tag{36}$$

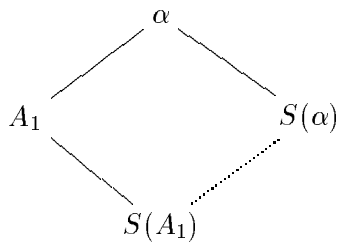
from which it follows that

$$A_1 \text{ is a variable in } \mathrm{Supp}(S) \tag{37}$$

Now, by (37) and assumption (3), $A_1$ is an observable variable, occurring in $\tau'$. Then (2) implies that $A_1$ and $S(A_1)$ must be comparable under the hypotheses $C$. We already know (36) that $A_1 \neq S(A_1)$, and if $C \vdash_P A_1 \leq S(A_1)$, then it would follow by (35) that $C \vdash_P A_1 \leq S(\alpha)$ in contradiction with (33). We must therefore conclude that

$$C \vdash_P S(A_1) \leq A_1 \text{ with no negative occurrence of } A_1 \text{ in } \tau' \tag{38}$$

Summing up so far, we now have the situation




where the dotted line means "less than or equal to" and the full lines mean "strictly less than" with respect to $C$. But this, together with (38) shows that the situation described in $(*)$ now holds with $A_1$ and $S(A_1)$ in place of $\alpha$ and $S(\alpha)$, and all the reasoning starting from $(*)$ can therefore be repeated for $A_1$ and $S(A_1)$, leading, in particular, to the existence of an atom $A_2$ with $C \vdash_P A_2 \leq A_1$ and $A_2 \neq A_1$ etc. Hence, by repetition of the argument starting at $(*)$, we obtain an arbitrarily long strictly decreasing chain (with respect to $C$)

$$\alpha > A_1 > A_2 > \ldots$$

implying the existence of a proper cycle in $C \cup P$ and hence contradicting (4). We must therefore reject the assumption (32), thereby proving the Proposition. □

Proposition 4.3 implies that, whenever $S(t) \prec t$ with $S$ not renaming on $t$ and $\mathrm{Supp}(S) \subseteq \mathrm{Obv}(t)$, then $t$ can be $S$-simplified. Hence, for a typing $t$ which is normalized under $S$-simplification there exists no strict substitution instance $S(t) \approx t$ such that $S$ affects only observable variables in $t$. Consequently, if $t = C, \Gamma \vdash_P M : \tau$ with $FTV(C) \subseteq \mathrm{Obv}(t)$, and if $t$ is $S$-simplified, then $t$ must be fully substituted; by Theorem 3.19 we therefore have

**Theorem 4.4** *If $t = C, \Gamma \vdash_P M : \tau$ is an $S$-simplified typing with $FTV(C) \subseteq Obv(t)$, then the typing $(C^\circ)^-, \Gamma \vdash_P M : \tau$ is minimal.*

Theorem 4.4 has a number of interesting applications. An important example occurs in Section 5 below, where we prove an exponential lower bound for the size of coercion sets in principal typings.

typings: Let $t_1 = C_1, \Gamma_1 \vdash_P M : \tau_1$ and $t_2 = C_2, \Gamma_2 \vdash_P M : \tau_2$; then

$$t_1 \mapsto_S t_2 \text{ iff } \begin{cases} (1) & \alpha \leq_S A \text{ wrt } C_1 \text{ and type\_closure}(t_1) \\ (2) & C_2 = C_1\{\alpha \mapsto A\} \\ (3) & \Gamma_2 = \Gamma_1\{\alpha \mapsto A\} \\ (4) & \tau_2 = \tau_1\{\alpha \mapsto A\} \\ (5) & \{\alpha \mapsto A\} \text{ is not a renaming on } t_1 \end{cases}$$

The transformation $\mapsto_S$ is a *simplification* like $\mapsto_G$, in that a single step of $\mapsto_S$ can only transform a typing by a simple substitution. However, there is a difference between $S$- and $G$-simplification in that $S$-simplification is designed to simplify a typing with respect to the *observable* variables, and therefore a substitution must satisfy more conditions to be applicable. In general, a transformation $t \mapsto t'$ must be *sound* in the sense that $t \approx t'$, and so, in order for a substitution $S$ to be a candidate for a sound transformation of a typing $t = C, \Gamma \vdash_P M : \tau$, we must require that $S(t) \prec t$. This implies in particular that $C \vdash_P S(\tau) \leq \tau$ (and a similar condition must hold for $\Gamma(x)$ and $S(\Gamma(x))$), hence if $\alpha \in \mathrm{Supp}(S) \cap \mathrm{Obv}(t)$, then $\alpha$ must be comparable to $S(\alpha)$ under $C$. This is a quite strong condition, and one may well ask whether $S$-simplification captures all sound transformations based on substituting into observable variables. In fact, $S$-simplification has an interesting completeness property which is established in the following proposition:

**Proposition 4.3** *(Completeness of $S$-simplification wrt. observable variables)*
*Let $t = C, \Gamma \vdash_P M : \tau$ and $\tau' = \mathit{type\_closure}(t)$. Assume $S$ is not a renaming on $FTV(C)$ with*

(1) $C \vdash_P S(C)$

(2) $C \vdash_P S(\tau') \leq \tau'$

(3) $\mathit{Supp}(S) \subseteq \mathit{Obv}(t)$

(4) *$C$ is acyclic*

*Then there exists $\alpha \in \mathit{Supp}(S)$ such that $\alpha \leq_S S(\alpha)$ with respect to $C$ and $\tau'$.*

*Proof*: The proof is by contradiction, so suppose, under the assumptions of the proposition, that

$$\text{there is no } \alpha \in \mathrm{Supp}(S) \text{ such that } \alpha \leq_S S(\alpha) \text{ with respect to } C \text{ and } \tau' \tag{32}$$

Pick any $\alpha \in \mathrm{Supp}(S)$ (by assumption $\mathrm{Supp}(S) \neq \emptyset$), so we have $\alpha \neq S(\alpha)$. By (2) and (3), $\alpha$ must be comparable to $S(\alpha)$ under $C$. Using these facts, it is easy to verify (by induction on $\tau'$, using (4) acyclicity of $C$) that (2) implies either $C \vdash_P S(\alpha) \leq \alpha$ with $\alpha$ not occurring negatively in $\tau'$, or else $C \vdash_P \alpha \leq S(\alpha)$ with $\alpha$ not occurring positively in $\tau'$. Assume that we have (the alternative case is similar)

$$(*) \qquad C \vdash_P S(\alpha) \leq \alpha \text{ with } \alpha \text{ not occurring negatively in } \tau'$$

By (32) we must have $\downarrow_C(\alpha) \setminus \{\alpha\} \not\subseteq \downarrow_C(S(\alpha))$, since otherwise we should have $\alpha \leq_S S(\alpha)$. So there must be an atomic type $A_1 \in \downarrow_C(\alpha) \setminus \{\alpha\}$ such that

$$A_1 \notin \downarrow_C(S(\alpha)) \tag{33}$$

In particular, we therefore have

$$A_1 \neq S(\alpha), A_1 \neq \alpha \text{ and } C \vdash_P A_1 \leq \alpha \tag{34}$$

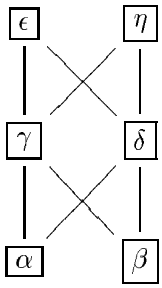



Figure 2: Constraint set $D$ with all variables observable

Clearly, the theorem will follow from $(*)$ by **NP**-completeness of the satisfiability problem for $K_2$, since the condition $D \vdash_\emptyset \widetilde{C}$ can be checked in polynomial time.

To prove $(*)$, assume first that $C$ is satisfiable over $K_2$. Then $\vdash_{K_2} \rho(C)$ for some valuation $\rho$ in $K_2$, hence $D \vdash_\emptyset (\pi \circ \rho)(C)$, because the lower part of $D$ is isomorphic to $K_2$ under $\pi$. Now, $(\pi \circ \rho)(C) = (\pi \circ \rho)(\pi(C))$, because $\rho$ is the identity on the domain and range of $\pi$. Furthermore, since $\pi \circ \rho$ is the identity on $D$ and $(\pi \circ \rho)(FTV(C)) \subseteq \{\alpha, \beta, \gamma, \delta\}$, we have $D \vdash_\emptyset (\pi \circ \rho)(\alpha') \leq (\pi \circ \rho)(\upsilon)$ for $\upsilon \in \{\epsilon, \eta\}$. It follows from these observations that $D \vdash_\emptyset (\pi \circ \rho)(\overline{C})$, which entails in particular that $\overline{C} \vdash_\emptyset (\pi \circ \rho)(\overline{C})$, so we have $\overline{C} \stackrel{\pi \circ \rho}{\longmapsto}_{G^+} (\pi \circ \rho)(\overline{C})$. Since $D \vdash_\emptyset (\pi \circ \rho)(\overline{C})$, it is easy to see that $(\pi \circ \rho)(\overline{C})$ must be in $G^+$-normal form, because all variables in $D$ are observable. By uniqueness of $G^+$-normal forms it follows that $\widetilde{C} \sim_\emptyset R((\pi \circ \rho)(\overline{C}))$ with $R$ a renaming on $FTV((\pi \circ \rho)(\overline{C}))$ and the identity on the observables in $(\pi \circ \rho)(\overline{C})$. Since $\rho$ maps $C$ to $K_2$, we can assume w.l.o.g. that all variables in $(\pi \circ \rho)(\overline{C})$ are contained in $FTV(D)$, and it then follows that $R$ is the identity on $(\pi \circ \rho)(\overline{C})$, and so we have $\widetilde{C} \sim_\emptyset (\pi \circ \rho)(C)$, and hence (by $D \vdash_\emptyset (\pi \circ \rho)(C)$) we have $D \vdash_\emptyset \widetilde{C}$, as desired.

For the converse implication, assume $D \vdash_\emptyset \widetilde{C}$. Then there is a substitution $S$ such that $S$ $G$-minimizes $\overline{C}$, with $\overline{C} \vdash_P S(\overline{C}) = \widetilde{C}$. Since all variables in $\overline{C}$ are required to be below the observable variables $\epsilon$ and $\eta$, it follows from $D \vdash_\emptyset \widetilde{C}$ that $S$ must map all the variables in $C$ to the lower part of $D$, i.e., $S(FTV(C)) \subseteq \{\alpha, \beta, \gamma, \delta\}$. But this shows that $\vdash_{K_2} (\pi^{-1} \circ S)(C)$, and $C$ is satisfiable in $K_2$. This proves $(*)$ and thereby the theorem. □

Theorem 4.2 generalizes to minimization (i.e., computing minimal typings in the sense of Definition 3.2) because we can always create situations in which no observable variables can be substituted in any substitution instance $S(t)$ with $S(t) \prec t$, by having both positive and negative occurrences of the observables in the type of $t$. For such typings minimization degenerates to $G$-minimization.

## 4.2 $S$-simplification

Given atomic coercion set $C$, type $\tau$, variable $\alpha$ and atom $A$, we say that $\alpha$ is *$S$-subsumed* by $A$ in $C$ and $\tau$, written $\alpha \leq_S A$, iff

1. *either* $C \vdash_P A \leq \alpha$ and $\alpha$ does not occur negatively in $\tau$ and $\downarrow_C(\alpha) \setminus \{\alpha\} \subseteq \downarrow_C(A)$
2. *or* $C \vdash_P \alpha \leq A$ and $\alpha$ does not occur positively in $\tau$ and $\uparrow_C(\alpha) \setminus \{\alpha\} \subseteq \uparrow_C(A)$

If $t = C, \Gamma \vdash M : \tau$, with $dom(\Gamma) = \{x_1, \ldots, x_n\}$, then we define the type type_closure$(t) = \Gamma(x_1) \to \ldots \to \Gamma(x_n) \to \tau$. We then define, following Fuh and Mishra [3], the reduction $\mapsto_S$ on

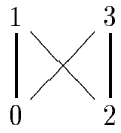
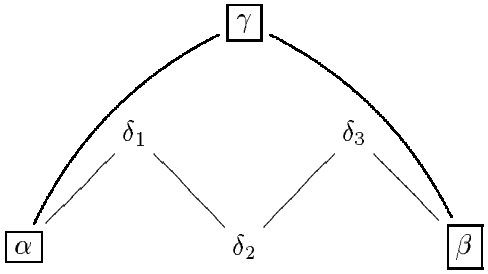



Figure 1: A 2-crown

*Consider coercion set $C$ below, where observable variables are shown inside a box:*




*It can be seen that $C$ is* $G$-simplified *(i.e., in normal form with respect to $\mapsto_G$), because it is not possible to find a simple, non-identity substitution $S$ of the form $\{\delta_i \mapsto A\}$, $A \in \{\alpha, \beta, \gamma, \delta_1, \delta_2, \delta_3\}$, such that $C \vdash_P S(C)$. However, the set $C$ is* not $G$-minimal, *because the substitution*

$$S_{min} = \{\delta_1 \mapsto \gamma, \delta_2 \mapsto \gamma, \delta_3 \mapsto \gamma\}$$

*satisfies $C \vdash_P S_{min}(C)$. Note that $S_{min}$ can be factored into a sequence of simple substitutions, as $S_{min} = \{\delta_1 \mapsto \gamma\} \circ \{\delta_2 \mapsto \gamma\} \circ \{\delta_3 \mapsto \gamma\}$, but there is no simple non-identity substitution $S_{simple}$ which satisfies $C \vdash_P S_{simple}(C)$.* □

The following theorem strengthens the remarks in [3] on the complexity of $G$-minimization. To prepare for the proof, recall from [14] (see also [9] and [17]) that the problem of satisfiability of atomic inequalities of the form $A \leq A'$ is **NP**-complete for the fixed partial order $K_2$, called *2-crown* in [14], consisting of 4 elements $0, 1, 2, 3$ ordered by $0 < 1, 0 < 3, 2 < 1, 2 < 3$, as shown in Figure 1.

**Theorem 4.2** *If* $\mathbf{P} \neq \mathbf{NP}$, *then $G$-minimization cannot be computed in polynomial time for any partial order $P$.*

*Proof*: Fix the constraint set $D$ shown in Figure 2 with variables $\alpha, \beta, \gamma, \delta, \epsilon, \eta$, all designated as observable. Note that $D$ can be regarded as two copies (in variables) of $K_2$ which are "spliced together" at $\gamma$ and $\delta$, and, in particular, the lower part $\{\alpha, \beta, \gamma, \delta\}$ of $D$ is isomorphic to $K_2$ under the embedding $\pi = \{0 \mapsto \alpha, 1 \mapsto \gamma, 2 \mapsto \beta, 3 \mapsto \delta\}$.

Now let $C$ be any atomic constraint set over $K_2$, and assume w.l.o.g. that $FTV(C) \cap FTV(D) = \emptyset$. We also assume w.l.o.g. that all constraint sets considered are irreflexive (i.e., they have no loops.) Translate $C$ to $\overline{C}$ with

$$\overline{C} = D \cup \pi(C) \cup \{\alpha' \leq \epsilon, \alpha' \leq \eta \mid \alpha' \in FTV(C)\}$$

with all $\alpha' \in FTV(C)$ designated as internal variables in $\overline{C}$. Let $\widetilde{C}$ be a $G^+$-normal form of $\overline{C}$. We *claim* that

$(*)$ $\quad C$ is satisfiable over $K_2$ if and only if $D \vdash_\emptyset \widetilde{C}$

### 4.1 $G$-simplification and $G$-minimization

To recall from [3] the definition of $G$-simplification, let $\uparrow_C(A) = \{A' \mid C \vdash_P A \leq A'\}$ and $\downarrow_C(A) = \{A' \mid C \vdash_P A' \leq A\}$; given variable $\alpha \in FTV(C)$, we say that $\alpha$ is *$G$-subsumed* by an atom $A$ with respect to $C$, written $\alpha \leq_G A$, iff $\uparrow_C(\alpha) \setminus \{\alpha\} \subseteq \uparrow_C(A)$ and $\downarrow_C(\alpha) \setminus \{\alpha\} \subseteq \downarrow_C(A)$. The transformation $\mapsto_G$ on typings is then defined as follows. Let $t_1 = C_1, \Gamma_1 \vdash_P M : \tau_1$ and $t_2 = C_2, \Gamma_2 \vdash_P M : \tau_2$. Then $t_1 \mapsto_G t_2$ iff $C_2 = C_1\{\alpha \mapsto A\}$ where $\alpha \in \mathrm{Intv}(C_1)$, $\alpha \neq A$ and $\alpha \leq_G A$ with respect to $C_1$. Note that $\Gamma_1 = \Gamma_2$ and $\tau_1 = \tau_2$, whenever $t_1 \mapsto_G t_2$. If $t_1 \mapsto_G t_2$, then $t_1 \approx t_2$, as shown in [3].

$G$-simplification can be regarded as an efficiently computable [3] approximation to a more powerful transformation $\mapsto_{G^+}$ called *$G$-minimization*, which can be defined as follows. Given typing $t = C, \Gamma \vdash_P M : \tau$, let $\mathrm{Subs}(t)$ be the set of substitutions $S$ such that

$$S(\alpha) = \begin{cases} \alpha & \text{if } \alpha \notin \mathrm{Intv}(t) \\ \text{a constant, or a variable in } FTV(C) & \text{if } \alpha \in \mathrm{Intv}(t) \end{cases}$$

Then $t_1 \mapsto_{G^+} t_2$ iff there exists $S \in \mathrm{Subs}(t_1)$ such that $C_2 = S(C_1)$, $S$ is not a renaming on $FTV(C_1)$ and $C_1 \vdash_P S(C_1)$.

Observe that both $\mapsto_G$ and $\mapsto_{G^+}$ are terminating, since the size of $\mathrm{Intv}(t)$ shrinks at every reduction step. Normalization with respect to $G^+$ is accomplished by the procedure MINIMIZE in [3]. $G$-minimization is unique up to equivalence of coercion sets and renaming in the sense that, if $C_1 = S_1(C)$ and $C_2 = S_2(C)$ are both $G^+$-normal forms of $C$, then there is a renaming $R$ on $FTV(C_2)$ such that $R$ is the identity on $\mathrm{Obv}(C_2) = \mathrm{Obv}(C_1) = \mathrm{Obv}(C)$ and $C_1 \sim_P R(C_2)$. This follows from Lemma 3.13, because by the assumptions we have $C \vdash_P C_1$ and $C \vdash_P C_2$, hence (by Substitution Lemma) $C_1 \vdash_P S_1(C_2)$ and $C_2 \vdash_P S_2(C_1)$, and so, since $C_1, C_2$ are $G^+$-normal forms, $S_2$ is a renaming on $FTV(C_1)$ and $S_1$ is a renaming on $FTV(C_2)$, and so Lemma 3.13 yields $C_1 \sim_P S_1(C_2)$. We can take $R = S_1$, because $S_1$ is the identity on $\mathrm{Obv}(C) = \mathrm{Obv}(C_2)$ by definition of $G^+$-normalization.

Note also that, as is the case for $\mapsto_G$, the transformation $\mapsto_{G^+}$ only changes internal variables of a typing. However, there is a crucial difference between $\mapsto_G$ and $\mapsto_{G^+}$. To see this, call a substitution $S$ *simple* if $\mathrm{Supp}(S)$ is a singleton set. Then the substitution $S$ applied in a single step of $\mapsto_G$ must be simple, whereas a single $\mapsto_{G^+}$-step need not be restricted in this way. As we shall see, this explains why $G$ is but an approximation of $G^+$ and it has negative consequences for the complexity of $G$-minimization. Fuh and Mishra [3] remark that finding a $G^+$-normal form of a typing $t$ appears to require exhaustive search through $\mathrm{Subst}(t)$ in the worst case, which, if true, would mean that $G$-minimization is exponential in the size of $C \cup P$. In fact, we can show that $G$-minimization is **NP**-complete *independently of the poset $P$*. Before showing this, let us first give an example which illustrates why $G$-simplification is indeed only an approximation to $G$-minimization.

**Example 4.1** (Incompleteness of $G$-simplification wrt. $G$-minimization)

[3] In time $\mathcal{O}(n^4)$, where $n$ is the size of $C \cup P$, as shown in [3]; in the analysis of [3], their algorithm runs in time $\mathcal{O}(n^3)$, but this is assuming a bitvector representation of constraints with constant time comparison, which is, however, not a legitimate assumption for the assessment of asymptotic complexity.

because $t_1$ is assumed to be fully substituted. Now, (24), (25), (28), (29) and (30) allow us to apply Lemma 3.17 to the substitution $S_2 \circ S_1$ on $C_1$, and the Lemma shows that we must have $S_2(S_1(\alpha)) = \alpha$. This contradicts (27), and so we must reject the assumption (26), and $(*)$ is thereby established.

Now, by $(*)$ together with (22) we get that

$$C_1 \vdash_P \tau_1 \leq S_2(\tau_2) \tag{31}$$

We know (30) that $C_1$ is acyclic. But then (19) and (31) show (using Lemma 3.15 and $C_1$ acyclic) that $\tau_1 = S_2(\tau_2)$, as desired. We have now shown property (2) of the proposition. Property (3) follows by the same reasoning. □

We are finally ready to prove that minimal typings exist:

**Theorem 3.19** *(Existence of minimal typings)*
*For every atomic, consistent typing judgement $t$ the typing judgement $\tilde{t}$ is a minimal judgement in $[t]$, when $\tilde{t}$ is defined by*

$$\tilde{t} = ((S(C))^\circ)^-, S(\Gamma) \vdash_P M : S(\tau)$$

*with $t = C, \Gamma \vdash_P M : \tau$ and $S(t)$ a full (atomic, consistent) substitution instance of $t$ within $[t]$.*

*Proof*: Clearly, $\tilde{t}$ is fully substituted, and moreover $\tilde{t} \approx t$ because $((S(C))^\circ)^- \sim_P S(C)$ by consistency of $S(C)$. Now let $t'$ be an arbitrary atomic, consistent judgement in $[t]$, and write $t' = C', \Gamma' \vdash_P M : \tau'$. We must show that $\tilde{t} \lesssim t'$. Let $S'(t')$ be a full (atomic, consistent) substitution instance of $t'$ within $[t'] = [t]$. Then $S(t) \approx S'(t')$, and hence (by Lemma 3.8 and Proposition 3.18) there exists a renaming $R$ on $S'(t')$ such that

(1) $S(C) \sim_P R(S'(C'))$

(2) $S(\tau) = R(S'(\tau'))$

(3) $Dm(\Gamma) = Dm(\Gamma')$ and $\forall x \in Dm(\Gamma).\ S(\Gamma(x)) = R(S'(\Gamma'(x)))$

Since $S(C)$ must be acyclic (by Lemma 3.16) and $S(C)$ is consistent, we have $((S(C))^\circ)^- \subseteq R(S'(C'))$. We have shown $\tilde{t} \lesssim_{R \circ S'} t'$, thereby proving the theorem. □

# 4 Minimization

Since we now have an independent notion of minimal typings, we can meaningfully discuss completeness properties of transformations which are aimed at minimizing typings. We consider transformations defined by Fuh and Mishra in [3], and we also give some new results on the complexity of minimization.

In [3], Fuh and Mishra describe a number of transformations on typings with the purpose of minimizing them. First, they assume that cycle- and loop elimination has been performed, so that coercion sets can always be taken to be acyclic and irreflexive before any other transformation is applied. The core part of [3] is preoccupied with a study of two other transformations, called *G-simplification* and *S-simplification*.

Given typing $t = C, \Gamma \vdash_P M : \tau$, let the *observable types* in $t$, denoted $\mathrm{Obv}(t)$, be the constants in $P$ and type variables appearing free in $\Gamma$ or $\tau$, i.e., $\mathrm{Obv}(t) = FTV(\Gamma) \cup FTV(\tau) \cup P$. Let the *internal variables* in $t$, denoted $\mathrm{Intv}(t)$, be the set $\mathrm{Intv}(t) = FTV(C) \setminus \mathrm{Obv}(t)$. Then the $G$-simplification changes only the internal variables of a typing, whereas the $S$-simplification changes only the observable variables of a typing.

and

$$C_2 \vdash_P S_1(C_1) \tag{18}$$

Part (2) and (3) are similar, so we consider only part (2).

To see that (2) holds with $S = S_2$, we first record that, by our assumptions, we have

$$C_1 \vdash_P S_2(\tau_2) \leq \tau_1 \tag{19}$$

and

$$C_2 \vdash_P S_1(\tau_1) \leq \tau_2 \tag{20}$$

By (20) and the Substitution Lemma we get

$$S_2(C_2) \vdash_P S_2(S_1(\tau_1)) \leq S_2(\tau_2) \tag{21}$$

and hence, by (17) and (21)

$$C_1 \vdash_P S_2(S_1(\tau_1)) \leq S_2(\tau_2) \tag{22}$$

and therefore, by (19) and (22)

$$C_1 \vdash_P S_2(S_1(\tau_1)) \leq \tau_1 \tag{23}$$

Now, we know from Lemma 3.7 that $S_2(S_1(t_1)) \prec_{id} t_1$, hence we have $S_2(S_1(t_1)) \approx t_1$. It therefore follows from the assumption that $t_1$ is fully susbtituted that

$$S_2 \circ S_1 \text{ is a renaming on } FTV(t_1) \tag{24}$$

Moreover, by $S_2(S_1(t_1)) \prec_{id} t_1$, we also have

$$C_1 \vdash_P S_2(S_1(C_1)) \tag{25}$$

We now *claim* that in fact we have

$$(*) \quad S_2(S_1(\tau_1)) = \tau_1$$

To see that $(*)$ is true, assume that

$$S_2(S_1(\tau_1)) \neq \tau_1 \tag{26}$$

Then there is a variable occurrence $\alpha$ in $\tau_1$ such that

$$(S_2 \circ S_1)(\alpha) \neq \alpha \tag{27}$$

Let $(S_2 \circ S_1)(\alpha) = A$. Then, because of (23), the Decomposition Lemma entails that $\alpha$ and $A$ must be comparable under $C_1$, and hence

$$\text{either } C_1 \vdash_P S_2(S_1(\alpha)) \leq \alpha \text{ or } C_1 \vdash_P \alpha \leq S_2(S_1(\alpha)) \tag{28}$$

Since $\alpha$ is a variable and $A \neq \alpha$, it must be the case that

$$\alpha \in FTV(C_1) \tag{29}$$

since otherwise $\alpha$ and $A$ could not be comparable under $C_1$. Finally, by Lemma 3.16, we know that

$$C_1 \text{ is acyclic} \tag{30}$$

$(ii)$ $S^n(\alpha) \neq S^{n-1}(\alpha)$

$(iii)$ $C \vdash_P S^n(\alpha) \leq S^{n-1}(\alpha)$

To prove the *claim*, we prove all three properties simultaneously by induction on $n > 0$: For $n = 1$, the set $D_1 = \{\alpha, S(\alpha)\}$, so to establish $(i)$ we need only to show that $S(\alpha) \in FTV(C)$. By assumption we have $C \vdash_P S(\alpha) \leq \alpha$ and $S(\alpha) \neq \alpha$. Since $S$ is a renaming on $FTV(C)$, it follows that $S(\alpha)$ is a variable, which is distinct from $\alpha$. But two distinct variables, $\alpha$ and $S(\alpha)$, can only be comparable under $C$ if they are both mentioned in $C$, so in particular we must have $S(\alpha) \in FTV(C)$. To see $(ii)$ for $n = 1$, we need only show that $S(\alpha) \neq \alpha$, which holds by assumption. To see $(iii)$ for $n = 1$, we need only show that $C \vdash_P S(\alpha) \leq \alpha$, which holds by assumption. We have now shown the *claim* in the base case where $n = 1$.

For the inductive step, assume $n > 1$. We have by induction hypothesis that the set $D_{n-1} \subseteq FTV(C)$, hence, in particular, $S^{n-1}(\alpha), S^{n-2}(\alpha) \in FTV(C)$. By induction hypothesis applied to $(iii)$, we have $C \vdash_P S^{n-1}(\alpha) \leq S^{n-2}(\alpha)$, hence we get by the Substitution Lemma that $S(C) \vdash_P S^n(\alpha) \leq S^{n-1}(\alpha)$ and so, by (2), $C \vdash_P S^n(\alpha) \leq S^{n-1}(\alpha)$. This shows that $(iii)$ holds in the inductive case. Further, by induction hypothesis applied to $(ii)$, we have $S^{n-1}(\alpha) \neq S^{n-2}(\alpha)$, and so, since $S$ is injective as renaming on $FTV(C)$ with $S^{n-1}(\alpha), S^{n-2}(\alpha) \in FTV(C)$, we have $S^n(\alpha) \neq S^{n-1}(\alpha)$, which establishes $(ii)$ for the inductive case. Now, to see $(i)$, note that, since $S^n(\alpha)$ and $S^{n-1}(\alpha)$ are two distinct variables (by $(ii)$ as just shown) which are comparable under $C$ (by $(iii)$ as just shown), it follows that both variables must be mentioned in $C$, hence, in particular, $S^n(\alpha) \in FTV(C)$. Since we have, by induction hypothesis, that $D_{n-1} \subseteq FTV(C)$, it follows that $D_n \subseteq FTV(C)$. The proof of the *claim* is now complete.

Now, to prove the lemma, consider the set $V = \{S^n(\alpha) \mid n \geq 0\}$. By property $(i)$ of our *claim* above, we have $V \subseteq FTV(C)$, and therefore $V$ is a finite set of variables with $S$ an injection of $V$ into itself (and, by finiteness of $V$, $S$ is therefore a bijection of $V$ onto itself.) Hence, by finiteness of $V$, there must exist $i < j$ such that $S^j(\alpha) = S^i(\alpha)$. By property $(ii)$ of the *claim* we have $S^j(\alpha) \neq S^{j-1}(\alpha)$, and by property $(iii)$ one easily sees that $C \vdash_P S^j \leq S^i$ whenever $i \leq j$. We have therefore established that, under the subtype assumptions $C$, we have

$$S^j(\alpha) < S^{j-1}(\alpha) < \ldots < S^i(\alpha)$$

with $S^i(\alpha) = S^j(\alpha)$ (and $S^j(\alpha) < S^{j-1}(\alpha)$ shorthand for $C \vdash_P S^j(\alpha) \leq S^{j-1}(\alpha)$ with $S^j(\alpha) \neq S^{j-1}(\alpha)$.) The sequence shown establishes that there is a proper cycle in $C$. □

Using Lemma 3.17 we can now prove the following important property:

**Proposition 3.18** *If $t_1 \approx^\bullet t_2$ and $t_1, t_2$ are both fully substituted, then there is a renaming $S$ on $t_2$ such that*

1. $C_1 \sim_P S(C_2)$
2. $\tau_1 = S(\tau_2)$
3. $Dm(\Gamma_1) = Dm(\Gamma_2)$ *and* $\forall x \in Dm(\Gamma_1).\ \Gamma_1(x) = S(\Gamma_2(x))$

*Proof*: By $t_1 \approx^\bullet t_2$, we know that $t_1 \prec_{S_1} t_2$ and $t_2 \prec_{S_2} t_1$ with $S_1$ a renaming on $t_1$ and $S_2$ a renaming on $t_2$. We claim that the proposition holds with $S = S_2$. Part (1) follows directly from Lemma 3.13 and the assumption which yields

$$C_1 \vdash_P S_2(C_2) \tag{17}$$

assumption, we have $A \leq A', A' \leq A \in (C \cup P)^*$, so that $(C \cup P)^*$ contains a proper cycle; then $C \cup P$ must also contain a proper cycle. □

The proof of the following lemma uses our assumption that only typing judgements with consistent coercion sets are considered.

**Lemma 3.16** *If $t = C, \Gamma \vdash_P M : \tau$ is fully substituted with $C$ consistent, then $C$ is acyclic.*

*Proof*: We must show that $C \cup P$ contains no proper cycle. Suppose that $C \cup P$ contains a proper cycle $A_1, \ldots, A_n$. If all of the elements of this cycle are constants from $P$, then $C$ cannot be consistent. By the assumption of concictency we can therefore assume that the cycle contains at least one variable, $\alpha$. But then there exists $t'$ such that $t \mapsto_{ce} t'$, so $t'$ is a strict substitution instance of $t$ with $t' \in [t]$, hence $t$ cannot be fully substituted. □

We now prove a main technical lemma. The proof of it has a simple abstract combinatorial core, which it may be instructive to sketch before giving the proof in detail. Suppose $f : A \to A$ is a bijection of a finite set $A$ onto itself, and suppose that $\leq$ is any binary relation on $A$ (we use a very suggestive notation, $\leq$, but it could be any binary relation) such that $f$ is "monotone" with respect to $\leq$, i.e., $a \leq a' \Rightarrow f(a) \leq f(a')$. Then it will be the case, for any $a \in A$, that if $f(a) \neq a$ and either $f(a) \leq a$ or $a \leq f(a)$, then $A$ must contain a proper cycle with respect to $\leq$, i.e., there is a sequence of elements $a_1 < \ldots < a_n$ with $a_1 = a_n$, where $a < a'$ means that $a \leq a'$ and $a \neq a'$. To see this, consider that $f$ injective and $f(a) \neq a$ imply $f^n(a) \neq f^{n-1}(a)$ for all $n$, and, moreover, assuming $f(a) \leq a$ (the other case where $a \leq f(a)$ is similar) we also have $f^n(a) \leq f^{n-1}(a)$ for all $n$, by "monotonicity" of $f$; in total we have $f^n(a) < f^{n-1}(a)$ for all $n$. Now consider the set $V = \{f^n(a) \mid n \geq 0\}$. Since $V \subseteq A$ is finite, there must be $i < j$ with $f^j(a) = f^i(a)$. Then $f^j(a) < f^{j-1}(a) < \ldots < f^i(a)$ constitutes a proper cycle in $A$.

In the sequel we use the following property several times: if $C$ is a coercion set such that $C \vdash_P \alpha \leq A$ or $C \vdash_P A \leq \alpha$, where $\alpha$ is a variable and $\alpha \neq A$, then $\alpha$ must be mentioned in $C$, i.e., $\alpha \in FTV(C)$. Note that the assumption $\alpha \neq A$ cannot be dropped here, since $C \vdash_P \alpha \leq \alpha$ for all $C$ and $\alpha$ by the reflexivity rule. We are now ready to prove:

**Lemma 3.17** *Let $S$ be a substitution and $C$ an atomic coercion set with variable $\alpha \in FTV(C)$. Assume*

(1) *$S$ is a renaming on $FTV(C)$*

(2) *$C \vdash_P S(C)$*

(3) *$C$ is acyclic*

(4) *either $C \vdash_P S(\alpha) \leq \alpha$ or $C \vdash_P \alpha \leq S(\alpha)$*

*Then $S(\alpha) = \alpha$.*

*Proof*: We prove that $S(\alpha) \neq \alpha$ together with (1), (2) and (4) imply that $C$ is cyclic. We show the implication under the assumption that $C \vdash_P S(\alpha) \leq \alpha$; the proof of the implication under the alternative assumption $C \vdash_P \alpha \leq S(\alpha)$ is similar and left out.

So assume $\alpha \in FTV(C)$, $S(\alpha) \neq \alpha$, (1), (2) and $C \vdash_P S(\alpha) \leq \alpha$. We must show that $C$ is cyclic. We first show the following *claim*:

*For all $n > 0$ one has:*

(*i*) *The set $D_n = \{S^k(\alpha) \mid 0 \leq k \leq n\}$ satisfies $D_n \subseteq FTV(C)$*

and

$$(S_1(C_1 \cup P))^* = (C_2 \cup P)^* \tag{14}$$

and from (13) and (14) we immediately have $C_1 \sim_P S_2(C_2)$ and $C_2 \sim_P S_1(C_1)$. □

We now consider what can be said about the types visible in typings $t_1$ and $t_2$ when both are fully substituted and $t_1 \approx^\bullet t_2$. This turns out to be rather delicate, and acyclicity of coercion sets will play a major role. The following lemma shows that one can eliminate cycles from a coercion set without loosing generality:

**Lemma 3.14** *Assume* $C \vdash_P \alpha \leq \tau$ *and* $C \vdash_P \tau \leq \alpha$. *Then one has*

1. $C \vdash_P \tau' \leq \tau'\{\alpha \mapsto \tau\}$ *for all* $\tau'$
2. $C \vdash_P \tau'\{\alpha \mapsto \tau\} \leq \tau'$ *for all* $\tau'$
3. $C \vdash_P C\{\alpha \mapsto \tau\}$

*Proof*: The first two claims are proven simultaneously by induction on $\tau'$, details are left for the reader. To see the third claim, let $\tau_1 \leq \tau_2 \in C$, then (by the first and second properties of this lemma) we get

$$C \vdash_P \tau_1\{\alpha \mapsto \tau\} \leq \tau_1 \tag{15}$$

and

$$C \vdash_P \tau_2 \leq \tau_2\{\alpha \mapsto \tau\} \tag{16}$$

By (15), (16) together with $C \vdash_P \tau_1 \leq \tau_2$ we then get by transitivity that $C \vdash_P \tau_1\{\alpha \mapsto \tau\} \leq \tau_2\{\alpha \mapsto \tau\}$. This shows the third claim. □

We formally define a transformation of typing judgements, called $\mapsto_{ce}$, which collapses cycles in coercion sets: Let $t = C, \Gamma \vdash_P M : \tau$ and suppose that $C \vdash_P \alpha \leq A$ and $C \vdash_P A \leq \alpha$ with $\alpha \neq A$. Then $t \mapsto_{ce} S(t)$ with $S = \{\alpha \mapsto A\}$. Note that, if $t \mapsto_{ce} t'$, then $t'$ is a *strict* substitution instance of $t$ and, by Lemma 3.14, we have $t' \prec t$, hence $t' \approx t$.

**Lemma 3.15** *(Anti-symmetry)*
*Let* $C$ *be atomic. If* $C \cup P$ *contains no proper cycle, then*

$$C \vdash_P \tau \leq \tau',\ C \vdash_P \tau' \leq \tau \Rightarrow \tau = \tau'$$

*Proof*: We first prove that,

(∗) if $C \vdash_P \tau \leq \tau'$ and $C \vdash_P \tau' \leq \tau$ with $\tau \neq \tau'$, then there are atoms $A, A'$ with $A \neq A'$ such that $C \vdash_P A \leq A'$ and $C \vdash_P A' \leq A$

The claim (∗) is shown by induction on $\tau$, as follows. Assume $C \vdash_P \tau \leq \tau'$ and $C \vdash_P \tau' \leq \tau$ with $\tau \neq \tau'$. If $\tau = A$, an atom, then the Match Lemma shows that $\tau' = A'$, also an atom, and (∗) follows in this case. For the inductive step, if $\tau = \tau_1 \to \tau_2$, then the Match Lemma entails that $\tau' = \tau_1' \to \tau_2'$; the assumptions together with the Decomposition Lemma then imply that $C \vdash_P \{\tau_1' \leq \tau_1, \tau_2 \leq \tau_2'\}$ and $C \vdash_P \{\tau_1 \leq \tau_1', \tau_2' \leq \tau_2\}$. Since $\tau \neq \tau'$, it must be the case that either $\tau_1 \neq \tau_1'$ or $\tau_2 \neq \tau_2'$. In either case, induction hypothesis yields the desired result. This completes the proof of (∗).

We now use (∗) to prove the lemma. By (∗) it is sufficient to prove that, whenever $C \vdash_P A \leq A'$, $C \vdash_P A' \leq A$ with $A \neq A'$, then $C \cup P$ must contain a proper cycle. But, under the

*Proof*: Since $S_1(C_1) \subseteq C_2$, we have $|S_1(C_1)| \leq |C_2|$; likewise, we have $|S_2(C_2)| \leq |C_1|$. Since (appropriate restrictions of) $S_1$ and $S_2$ are injective, we have $|S_2(C_2)| = |C_2|$ and $|S_1(C_1)| = |C_1|$. Hence, we have

$$|C_1| = |S_1(C_1)| \leq |C_2| = |S_2(C_2)|$$

and therefore $|C_1| = |S_2(C_2)|$. Since $S_2(C_2) \subseteq C_1$, it then follows that $S_2(C_2) = C_1$. That $S_1(C_1) = C_2$ is shown analogously. □

Transitive closure does not, in general, commute with substitution; but when the substitution is injective, then it does:

**Lemma 3.12** *If $S$ is an atomic substitution, $C$ an atomic coercion set and $S\,|_{FTV(C)}$ is injective, then $S(C)^* = S(C^*)$.*

*Proof*: To see that $S(C)^* \subseteq S(C^*)$, it is sufficient to show that $S(C^*)$ is transitively closed. So assume $A_1 \leq A_2 \in S(C^*)$ and $A_2 \leq A_3 \in S(C^*)$. Then there are $A_1', A_2', A_2'', A_3'$ such that $A_1' \leq A_2', A_2'' \leq A_3' \in C^*$ with $S(A_1') = A_1, S(A_2') = A_2, S(A_2'') = A_2, S(A_3') = A_3$. Since $S$ is injective, it follows that $A_2' = A_2''$, hence $A_1' \leq A_3' \in C^*$, and so $S(A_1') \leq S(A_3') \in S(C^*)$, i.e., $A_1 \leq A_3 \in S(C^*)$.

To see that $S(C^*) \subseteq S(C)^*$, we define, for $n \geq 0$, the set $C^n$, by setting $C^0 = C$ and $C^{n+1} = C^n \cup \{A \leq A' \mid \exists A''.\ A \leq A'', A'' \leq A' \in C^n\}$. Then clearly, $A \leq A' \in C^*$ if and only if $A \leq A' \in C^n$ for some $n$. Now let $A_1 \leq A_2 \in S(C^*)$, then $A_1 = S(A_1')$ and $A_2 = S(A_2')$ for some inequality $A_1' \leq A_2' \in C^*$. We proceed to show that $A_1' \leq A_2' \in C^n$ implies $S(A_1') \leq S(A_2') \in S(C)^*$ for all $n$, by induction on $n$. For $n = 0$ we have $A_1' \leq A_2' \in C$, hence $S(A_1') \leq S(A_2') \in S(C) \subseteq S(C)^*$, and the claim holds in this case. If $A_1' \leq A_2' \in C^{n+1}$ because $A_1' \leq A, A \leq A_2' \in C^n$, then induction hypothesis entails that $S(A_1') \leq S(A), S(A) \leq S(A_2') \in S(C)^*$, hence (since $S(C)^*$ is transitively closed) we have $S(A_1') \leq S(A_2') \in S(C)^*$. □

**Lemma 3.13** *Let $C_1, C_2$ be atomic coercion sets, and assume that $C_1 \vdash_P S_2(C_2)$ and $C_2 \vdash_P S_1(C_1)$ where $S_i$ is a renaming on $FTV(C_i)$. Then $C_1 \sim_P S_2(C_2)$ and $C_2 \sim_P S_1(C_1)$.*

*Proof*: Assuming w.l.o.g. that $(C_1)^r = (C_2)^r = \emptyset$, $C_1 \vdash_P S_2(C_2)$ entails that

$$(S_2(C_2 \cup P))^* \subseteq (C_1 \cup P)^* \tag{7}$$

and $C_2 \vdash_P S_1(C_1)$ entails that

$$(S_1(C_1 \cup P))^* \subseteq (C_2 \cup P)^* \tag{8}$$

Since $S_1$ and $S_2$ are injections (as renamings) we get from (7) and (8), by Lemma 3.12, that

$$S_2((C_2 \cup P)^*) \subseteq (C_1 \cup P)^* \tag{9}$$

and

$$S_1((C_1 \cup P)^*) \subseteq (C_2 \cup P)^* \tag{10}$$

Then Lemma 3.11 entails that

$$S_2((C_2 \cup P)^*) = (C_1 \cup P)^* \tag{11}$$

and

$$S_1((C_1 \cup P)^*) = (C_2 \cup P)^* \tag{12}$$

Using Lemma 3.12 again, with (11) and (12), we get

$$(S_2(C_2 \cup P))^* = (C_1 \cup P)^* \tag{13}$$

**Definition 3.10** (Cyclic coercion sets) *We say that an atomic coercion set $C$ is* cyclic *if there are atoms $A_1, \ldots, A_n$, $n \geq 2$, with $A_1 = A_n$ and $A_1 \leq A_2, \ldots, A_{n-1} \leq A_n \in C \cup P$ and with $A_i \neq A_j$ for some $i, j \in \{1, \ldots, n\}$. The sequence $A_1, \ldots, A_n$ is called a* proper cycle. *Note that inequalities like $A \leq A$ (i.e., a "loop") can occur in an acyclic coercion set.* □

In case $C$ is acyclic, the set $C^-$ is called the *transitive reduction* of $C$ (see [1]) and $C^-$ satisfies $(C^-)^* = C^*$ and whenever $(C')^* = C^*$ then $C^- \subseteq C'$, hence $C^-$ is the unique, minimal set which generates the transitive closure of $C$ (see [1] for more details.) Observe also that, by the Decomposition Lemma, we have $C_1 \sim_P C_2$ for any two atomic coercion sets $C_1, C_2$ if and only if $C_1$ and $C_2$ prove the same set of atomic inequalities $A \leq A'$. Hence, comparing two atomic sets with respect to $\vdash_P$ reduces to comparing the sets with respect to atomic inequalities. We list a few more useful properties:

(1) If $C_1$ and $C_2$ are atomic and $(C_1)^r = (C_2)^r$, then $C_1 \sim_P C_2 \Leftrightarrow (C_1 \cup P)^* = (C_2 \cup P)^*$

(2) If $C_1$ and $C_2$ are atomic, then $C_1 \sim_P C_2 \Leftrightarrow C_1 \setminus (C_1)^r \sim_P C_2 \setminus (C_2)^r$

(3) If $C_1$ and $C_2$ are both atomic and consistent with $P$, then $C_1 \sim_P C_2 \Leftrightarrow (C_1)^\circ \sim_P (C_2)^\circ$

(4) If $C_1$ and $C_2$ are both atomic and consistent with $P$, then $C_1 \sim_P C_2 \Leftrightarrow ((C_1)^\circ)^* = ((C_2)^\circ)^*$

(5) If $C$ is atomic, consistent with $P$ and acyclic, then $C' \sim_P C \Rightarrow (C^\circ)^- \subseteq C'$

(6) If $C$ is atomic and consistent with $P$, then $C \sim_P (C^\circ)^-$

*Ad* 1. It is clear that $\Leftarrow$ holds by the reflexivity rule and the fact that, for $A \neq A'$, $C \vdash_P A \leq A' \Leftrightarrow A \leq A' \in (C \cup P)^*$, for any atomic $C$. To see $\Rightarrow$, assume $C_1 \sim_P C_2$ and suppose $A \leq A \in (C_1 \cup P)^*$. Then, either $(i)$ there is a proper cycle in $C_1 \cup P$ connecting $A$ to itself, or $(ii)$ $A \leq A \in (C_1)^r$. In case $(i)$, $C_1 \sim_P C_2$ shows that the same cycle must be in $(C_2 \cup P)$ and hence $A \leq A \in (C_2 \cup P)^*$; in case $(ii)$, we know by $(C_1)^r = (C_2)^r$ that $A \leq A \in (C_2)^r$, too. This shows that $(C_1 \cup P)^* \subseteq (C_2 \cup P)^*$, and the other inclusion follows by symmetry.

*Ad* 2. We clearly have $C \sim_P C^r$ for any $C$, by the reflexivity rule.

*Ad* 3. The provability relation $\vdash_P$ adds all the information in $C^r \cup C^P$ when $C$ is consistent with $P$.

*Ad* 4. If $((C_1)^\circ)^* = ((C_2)^\circ)^*$, then $((C_1)^\circ \cup P)^* = ((C_2)^\circ \cup P)^*$, and since $((C_1)^\circ)^r = ((C_2)^\circ)^r = \emptyset$, it follows from (1) that $(C_1)^\circ \sim_P (C_2)^\circ$, and hence, by (3), $C_1 \sim_P C_2$. Conversely, assume $C_1 \sim_P C_2$ and suppose that $A \leq A' \in ((C_1)^\circ)^*$. Then there is a path $p$ from $A$ to $A'$ in $(C_1)^\circ$ which has no edge in common with $P$ (regarded as a digraph.) Since $C_1 \sim_P C_2$ implies $((C_1)^\circ \cup P)^* = ((C_2)^\circ \cup P)^*$ by (1) and (3), the same path $p$ must be in $((C_2)^\circ \cup P)^*$ with no edge in common with $P$, and therefore $p$ must be a path within $(C_2)^\circ$, hence $A \leq A' \in ((C_2)^\circ)^*$. We have shown $((C_1)^\circ)^* \subseteq ((C_2)^\circ)^*$, and the other inclusion follows by symmetry.

*Ad* 5. Since $C$ is consistent and $C' \sim_P C$, $C'$ is also consistent, and so $C' \sim_P C$ implies (by (4)) that $((C')^\circ)^* = (C^\circ)^*$. Since $C$ is acyclic, this implies $(C^\circ)^- \subseteq (C')^\circ$, hence $(C^\circ)^- \subseteq C'$.

*Ad* 6. By consistency of $C$ we clearly have $C \sim_P C^\circ$. Since $(C^\circ)^* = ((C^\circ)^-)^*$, we get $(C^\circ \cup P)^* = ((C^\circ)^- \cup P)^*$, hence (by (1) and $(C^\circ)^r = \emptyset = ((C^\circ)^-)^r$) we have $C^\circ \sim_P (C^\circ)^-$, and consequently $C \sim_P (C^\circ)^-$.

**Lemma 3.11** *Assume $S_1(C_1) \subseteq C_2$ and $S_2(C_2) \subseteq C_1$ for substitutions $S_1$, $S_2$ such that $S_1$ is injective on $FTV(C_1)$ and $S_2$ is injective on $FTV(C_2)$. Then $S_1(C_1) = C_2$ and $S_2(C_2) = C_1$.*

Moreover, by $(ii)$ and the Substitution Lemma we have

$$S_2(C_2) \vdash_P S_2(S_1(\tau_1)) \leq S_2(\tau_2) \tag{3}$$

hence, by $(iv)$ and (3) we get

$$C_1 \vdash_P S_2(S_1(\tau_1)) \leq S_2(\tau_2) \tag{4}$$

and then, by $(v)$ and (4)

$$C_1 \vdash_P S_2(S_1(\tau_1)) \leq \tau_1 \tag{5}$$

In analogous manner one obtains

$$C_1 \vdash_P \Gamma_1(x) \leq S_2(S_1(\Gamma_1(x))) \tag{6}$$

Then (2), (5) and (6) show that $S_1(t_1) \prec_{S_2} t_1$. The proof that $S_2(t_2) \prec_{S_1} t_2$ is symmetric and left out. □

Note that Lemma 3.7 immediately entails that, if $t_1 \prec_{S_1} t_2$ and $t_2 \prec_{S_2} t_1$, then $S_2(S_1(t_1)) \prec_{id} t_1$ and $S_1(S_2(t_2)) \prec_{id} t_2$.

**Lemma 3.8** *If $t_1 \approx t_2$ and $t_1$ and $t_2$ are fully substituted, then $t_1 \approx^\bullet t_2$.*

*Proof*: By $t_1 \approx t_2$ we have $t_1 \prec_{S_1} t_2$ and $t_2 \prec_{S_2} t_1$ for some substitutions $S_1, S_2$. By Lemma 3.7 we then have $S_1(t_1) \prec_{S_2} t_1$ and $S_2(t_2) \prec_{S_1} t_2$, hence $S_1(t_1) \in [t_1]$ and $S_2(t_2) \in [t_2]$. Since $t_1$ is fully susbtituted, it follows that $S_1$ is a renaming on $FTV(t_1)$ and since $t_2$ is fully substituted, it follows that $S_2$ is a renaming on $FTV(t_2)$. Then $t_1 \prec_{S_1} t_2$ and $t_2 \prec_{S_2} t_1$ establish that $t_1 \approx^\bullet t_2$. □

We now aim at strengthening the conclusion of Lemma 3.8. The driving motivation behind the following development is that we wish to answer the question: what can we say about typings $t_1$ and $t_2$ if we know that both are fully substituted and $t_1 \approx^\bullet t_2$? Answering this question is a main technical part of our result, and it turns out to be slightly subtle.

First we consider what can be said about atomic coercion sets which are equivalent modulo renaming substitutions. This is rather straight-forward. We begin with some useful operations on coercion sets, some of which will not be used until later.

**Definition 3.9** *(Operations on coercion sets)*
*Let $C$ be an atomic coercion set. Then we define the following operations on $C$:*

$C^r = \{A \leq A \mid A \leq A \in C\}$

$C^P = \{b \leq b' \in C \mid b, b' \in P\}$

$C^* =$ *the transitive closure of* $C$

$C^\circ = C \setminus (C^r \cup C^P)$

$C^- = \bigcap\{C' \mid (C')^* = C^*\}$

□

We consider the poset $P$ as a coercion set, or, equivalently, a digraph, by representing $P$ by its Hasse diagram, and we can then consider $C \cup P$ as a coercion set. Clearly, for $A \neq A'$ we have $C \vdash_P A \leq A'$ if and only if $A \leq A' \in (C \cup P)^*$.

Our first task will be to show a uniqueness property for fully substituted, equivalent typings: if $t_1$ and $t_2$ are both fully substituted and $t_1 \approx t_2$ then $t_1 \approx^\bullet t_2$. As it turns out, this property can be considerably strengthened, but we use it as a convenient intermediate stage towards a stronger uniqueness property.

We first establish a few basic properties of $\approx$. The first one shows that we can restrict attention to instantiation under *atomic substitutions*.

**Lemma 3.5** *If there exist substitutions $S, S'$ such that $S(\tau)$ matches $\tau'$ and $S'(\tau')$ matches $\tau$, then $\tau$ matches $\tau'$.*

*Proof*: The proof is by a straight-forward induction on $\tau$. □

**Lemma 3.6** *If $t \overset{S}{\mapsto} t'$ and $t' \in [t]$, then $S \mid_{FTV(t)}$ is an atomic substitution.*

*Proof*: Let $t = C, \Gamma \vdash_P M : \tau$. By the assumptions, $t' = S(t)$ is an atomic judgement, so $S \mid_{FTV(C)}$ is an atomic substitution. Since $t' \in [t]$, we have $t \approx t'$, and therefore Lemma 3.5 shows that $S(\tau)$ matches $\tau$ and that $\Gamma(x)$ matches $S(\Gamma(x))$ for all $x \in Dm(\Gamma)$. It follows that $S \mid_{FTV(\tau) \cup FTV(\Gamma)}$ is an atomic substitution. We have now shown that $S \mid_{FTV(t)}$ is atomic. □

Lemma 3.6 shows that, for every atomic judgement $t$, the equivalence class $[t]$ contains no infinite descending chains $t_1 \overset{S_1}{\mapsto}_\circ t_2 \overset{S_2}{\mapsto}_\circ \ldots$, because for any *atomic* substitution $S$ such that $t_i \overset{S}{\mapsto}_\circ t_j$ we must have $|FTV(t_j)| < |FTV(t_i)|$. Hence, every equivalence class $[t]$ has at least one *last element* wrt. $\mapsto_\circ$, i.e., an element $t' \in [t]$ such that for no $t'' \in [t]$ is it the case that $t' \mapsto_\circ t''$. Moreover, we see that for any typing judgement $t$, there exists a full substitution instance of $t'$ of $t$ within $[t]$: to obtain $t'$, let $t'$ be the last element in a maximal chain of atomic, consistent judgements within $[t]$ starting from $t$, $t \mapsto_\circ t_1 \mapsto_\circ \ldots \mapsto_\circ t'$.

**Lemma 3.7** *If $t_1 \prec_{S_1} t_2$ and $t_2 \prec_{S_2} t_1$, then $S_1(t_1) \prec_{S_2} t_1$ and $S_2(t_2) \prec_{S_1} t_2$.*

*Proof*: Let $t_1 = C_1, \Gamma_1 \vdash_P M : \tau_1$, $t_2 = C_2, \Gamma_2 \vdash_P M : \tau_2$. By the assumption, we have

$(i)$ $C_2 \vdash_P S_1(C_1)$

$(ii)$ $C_2 \vdash_P S_1(\tau_1) \leq \tau_2$

$(iii)$ $Dm(\Gamma_1) \subseteq Dm(\Gamma_2)$ and $\forall x \in Dm(\Gamma_1).\ C_2 \vdash_P \Gamma_2(x) \leq S_1(\Gamma_1(x))$

and

$(iv)$ $C_1 \vdash_P S_2(C_2)$

$(v)$ $C_1 \vdash_P S_2(\tau_2) \leq \tau_1$

$(vi)$ $Dm(\Gamma_2) \subseteq Dm(\Gamma_1)$ and $\forall x \in Dm(\Gamma_2).\ C_1 \vdash_P \Gamma_1(x) \leq S_2(\Gamma_2(x))$

By $(i)$ and the Substitution Lemma we have

$$S_2(C_2) \vdash_P S_2(S_1(C_1)) \tag{1}$$

and so, by $(iv)$ and (1), we get

$$C_1 \vdash_P S_2(S_1(C_1)) \tag{2}$$

**Definition 3.2** *(Minimality)*
*A typing judgement $t$ is called* minimal *iff it holds for all $t' \in [t]$ that $t \lesssim t'$.* □

If $t = C, \Gamma \vdash_P M : \tau$ and $S$ is a type substitution, we write $S(t) = S(C), S(\Gamma) \vdash_P M : S(\tau)$. If $t_1 = S(t_2)$ with $S$ a renaming on $FTV(t_2)$, then we say that $t_1$ and $t_2$ are *congruent*, written $t_1 \cong t_2$. Clearly, $\cong$ is an equivalence relation. Let us observe immediately that minimal typings are unique up to renaming if they exist:

**Theorem 3.3** *(Uniqueness of minimal typings)*
*The relation $\lesssim$ is a partial order up to renaming substitutions: if $t_1 \lesssim_{S_2} t_2$ and $t_2 \lesssim_{S_1} t_1$ then $t_1 \cong t_2$ with $S_1$ a renaming on $t_1$, $S_2$ a renaming on $t_2$ and $t_1 = S_2(t_2)$ and $t_2 = S_1(t_1)$.*

*Proof*: Let $t_1 = C_1, \Gamma_1 \vdash_P M : \tau_1$ and $t_2 = C_2, \Gamma_2 \vdash_P M : \tau_2$. If $t_1 \lesssim_{S_2} t_2$ and $t_2 \lesssim_{S_1} t_1$ then, in particular, $C_1 \subseteq S_2(C_2)$ and $C_2 \subseteq S_1(C_1)$. Then we have $|S_2(C_2)| \leq |C_2| \leq |S_1(C_1)| \leq |C_1|$ (using the second inclusion) and hence (by the first inclusion) we have $C_1 = S_2(C_2)$; that $C_2 = S_1(C_1)$ follows by an analogous argument. But then, by composing substitutions, we get from $t_1 \lesssim_{S_2} t_2$ and $t_2 \lesssim_{S_1} t_1$ that

(1) $C_1 = S_2(S_1(C_1))$

(2) $\tau_1 = S_2(S_1(\tau_1))$

(3) $\forall x \in Dm(\Gamma_1).\ \Gamma_1(x) = S_2(S_1(\Gamma_1(x)))$

This shows that $S_2 \circ S_1$ is a renaming on $t_1$, hence $S_1$ is a renaming on $t_1$ and $t_2 = S_1(t_1)$; by a similar argument, $S_2$ is a renaming on $t_2$ with $t_1 = S_2(t_2)$. □

## 3.3 Existence of minimal typings

The hard part is to show that minimal typings exist in every equivalence class $[t]$. Basically, it will turn out that we shall always find a minimal typing in $[t]$ by taking a *full substitution instance* (see definition below) of $t$ within $[t]$ and then optimizing the representation of the coercion set of the full substitution instance. The fact that a judgement obtained in this way is minimal will follow from a uniqueness property of fully substituted judgements within the equivalence class $[t]$.

**Definition 3.4** *(Substitution instance, full substitution instance)*
*If $t' = S(t)$, then we say that $t'$ is a* substitution instance of $t$ under $S$, *written $t \overset{S}{\mapsto} t'$. We say that $t'$* is a substitution instance of $t$, *written $t \mapsto t'$, if there exists a substitution $S$ such that $t \overset{S}{\mapsto} t'$. If $t \overset{S}{\mapsto} t'$ where $S$ is* not a renaming *on $FTV(t)$, then we say that $t'$ is a* strict substitution instance *of $t$ under $S$, and we write $t \overset{S}{\mapsto}_\circ t'$ in this case and $t \mapsto_\circ t'$ if $t \overset{S}{\mapsto}_\circ t'$ for some $S$.*

*A typing judgement $t$ is called* fully substituted *iff, whenever $S(t) \in [t]$, then $S$ is a renaming on $FTV(t)$. In other words, if $t$ is fully substituted then there is no strict substitution instance of $t$ within $[t]$. A* full substitution instance *of a typing $t$ is a substitution instance of $t$ which is fully substituted.* □

Note that any substitution instance $S(t)$ of $t$ is an instance of $t$, provided $S$ is an atomic substitution such that $S(C)$ is consistent, since in this case $S(t)$ is an atomic typing judgement with $t \prec_S S(t)$. Hence, if $S(t) \prec t$ then $S(t) \approx t$.

way, which we shall adopt here, of defining the information content of a typing, is by defining a notion of *instance* and taking the information content of a typing to be the set of its instances. In our setting, this leads to the requirement that, if $\mapsto$ is a transformation on judgements and $t$ and $t'$ are judgements such that $t \mapsto t'$, then $t \approx t'$ must hold. The property $t \mapsto t' \Rightarrow t \approx t'$ can be regarded as a soundness property for $\mapsto$, which guarantees that the transformation looses no typing power; in particular, a sound transformation will preserve principality of typings.

Other notions of soundness are possible. For instance, one can imagine more powerful notions of soundness based on semantic concepts rather than the syntactic notion of instance used here [1] By "more powerful" we mean here "more admissible" in the sense that more typings can be regarded as equivalent with respect to information content. The instance relation $\prec$ is interesting as an object of study, though, because it is natural and powerful enough to validate many non-trivial typing transformations. Its definition fully exploits the *logical* [2] rules of the type system. This is in contrast with the notion of instance used in [12], which uses the more restrictive conditions $S(\tau_1) = \tau_2$ (instead of condition (2) of Definition 2.1) and $\Gamma_2(x) = S(\Gamma_1(x))$ (instead of condition (3) of Definition 2.1.) Using this notion of instance we cannot validate quite simple typing transformations such as, e.g., cycle elimination in coercion sets. For example, it is natural to consider the typing $\{\alpha \leq \beta, \beta \leq \alpha\}, \emptyset \vdash_P \lambda x.x : \alpha \to \beta$ equivalent to (the more desirable) $\emptyset, \emptyset \vdash_P \lambda x.x : \alpha \to \alpha$, but since $\alpha \to \beta$ is not a substitution instance of $\alpha \to \alpha$, the two typings are not equivalent with respect to the more restrictive notion of instance.

In the remainder of this section we first define what it means for a typing judgement to be *minimal*. We then prove that minimal typings (if they exist) are unique up to renaming substitutions. We then prove that minimal typing judgements always exist. The hard part is to show existence.

## 3.2 Minimality and uniqueness of minimal typings

For a typing judgement $t$ we let $[t]$ denote the $\approx$-equivalence class of $t$, i.e., $[t] = \{t' \mid t' \approx t\}$.

**Definition 3.1** *(Specialization)*
*Let* $t_1 = C_1, \Gamma_1 \vdash_P M : \tau_1, t_2 = C_2, \Gamma_2 \vdash_P M : \tau_2$. *We say that* $t_1$ is a specialization of $t_2$, *written* $t_1 \lesssim t_2$, *iff there exists a substitution* $S$ *on* $t_2$ *such that*

1. $C_1 \subseteq S(C_2)$
2. $\tau_1 = S(\tau_2)$
3. $Dm(\Gamma_1) \subseteq Dm(\Gamma_2)$ *and* $\forall x \in Dm(\Gamma_1).\ \Gamma_1(x) = S(\Gamma_2(x))$

*We may write* $t_1 \lesssim_S t_2$ *to signify that* $t_1 \lesssim t_2$ *under substitution* $S$. □

Note that, if $t \lesssim_S t'$, then $t \prec_{id} S(t')$.

We now define what it means for a typing judgement $t$ to be *minimal*. Intuitively, a minimal typing $t$ is a *most specialized* element within $[t]$. In a sense, therefore, a minimal typing is an irredundant representative in its equivalence class; it contains minimal information needed to generate (by the logical operations of instantiation) all the elements of its equivalence class.

[1] See also [5] where a *generic* concept of instance is defined and studied. The introduction of this concept is an attempt to capture the properties which should be satisfied by any notion of instance.

[2] A typing rule is logical if it *non-structural*, that is, it can be applied to a term independently of the syntactic structure of the term. The rule [*sub*] is the only logical typing rule of the subtype system considered here.

homomorphically to types. The *support* of $S$, written $\mathrm{Supp}(S)$, is $\{\alpha \in \mathcal{V} \mid S(\alpha) \neq \alpha\}$, i.e., the set of variables not mapped to themselves by $S$. If the support of $S$ is finite with $\mathrm{Supp}(S) = \{\alpha_1, \ldots, \alpha_n\}$, then we sometimes write just $S = \{\alpha_1 \mapsto S(\alpha_1), \ldots, \alpha_n \mapsto S(\alpha_n)\}$. If $V \subseteq \mathcal{V}$ then the *restriction of $S$ to $V$*, denoted $S\,|_V$, is the substitution $S'$ such that $S'(\alpha) = S(\alpha)$ for $\alpha \in V$ and $S'(\alpha) = \alpha$ for $\alpha \notin V$. If $S(\alpha)$ is an *atom* (i.e., a constant in $P$ or a variable) for all $\alpha$ then $S$ is called an *atomic substitution*. If $S(\alpha) \neq S(\beta)$ for all $\alpha, \beta \in V$ with $\alpha \neq \beta$, then $S$ is said to be *injective on $V$*. If $S$ maps all variables in $V$ to variables and $S$ is injective on $V$, then $S$ is called a *renaming on $V$*. If $t = C, \Gamma \vdash_P M : \tau$ we let $FTV(C), FTV(\Gamma), FTV(\tau)$ denote, respectively, the free type variables appearing in $C$, $\Gamma$, $\tau$. We let $FTV(t) = FTV(C) \cup FTV(\Gamma) \cup FTV(\tau)$. If $S$ is a renaming on $FTV(t)$ we sometimes say just that $S$ is a renaming on $t$, for short.

The following definition gives the instance relation as defined by Fuh and Mishra in [3]. There the relation is called *lazy instance*, here it is just called the instance relation.

**Definition 2.1** *(Instance relation)*
*Let $t_1 = C_1, \Gamma_1 \vdash_P M : \tau_1$, $t_2 = C_2, \Gamma_2 \vdash_P M : \tau_2$ be two atomic judgements, and let $S$ be a type substitution. We say that $t_2$* is an instance of *$t_1$* under *$S$, written $t_1 \prec_S t_2$, iff*

1. $C_2 \vdash_P S(C_1)$
2. $C_2 \vdash_P S(\tau_1) \leq \tau_2$
3. $Dm(\Gamma_1) \subseteq Dm(\Gamma_2)$ *and* $\forall x \in Dm(\Gamma_1).\ C_2 \vdash_P \Gamma_2(x) \leq S(\Gamma_1(x))$

*We say that $t_2$* is an instance of *$t_1$, written $t_1 \prec t_2$, iff there exists a type substitution $S$ such that $t_1 \prec_S t_2$. We say that $t_1$ and $t_2$ are* equivalent, *written $t_1 \approx t_2$, iff $t_1 \prec t_2$ and $t_2 \prec t_1$. If $t_1 \prec_{S_1} t_2$ and $t_2 \prec_{S_2} t_1$ where $S_1$ is a renaming on $FTV(t_1)$ and $S_2$ is a renaming on $FTV(t_2)$, then we say that $t_1$ and $t_2$ are* equivalent under renaming *and we write $t_1 \approx^\bullet t_2$ in this case. Clearly, $\approx$ and $\approx^\bullet$ are equivalence relations on typing judgements.* □

A typing judgement for a term $M$ is called a *principal typing for $M$* if it is a derivable judgement in the type system, and it has all other derivable judgements for $M$ as instances.

We end our preliminaries by recalling (without proof) three fundamental properties of the subtype logic:

**Lemma 2.2** *(Substitution Lemma)*
*If $C \vdash_P \tau \leq \tau'$, then $S(C) \vdash_P S(\tau) \leq S(\tau')$.*

**Lemma 2.3** *(Match Lemma)*
*If $C$ is atomic, and $C \vdash_P \tau \leq \tau'$, then $\tau$ and $\tau'$ match.*

**Lemma 2.4** *(Decomposition Lemma)*
*If $C$ is atomic, then $C \vdash_P \tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'$ if and only if $C \vdash_P \{\tau_1' \leq \tau_1, \tau_2 \leq \tau_2'\}$.*

# 3 Existence and uniqueness of minimal typings

## 3.1 Soundness of typing transformations

The goal of subtype simplification, as considered in this paper, is to transform judgements into more desirable forms. A key technique here is to eliminate redundant variables from coercion sets. However, we require that such transformations *preserve the information content of typings*. One

(3) We prove, perhaps the main result of the paper, a tight exponential lower bound for the size of most general typings relative to the instance relation of [3]. This is a non-trivial result, because the instance relation used validates many interesting simplifications. To the best of our knowledge, the best previous result is the linear lower bound proven in [5] for a generic instance relation. Our result shows that there is an intrinsic limit to what simplification under instance relations not stronger than that of [3] can ever achieve. The proof uses new techniques and draws upon the characterization of minimal typings as well as a completeness result mentioned above.

The results are not only negative, for they enhance our understanding of type simplification and its complexity Such understanding may be instrumental for devising type systems that avoid the problems of complexity shown here (see also [5] for motivation of such results.) Moreover, a partial completeness result shown in Section 4.2 is of a positive nature.

## 2 Preliminaries

We assume types ranged over by $\tau$ and defined by

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

where $\alpha$ ranges over a denumerable set $\mathcal{V}$ of type variables and $b$ ranges over base types drawn from a partially ordered set $P$ of type constants, such as, e.g., `int` (integers) and `real` (reals). An *atomic type* (or, an *atom*, for short) is either a variable or a constant. Atoms are ranged over by $A$. The term language is the lambda calculus extended with term constants,

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

Here $x$ ranges over term variables and $c$ ranges over a given set of constants, such as, e.g., numerals and numerical functions.

We consider *atomic typing judgements* $t$ of the form $t = C, \Gamma \vdash_P M : \tau$ in the standard system of atomic subtyping studied in, e.g., [11], [12], [3], [4]. The type system is a proof system for deriving subtyping judgements, and it is given in Appendix A for reference. In a judgement, $C$ is a *constraint set*, that is a set of subtyping hypotheses of the form $\tau \leq \tau'$, and $\Gamma$ is a set of type assumptions $x : \tau$ for the free variables of $M$. An *atomic constraint set* is a constraint set in which all inequalities have the form $A \leq A'$, i.e., only hypotheses involving atomic types are allowed. The type system contains a logic of subtype relations, which allows derivations of the form $C \vdash_P \tau \leq \tau'$, meaning that under the subtyping hypotheses $C$, the subtyping relation $\tau \leq \tau'$ is derivable from $P$. This logic is also given in Appendix A. In *atomic subtyping*, we consider a judgement $C, \Gamma \vdash_P M : \tau$ a *derivable atomic judgement*, if and only if it can be derived using the rules of the type system, *and*, moreover, $C$ is an atomic constraint set. In atomic subtyping, only the derivable atomic judgements define well-typings. We say that a constraint set $C_1$ *syntactically entails* constraint set $C_2$, written $C_1 \vdash_P C_2$, if and only if $C_1 \vdash_P \tau \leq \tau'$ for all $\tau \leq \tau' \in C_2$; we say that $C_1$ and $C_2$ are *equivalent*, written $C_1 \sim_P C_2$, if and only if $C_1 \vdash_P C_2$ and $C_2 \vdash_P C_1$.

In the following development *we require $C$ to be consistent with $P$*, i.e., we consider only judgements $t = C, \Gamma \vdash_P M : \tau$ such that, whenever $b, b' \in P$ and $C \vdash_P b \leq b'$, then $b \leq_P b'$ holds in $P$. The reason for this restriction is that the theory of minimality for typings which can contain inconsistent coercion sets becomes more complicated, and since we usually have no interest in inconsistent typings, this restriction seems natural.

If $f : A \to B$ is a partial function from $A$ to $B$ then $Dm(f)$ denotes the domain of $f$. A *type susbtitution* $S$ is a function mapping type variables to types, and a substitution is lifted

# 1 Introduction

Subtyping is a fundamental idea in type systems for programming languages, which can in principle be integrated into standard type systems and type inference for languages such as, e.g., the simply typed lambda calculus [2], ML [10], Haskell [6], Miranda [19] as well as being a basic notion in typed object oriented languages. Type inference algorithms infer type information from programs without requiring programmers to insert explicit type declarations in the program and discovers many programming errors at compile time. Moreover, systems of subtyping are becoming increasingly used used in non-standard ways in type based program analysis, where program properties of a typed language are automatically extracted from the program text using inference algorithms. In all cases, subtyping adds extra expressiveness to a type system by allowing an expression to have several types depending on the context in which it occurs. This is achieved by imposing an order relation (subtype order) on types together with a rule (subsumption) which allows any expression of type $\tau$ to have also any type $\tau'$ which is larger than $\tau$ in the given ordering. For example, an integer of type `int` may be converted to a real of type `real`, which can be modeled by having `int` a subtype of `real`.

Subtypings associate not only a type to an expression but also a set of subtyping *constraints* (coercions) which express assumptions about the subtype order which must hold for the given typing. The presence of such assumptions are necessitated by the desire to have most general (principal) typings, which summarize all possible typings for a given term. A form of subtyping which is logically simple and natural, yet expressive enough to be interesting, is *atomic subtyping* [11], [12], [4] where only subtyping relations between atoms (type variables and constants) can occur in typings. However, even though several type inference algorithms have appeared for atomic subtyping (e.g., [12], [3], [4]), it is generally recognized that major obstacles remain for subtype inference to become practicable in a large scale setting. To quote from [5], "the main problems seem to be that the algorithm is inefficient, and the output, even for relatively simple input expressions, appears excessively long and cumbersome to read".

It is the second problem mentioned in the quotation above that we study in this paper. The problem has generated a significant amount of work which aims at *simplifying* constraints in the typings generated by type inference algorithms, e.g., [3], [7], [16],[13], [18].

## Contribution of the paper

This paper contains three main new contributions:

(1) We define a notion of *minimal* typings for atomic subtyping and show that every typable term has a unique minimal typing. Intuitively, a minimal typing is a logically most succinct presentation of all possible equivalent typings for a term, where equivalence is defined in terms of the instance relation defined in [3]. Surprisingly, no such notion has, to our knowledge, been studied previously. Rather than starting from specific simplification techniques and defining minimality as being whatever those techniques yield (as is done in [3]), our notion of minimality is purely logical, defined only in terms of the type system itself, together with a notion of instantiation. The instance relation of [3] is important, because it is natural and powerful enough to validate pragmatically important simplifications. The notion of minimality is surprisingly strong, and therefore the result that minimal typings exist can be used to derive non-trivial properties about the type system (see below.)

(2) Having an independent notion of minimality we can ask whether known simplification techniques are complete for minimization (computing minimal typings.) We give completeness and incompleteness results for the widely used techniques studied in [3].

**Abstract**

This paper studies the problem of simplifying typings and the size-complexity of most general typings in typed programming languages with subtyping. We define a notion of minimal typings relating all typings which are equivalent with respect to instantiation. The notion of instance is that of Fuh and Mishra [3], which supports many interesting simplifications. We prove that every typable term has a unique minimal typing, which is the logically most succinct among all equivalent typings. The notion of minimality is purely logical, defined independently of any particular type simplification technique. We study completeness properties, with respect to our notion of minimality, of well-known simplification techniques. Drawing upon these results, we prove a tight exponential lower bound for the worst case dag-size of coercion sets as well as of types in most general typings. To the best of our knowledge, the best previously proven lower bound was linear.

# Minimal Typings in Atomic Subtyping

Jakob Rehof
DIKU, Department of Computer Science
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark
Electronic mail: rehof@diku.dk
Phone: +45 35321408. Fax: +45 35321405

DIKU Technical report no. D-278

## Contents