Local Type Reconstruction by means of Symbolic Fixed Point Iteration

Torben Amtoft
internet: tamtoft@daimi.aau.dk

DAIMI, Aarhus University
Ny Munkegade, DK-8000 Århus C, Denmark

Abstract. We convert, via a version that uses constraints, a type inference system for strictness analysis into an algorithm which given an expression finds the set of possible typings. Although this set in general does not possess a minimal element, it can be represented compactly by means of *symbolic expressions* in *normal form* – such expressions have the property that once values for the constraint variables with *negative polarity* have been supplied it is straight-forward to compute the minimal values for the constraint variables with *positive polarity*. The normalization process works *on the fly*, i.e. by a leaf-to-root traversal of the inference tree.

1 Background and Motivation

Recently much interest has been devoted to the formulation of program analysis in terms of inference systems, as opposed to e.g. abstract interpretation (for the relationship between those methods see e.g. [Jen91]). This approach is appealing since it separates the question "what is done?" from the question "how is it done?". Of course, the latter issue (that is, implementation of the inference system) has to be dealt with, and a very popular method (often inspired by [Hen91]) is to (re)formulate the inference system in terms of *constraints* and then come up with an algorithm for solving these.

In this paper we shall continue this trend, the type system in question being one for *strictness analysis* (the system has also been presented in [Amt93a]). The characteristic features of our approach are:

- for constraints we define a notion of *normal form* which distinguishes between constraint variables according to their polarity (i.e. whether they occur in co/contravariant position in the type).
- the constraints are normalized *on the fly*, that is during a leaf-to-root traversal of the inference tree (as opposed to first collecting them all and then solve them).
- during the normalization process, some *approximations* are made – without losing precision, however, since the approximation only concerns suboptimal solutions.
- the normalization process involves *symbolic fixed point iteration*.

2 Introduction

We shall be working with the typed and extended λ-calculus. An expression is either a constant c; a variable x; an abstraction $\lambda x.e$; an application $e_1 e_2$; a conditional $\mathsf{if}\ e_1\ e_2\ e_3$ or a recursive definition $\mathsf{rec}\ f\ e$. The set of *underlying* types will be denoted $\mathcal{U}$; such a type is either a base type (Int, Bool, Unit etc.) or a function type $u_1 \to u_2$. Base will denote some base type.

Strictness Analysis: The analysis we want to perform is *strictness analysis*, that is the task of detecting whether a function needs its argument(s). Recent years have seen many approaches to strictness analysis, most based on abstract interpretation – the starting point being the work of Mycroft [Myc80] which was extended to higher order functions by Burn, Hankin and Abramsky [BHA86].

Kuo and Mishra [KM89] presented a type system where types t are formed from 0 (denoting non-termination), 1 (denoting non-termination or termination, i.e. any term) and $t_1 \to t_2$. Accordingly, if it is possible to assign a function the type $0 \to 0$ we know that the function is strict. Wright has proposed alternative type systems [Wri91] where the idea is to annotate the *arrows*, according to whether a function is strict or not.

Strictness Types: The type system used in this paper is the one of Wright, except that we use subtyping instead of polymorphism. Accordingly we define the set of *strictness types*, $\mathcal{T}$, as follows: a strictness type t is either a base type Base or a *strict* function type $t_1 \to_0 t_2$ (denoting that we *know* that the function is strict) or a *general* function type $t_1 \to_1 t_2$ (denoting that we do not know whether the function is strict).

We shall impose an ordering $\leq$ on strictness types, defined by stipulating that $t_1 \to_b t_2 \leq t_1' \to_{b'} t_2'$ iff $t_1' \leq t_1$, $b \leq b'$ and $t_2 \leq t_2'$, and by stipulating that $\mathsf{Int} \leq \mathsf{Int}$ etc. The intuitive meaning of $t \leq t'$ is that t is more informative than t'; for instance it is more informative to know that a function is of type $\mathsf{Int} \to_0 \mathsf{Int}$ than to know that it is of type $\mathsf{Int} \to_1 \mathsf{Int}$. Similarly, it is more informative to know that a function is of type $(\mathsf{Int} \to_1 \mathsf{Int}) \to_0 (\mathsf{Int} \to_0 \mathsf{Int})$ (it maps *arbitrary* functions into strict functions) than to know that is is of type $(\mathsf{Int} \to_0 \mathsf{Int}) \to_0 (\mathsf{Int} \to_0 \mathsf{Int})$ (it maps strict functions into strict functions).

An Inference System: In Fig. 1 we present the inference system the implementation of which is the topic of this paper. An application of the inference system can be found in [Amt93a], where it is shown how one by means of an inference tree can produce a (provably correct) translation from call-by-name into call-by-value which inserts fewer "suspensions" than the naive translation.

A judgment in the system takes the form $\Gamma \vdash e : t, W$ where

- Γ is an environment, represented as a list (i.e. of form $((x_1 : t_1) \ldots (x_n : t_n))$), assigning strictness types to variables;
- e is an expression such that if x is a free variable of e then $\Gamma(x)$ is defined;
- t is a strictness type;

– W maps the domain of Γ into $\{0,1\}$. It may be helpful to think of W as follows: if $W(x) = 0$ then x is needed in order to evaluate e to "head normal form". If $\Gamma = ((x_1 : t_1) \ldots (x_n : t_n))$ and if $W(x_i) = b_i$ we shall often write $W = (b_1 \ldots b_n)$.

A brief explanation of the inference system: the first inference rule (the "subsumption rule") is non-structural and expresses the ability to forget information: if an expression has type t and needs the variables in W, it also has any more imprecise type and will also need any subset of W. The application of this rule might for instance be needed in order to assign the same type to the two branches in a conditional. The rule for constants employs a function CT with the property that for all constants c it holds that $CT(c)$ is a *strict iterated base type*; that is either Base or of type $\mathsf{Base}\rightarrow_0 t$ with t a strict iterated base type. Note in the rule for variables that in order to evaluate x it is necessary to evaluate x but no other variables are needed. The rule for abstractions says that if x is among the variables needed by e then $\lambda x.e$ can be assigned a strict type ($\rightarrow_0$), otherwise not. The rule for applications says that the variables needed to evaluate e_1 are also needed to evaluate $e_1 e_2$; and if e_1 is strict then the variables needed to evaluate e_2 will also be needed to evaluate $e_1 e_2$. The rule for conditionals says that if a variable is needed to evaluate the test then it is also needed to evaluate the whole expression; and also if a variable is needed in order to evaluate *both* branches it will be needed to evaluate the whole expression. The rule for recursion says that if the assumption that f has type t suffices for proving that the body e has type t then the construction $\mathsf{rec}\ f\ e$ also has type t; and that if a variable different from f is needed to evaluate e then this variable is also needed to evaluate $\mathsf{rec}\ f\ e$.

Fig. 1 is a *specification* of an analysis (and in [Amt93a] the specification is proven "correct" wrt. an operational semantics); in particular no clue is given on how to *implement* the analysis, that is how to construct a type for a given expression (and its subexpressions). As we want to focus upon the strictness annotations, we shall assume that the underlying types have been given in advance.

Principal Types: For type inference (type reconstruction) in general, an expression can be assigned many different types – fortunately, however, it is often possible to find a "principal type" such that all other types can be "derived" from this type. We shall now investigate to which extent there exists a "principal typing" in our setting. As a (running) example, we consider the *twice* function defined by $\lambda f.\lambda x.f(f(x))$. If the underlying type of *twice* has been fixed to $(\mathsf{Int}\rightarrow\mathsf{Int})\rightarrow(\mathsf{Int}\rightarrow\mathsf{Int})$, a valid strictness type must be of form $(\mathsf{Int}\rightarrow_{b_1}\mathsf{Int})\rightarrow_{b_2}(\mathsf{Int}\rightarrow_{b_3}\mathsf{Int})$ where the b_i's are *strictness variables*, that is variables which range over $\{0,1\}$. It is not too difficult to see that all assignments to the b_i's are valid except the typings of form $(\mathsf{Int}\rightarrow_1\mathsf{Int})\rightarrow_{b_2}(\mathsf{Int}\rightarrow_0\mathsf{Int})$ (as *twice*, given a non-strict function, doesn't produce a strict function); in other words we have the (sufficient and necessary) *constraint* $b_3 \geq b_1$.

$$\frac{\Gamma\vdash e : t, W}{\Gamma\vdash e : t', W'} \text{ if } t' \geq t, W' \geq W$$

$$\Gamma\vdash c : CT(c), \vec{1}$$

$$(\Gamma_1, (x : t), \Gamma_2)\vdash x : t, (\vec{1}, 0, \vec{1})$$

$$\frac{((x : t_1), \Gamma)\vdash e : t, (b, W)}{\Gamma\vdash \lambda x.e : t_1 \rightarrow_b t, W}$$

$$\frac{\Gamma\vdash e_1 : t_2 \rightarrow_b t_1, W_1 \;\; \Gamma\vdash e_2 : t_2, W_2}{\Gamma\vdash e_1 e_2 : t_1, W}$$
$$\text{if } W(x) = W_1(x) \sqcap (b \sqcup W_2(x)) \text{ for all } x$$

$$\frac{\Gamma\vdash e_1 : \mathsf{Bool}, W_1 \;\; \Gamma\vdash e_2 : t, W_2 \;\; \Gamma\vdash e_3 : t, W_3}{\Gamma\vdash (\mathsf{if}\ e_1\ e_2\ e_3) : t, W}$$
$$\text{if } W(x) = W_1(x) \sqcap (W_2(x) \sqcup W_3(x)) \text{ for all } x$$

$$\frac{((f : t), \Gamma)\vdash e : t, (b, W)}{\Gamma\vdash (\mathsf{rec}\ f\ e) : t, W}$$

Fig. 1. An inference system for strictness types.

Polarity: In the *twice* example, the polarity of b_1 is *negative* (as it occurs nested within an odd number of contravariant positions) and the polarity of b_2 and b_3 is *positive*. So the constraint produced ($b_3 \geq b_1$) yields a principal typing in the following sense: given a value for the negative strictness variable, it is trivial to find the set of possible values for the positive strictness variables. As we shall see later in this paper, this is a general phenomenon: if $\vec{b}^+$ are the positive strictness variables and $\vec{b}^-$ the negative strictness variables of the overall type (i.e. the type of the "whole" expression), it is possible to generate a "vector" constraint which is of form $\vec{b}^+ \geq g(\vec{b}^-)$ with g a monotone function. (In the *twice* example we have $(b_2, b_3) \geq (0, b_1)$, that is $g(b_1) = (0, b_1)$.)

Strictness Expressions. Monotone functions will be represented by *strictness expressions*, which are built from strictness variables, 0, 1, $\sqcap$ and $\sqcup$. A strictness expression s gives rise to a monotone function g with domain the free variables of s (and a little thought reveals that all monotone functions on finite domains can be represented by strictness expressions).

Approximating Solutions: We are not satisfied by only knowing the annotations of the arrows occurring in the *overall* type; we would like to know the typing of the *subexpressions* also (as this is needed by e.g. the translation algorithm in [Amt93a]). As an example, consider the term ((*twice id*) 7) (with *twice* as before and with $id = \lambda x.x$). The overall type of this term is Int; as we has seen the type of *twice* is of form $(\mathsf{Int}\rightarrow_{b_1}\mathsf{Int})\rightarrow_{b_2}(\mathsf{Int}\rightarrow_{b_3}\mathsf{Int})$; and the type of *id* is of form

$\mathsf{Int}\rightarrow_{b_4}\mathsf{Int}$. As before there is a constraint $b_3 \geq b_1$, and as the type of a function must match the type of its argument we also have a constraint $b_1 = b_4$. Just as the values of the positive variables of the overall type could be found from the negative variables of the overall type, we would like to be able to find the values of the "interior" strictness variables (i.e. b_1, b_2, b_3, b_4) from the negative variables of the overall type (of which there are none in the above case). A first attempt at achieving this would be to look for constants c_i such that the solutions are given by the constraints $b_i \geq c_i$. This, however, is not possible: we must clearly have $c_1 = c_4 = 0$ but the constraints $b_1 \geq 0$, $b_4 \geq 0$ are not sufficient for a solution as they forget the requirement that $b_1 = b_4$. In order to represent those requirements succintly we devise a symbol $\geqq$ which is "sandwiched" between $=$ and $\geq$ in a sense made precise by the following two definitions:

Definition 1. Let N be an extended constraint system (i.e. possibly containing $\geqq$). Let ϕ be a mapping from the strictness variables of N into $\{0, 1\}$. We say that ϕ is a *strong solution* to N iff ϕ is a solution to the system resulting from replacing all $\geqq$'s in N by $=$; and we say that ϕ is a *weak solution* to N iff ϕ is a solution to the system resulting from replacing all $\geqq$'s in N by $\geq$.

Notice that for constraint systems not containing $\geqq$, strong and weak solutions coincide. Also note that a strong solution is also a weak solution.

Definition 2. Let N_1 and N_2 be extended constraint systems. We say that N_2 approximates N_1, to be written $N_1 \lhd N_2$, iff

- all strong solutions to N_2 are strong solutions to N_1, and
- all weak solutions to N_1 are weak solutions to N_2.

Clearly $\lhd$ is reflexive and transitive; and referring back to the example it is easy to see that $\{b_1 = b_4\} \lhd \{b_1 \geqq 0, b_4 \geqq 0\}$. The right hand side contains less information than the left hand side, but as we can assume that we aim for minimal solutions this does not really matter.

Normalizing Constraints: In Sect. 3 we shall see that Fig. 1 can be rewritten into a system with judgments of form $\Gamma \vdash e : t, W, C$ where C is a set of constraints. The strictness variables occurring in C can be divided into two groups: those occurring in t or Γ, to be denoted $\vec{b}_1^+$ and $\vec{b}_1^-$, and those which do not occur in t or Γ (i.e. the *interior* variables, which occur further up in the proof tree), to be denoted $\vec{b}_0$ (we do not consider polarity here). As indicated above there exists a *normalization* algorithm which produces an extended constraint system N with the following property:

- $C \lhd N$;
- N is of form $\{\vec{b}_1^+ \geq \vec{s}_1,\ \vec{b}_0 \geqq \vec{s}_0\}$, with $\vec{s}_0$ and $\vec{s}_1$ being strictness expressions whose (free) variables belong to $\vec{b}_1^-$.

The algorithm is defined inductively in the inference tree – for details, see Sect. 5.

Manipulating Symbolic Expressions: From the above it follows that one can view type reconstruction as a function $T(e, \vec{b^-})$ which given an expression together with values for the negative strictness variables of its type returns the (minimal) values for the positive strictness variables (and the interior variables). A key point of our approach is that the normalization algorithm, which manipulates symbolic expressions, essentially does *partial evaluation* of T wrt. e – all computation dependent on e is done once only resulting in a piece of code which rapidly can be applied to many different values of $\vec{b^-}$.

This paradigm seems rather widely applicable; previous examples include [Con91] (for a first-order strictness analysis) and [Ros79] (for a flow-analysis of an imperative language).

Fixed Point Solving: An important technique to be applied during normalization is *fixed point iteration*. To see an example of this, consider the function

$$\lambda g.\lambda h.\mathsf{rec}\ f\ \lambda x.\mathsf{if}\ (g\ x)\ (h\ x)\ (f\ (-\ 1\ x))\ .$$

Let us *outline* how to type this function: we can assume that g has type $\mathsf{Int}\rightarrow_{b_g}\mathsf{Bool}$ and h has type $\mathsf{Int}\rightarrow_{b_h}\mathsf{Int}$. Likewise we can assume that the formal parameter f has type $\mathsf{Int}\rightarrow_{b_f}\mathsf{Int}$ and that the body of f has type $\mathsf{Int}\rightarrow_{b}\mathsf{Int}$. It is not too hard to see that we must have the constraint $b \geq b_g \sqcap (b_h \sqcup b_f)$ and due to the rule for recursion we must also have $b = b_f$, that is we have the constraint

$$b_f \geq b_g \sqcap (b_h \sqcup b_f) \tag{1}$$

The negative variables of the overall type are b_g and b_h; we are thus left with the task of expressing b_f in terms of these two strictness variables, i.e. to find an expression s where only b_g and b_h occur free such that the right hand side of (1) can be rewritten into s. We shall find s as the limit of the chain s_0,s_1,$s_2 \ldots$ where

- $s_0 = 0$;
- $s_1 = b_g \sqcap (b_h \sqcup s_0) = b_g \sqcap (b_h \sqcup 0) = b_g \sqcap b_h$;
- $s_2 = b_g \sqcap (b_h \sqcup s_1) = b_g \sqcap (b_h \sqcup (b_g \sqcap b_h)) = (b_g \sqcap b_h) \sqcup (b_g \sqcap b_g \sqcap b_h) = b_g \sqcap b_h$.

We see that the chain reaches its limit after one iteration, which was to be expected: since the lattice $\{0 \sqsubseteq 1\}$ has height one, for each value of b_g and b_h a fixed point will be reached in one step – we shall therefore reach a *representation* of all those fixed points after one step. Thus we can replace (1) by the constraint

$$b_f \geqq b_g \sqcap b_h$$

and have thus found out that f will be strict if either g or h is strict – this should come as no surprise. The reason for writing $\geqq$ instead of $\geq$ is that even though the right hand side represents the *least* fixed point, in general it does not hold that any value above this point is a fixed point – however, in this simple case where only one variable is present it would make no difference whether we write $\geq$ or $\geqq$.

For a formalization of the reasoning performed (implicitly) above, see Lemma 8 (and its proof).

One of the reasons for preferring type inference to abstract interpretation is that the latter approach usually involves (expensive) fixed point computation (whereas the former employs unification). Therefore it may seem suspicious that fixed point computation has crept into our type inference framework. On the other hand, it turns out that for a given expression the number of fixed point iterations we have to perform will be bounded by the sum of the sizes of the types of its subexpressions. Hence we can assume that for "typical" programs the cost of fixed point iteration will not be exorbitant.

Structure of Paper: The rest of the paper is organized as follows: in Sect. 3 we transform the inference system from Fig. 1 into one based on constraints; in Sect. 4 we list some tools to be used during the normalization process; and in Sect. 5 we give a detailed account of how to combine these tools into an algorithm for solving the constraints. Section 6 concludes.

3 Rewriting the Inference System

The first step towards getting a system more suitable for implementation is to "inline" the subsumption rule into (some of) the other rules[1]. The result is depicted in Fig. 2; it is not difficult to see that this system is equivalent to the one in Fig. 1 in the sense that the same judgments can be derived.

$$\Gamma \vdash c : t, \vec{1} \text{ if } t \geq CT(c)$$

$$(\Gamma_1, (x : t), \Gamma_2) \vdash x : t', (\vec{1}, b, \vec{1}) \text{ if } t' \geq t, b \geq 0$$

$$\frac{((x : t_1), \Gamma) \vdash e : t, (b, W)}{\Gamma \vdash \lambda x.e : t_1 \rightarrow_b t, W}$$

$$\frac{\Gamma \vdash e_1 : t_2 \rightarrow_b t_1, W_1 \;\; \Gamma \vdash e_2 : t_2, W_2}{\Gamma \vdash e_1 e_2 : t_1, W} \text{ if } W(x) \geq W_1(x) \sqcap (b \sqcup W_2(x)) \text{ for all } x$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Bool}, W_1 \;\; \Gamma \vdash e_2 : t_2, W_2 \;\; \Gamma \vdash e_3 : t_3, W_3}{\Gamma \vdash (\mathsf{if}\ e_1\ e_2\ e_3) : t, W} \text{ if } t \geq t_2, t \geq t_3, W(x) \geq W_1(x) \sqcap (W_2(x) \sqcup W_3(x)) \text{ for all } x$$

$$\frac{((f : t), \Gamma) \vdash e : t, (b, W)}{\Gamma \vdash (\mathsf{rec}\ f\ e) : t', W} \text{ if } t' \geq t$$

Fig. 2. The result of inlining the subsumption rule.

[1] This inlining of the non-structural rules into the structural rules is a very common technique for getting an implementable system; another (more complex) example can be found in [SNN92].

The next step is to make the annotations on arrows more explicit; at the same time distinguishing between positive/negative polarity. To this end we introduce some notation: if u is a (standard) type in $\mathcal{U}$, and if $\vec{b}^+$ and $\vec{b}^-$ are vectors of 0 or 1's, then $u[\vec{b}^+, \vec{b}^-]$ denotes u where all positive arrows are marked (from left to right) as indicated by $\vec{b}^+$ and where all negative arrows are marked (from left to right) as indicated by $\vec{b}^-$. More formally, we have

- $\mathsf{Base}[(),()] = \mathsf{Base}$;
- $(u_1 \to u_2)[(\vec{b}_1^+, b^+, \vec{b}_2^+), (\vec{b}_1^-, \vec{b}_2^-)] = u_1[\vec{b}_1^-, \vec{b}_1^+] \to_{b^+} u_2[\vec{b}_2^+, \vec{b}_2^-]$.

Example: with $u = ((\mathsf{Int}\to\mathsf{Int})\to\mathsf{Int})\to((\mathsf{Int}\to\mathsf{Int})\to\mathsf{Int})$, we have

$$u[(b_1,b_2,b_3),(b_4,b_5)] = ((\mathsf{Int}\to_{b_1}\mathsf{Int})\to_{b_4}\mathsf{Int})\to_{b_2}((\mathsf{Int}\to_{b_5}\mathsf{Int})\to_{b_3}\mathsf{Int}) \ .$$

If $\Gamma = ((x_1 : u_1) \ldots (x_n : u_n))$ then

$$\Gamma[(\vec{b}_1^-,\ldots,\vec{b}_n^-),(\vec{b}_1^+,\ldots,\vec{b}_n^+)] = ((x_1 : u_1[\vec{b}_1^-,\vec{b}_1^+]),\ldots,(x_n : u_n[\vec{b}_n^-,\vec{b}_n^+])) \ .$$

Fact 3. *$u[\vec{b}_1^+, \vec{b}_1^-] \le u[\vec{b}_2^+, \vec{b}_2^-]$ iff $\vec{b}_1^+ \le \vec{b}_2^+$ and $\vec{b}_2^- \le \vec{b}_1^-$ (pointwise).*

Using this notation the system from Fig. 2 rewrites into the system depicted[2] in Fig. 3. A remark about polarity: the turnstile $\vdash$ acts like an $\to$, so if $t = \Gamma(x)$ then positive positions in t will be considered as appearing negatively in the judgment, and vice versa. On the other hand, something appearing in the range of W is considered as being in positive position in the judgment. It is important to notice that the polarity of a strictness variable is an unambiguous notion, i.e. it is always the same in the premise of a rule as in the conclusion. We shall consistently use the convention that the polarity of a variable can be read from its superscript (as e.g. in $\vec{b}_1^+$).

It is straightforward to transform the system in Fig. 3 into one using *constraints* among the strictness variables, that is one with judgments of form $\Gamma \vdash e : t, W, C$ with C a set of constraints: just turn the side conditions into constraints. The resulting system is depicted in Fig. 4.

Recall our assumption that the underlying types have been given in advance. Then the inference system in Fig. 4 in an obvious way gives rise to a deterministic algorithm collecting a set of constraints. We are thus left with the task of *solving* those constraints. As indicated in Sect. 2 our approach will be to show how to *normalize* the constraints inductively in the proof tree ("on the fly"), thus demonstrating that the solutions have a particular form.

Before embarking on the normalization process, it will be convenient to describe some of the tools to be used.

[2] For space reasons we shall employ the convention that e.g. "$\Gamma \vdash e_1 : t_1, W_1$ and $e_2 : t_2, W_2$" means "$\Gamma \vdash e_1 : t_1, W_1$ and $\Gamma \vdash e_2 : t_2, W_2$". Also, with some abuse of notation, we shall write CT for the function (defined on the set of constants) into $\mathcal{U}$ which first applies the "old" CT and then "removes the annotations from the arrows".

$$\Gamma[\vec{b}^-, \vec{b}^+] \vdash c : CT(c)[\vec{b}_1^+, ()], \vec{b}_2^+ \quad \text{if } \vec{b}_1^+ \geq \vec{0}, \vec{b}_2^+ \geq \vec{1}$$

$$(\Gamma_1[\vec{b}_1^-, \vec{b}_1^+], (x : u[\vec{b}_2^-, \vec{b}_2^+]), \Gamma_2[\vec{b}_3^-, \vec{b}_3^+]) \vdash x : u[\vec{b}_4^+, \vec{b}_4^-], (\vec{b}_5^+, b_6^+, \vec{b}_7^+) \quad \text{if } \vec{b}_4^+ \geq \vec{b}_2^-, \vec{b}_2^+ \geq \vec{b}_4^-, \vec{b}_5^+ \geq 1, b_6^+ \geq 0, \vec{b}_7^+ \geq 1$$

$$\frac{((x : u_1[\vec{b}_1^-, \vec{b}_1^+]), \Gamma[\vec{b}^-, \vec{b}^+]) \vdash e : u[\vec{b}_2^+, \vec{b}_2^-], (b_3^+, \vec{b}_4^+)}{\Gamma[\vec{b}^-, \vec{b}^+] \vdash \lambda x.e : u_1[\vec{b}_1^-, \vec{b}_1^+] \rightarrow_{b_3^+} u[\vec{b}_2^+, \vec{b}_2^-], \vec{b}_4^+}$$

$$\frac{\Gamma[\vec{b}^-, \vec{b}^+] \vdash e_1 : u_2[\vec{b}_2^-, \vec{b}_2^+] \rightarrow_{b_3^+} u_1[\vec{b}_4^+, \vec{b}_4^-], \vec{b}_5^+ \text{ and } e_2 : u_2[\vec{b}_6^+, \vec{b}_6^-], \vec{b}_7^+}{\Gamma[\vec{b}^-, \vec{b}^+] \vdash e_1 e_2 : u_1[\vec{b}_4^+, \vec{b}_4^-], \vec{b}_8^+ \quad \text{if } \vec{b}_6^+ = \vec{b}_2^-, \vec{b}_2^+ = \vec{b}_6^-, \vec{b}_8^+ \geq \vec{b}_5^+ \sqcap (b_3^+ \sqcup \vec{b}_7^+)}$$

$$\frac{\Gamma[\vec{b}^-, \vec{b}^+] \vdash e_1 : \mathsf{Bool}, \vec{b}_3^+ \text{ and } e_2 : u[\vec{b}_4^+, \vec{b}_4^-], \vec{b}_5^+ \text{ and } e_3 : u[\vec{b}_6^+, \vec{b}_6^-], \vec{b}_7^+}{\Gamma[\vec{b}^-, \vec{b}^+] \vdash (\mathsf{if}\ e_1\ e_2\ e_3) : u[\vec{b}_8^+, \vec{b}_8^-], \vec{b}_9^+ \quad \text{if } \vec{b}_8^+ \geq \vec{b}_4^+, \vec{b}_4^- \geq \vec{b}_8^-, \vec{b}_8^+ \geq \vec{b}_6^+, \vec{b}_6^- \geq \vec{b}_8^-, \vec{b}_9^+ \geq \vec{b}_3^+ \sqcap (\vec{b}_5^+ \sqcup \vec{b}_7^+)}$$

$$\frac{((f : u[\vec{b}_1^-, \vec{b}_1^+]), \Gamma[\vec{b}^-, \vec{b}^+]) \vdash e : u[\vec{b}_2^+, \vec{b}_2^-], (b_3^+, \vec{b}_4^+)}{\Gamma[\vec{b}^-, \vec{b}^+] \vdash (\mathsf{rec}\ f\ e) : u[\vec{b}_5^+, \vec{b}_5^-], \vec{b}_4^+ \quad \text{if } \vec{b}_5^+ \geq \vec{b}_2^+, \vec{b}_2^- \geq \vec{b}_5^-, \vec{b}_2^+ = \vec{b}_1^-, \vec{b}_2^- = \vec{b}_1^+}$$

Fig. 3. Annotations on arrows made explicit.

4 Some Transformation Rules

Given a constraint system N, there are several ways to come up with a system N' such that $N \lhd N'$. Below we list some of these methods:

Fact 4. *Suppose N contains the constraints $\vec{b} \geq \vec{s}_1$, $\vec{b} \geq \vec{s}_2$. Then these can be replaced by the constraint $\vec{b} \geq \vec{s}_1 \sqcup \vec{s}_2$.*

Fact 5. *Suppose N contains the constraint $\vec{b} \geq \vec{s}$ or the constraint $\vec{b} = \vec{s}$. Then this can be replaced by $\vec{b} \geqq \vec{s}$.*

Lemma 6. *Suppose N contains the constraints $\vec{b}_1 \geq g_1(\vec{b}_2)$ and $\vec{b}_2 \geqq g_2(\vec{b}_3)$. Then the former constraint can be replaced by the constraint $\vec{b}_1 \geq g_1(g_2(\vec{b}_3))$, yielding a new constraint system N'.*

(In other words, if we have the constraints $\vec{b} \geq \vec{s}$ and $\vec{b}_i \geqq \vec{s}_i$ it is safe to replace the former constraint by $\vec{b} \geq \vec{s}[\vec{s}_i/\vec{b}_i]$.)

Proof. Let a strong solution to N' be given. Wrt. this solution, we have $\vec{b}_2 = g_2(\vec{b}_3)$, $\vec{b}_1 \geq g_1(g_2(\vec{b}_3))$ and hence also $\vec{b}_1 \geq g_1(\vec{b}_2)$ – thus this solution is also a strong solution to N.

$$\Gamma[\vec{b}^-,\vec{b}^+]\vdash c : CT(c)[\vec{b}_1^+,()],\vec{b}_2^+,\ \{\vec{b}_1^+\geq\vec{0},\vec{b}_2^+\geq\vec{1}\}$$

$$(\Gamma_1[\vec{b}_1^-,\vec{b}_1^+],(x:u[\vec{b}_2^-,\vec{b}_2^+]),\Gamma_2[\vec{b}_3^-,\vec{b}_3^+])\vdash x : u[\vec{b}_4^+,\vec{b}_4^-],(\vec{b}_5^+,b_6^+,\vec{b}_7^+),\ \{\vec{b}_4^+\geq\vec{b}_2^-,\vec{b}_2^+\geq\vec{b}_4^-,\vec{b}_5^+\geq 1,b_6^+\geq 0,\vec{b}_7^+\geq 1\}$$

$$\frac{((x:u_1[\vec{b}_1^-,\vec{b}_1^+]),\Gamma[\vec{b}^-,\vec{b}^+])\vdash e : u[\vec{b}_2^+,\vec{b}_2^-],(b_3^+,\vec{b}_4^+),C}{\Gamma[\vec{b}^-,\vec{b}^+]\vdash\lambda x.e : u_1[\vec{b}_1^-,\vec{b}_1^+]\rightarrow_{b_3^+}u[\vec{b}_2^+,\vec{b}_2^-],\vec{b}_4^+,C}$$

$$\frac{\Gamma[\vec{b}^-,\vec{b}^+]\vdash e_1 : u_2[\vec{b}_2^-,\vec{b}_2^+]\rightarrow_{b_3^+}u_1[\vec{b}_4^+,\vec{b}_4^-],\vec{b}_5^+,C_1 \text{ and } e_2 : u_2[\vec{b}_6^+,\vec{b}_6^-],\vec{b}_7^+,C_2}{\Gamma[\vec{b}^-,\vec{b}^+]\vdash e_1e_2 : u_1[\vec{b}_4^+,\vec{b}_4^-],\vec{b}_8^+,C_1\cup C_2\cup C \text{ where } C=\{\vec{b}_6^+=\vec{b}_2^-,\vec{b}_2^+=\vec{b}_6^-,\vec{b}_8^+\geq\vec{b}_5^+\sqcap(b_3^+\sqcup\vec{b}_7^+)\}}$$

$$\frac{\Gamma[\vec{b}^-,\vec{b}^+]\vdash e_1 : \mathsf{Bool},\vec{b}_3^+,C_1 \text{ and } e_2 : u[\vec{b}_4^+,\vec{b}_4^-],\vec{b}_5^+,C_2 \text{ and } e_3 : u[\vec{b}_6^+,\vec{b}_6^-],\vec{b}_7^+,C_3}{\Gamma[\vec{b}^-,\vec{b}^+]\vdash(\mathsf{if}\ e_1\ e_2\ e_3) : u[\vec{b}_8^+,\vec{b}_8^-],\vec{b}_9^+,C_1\cup C_2\cup C_3\cup C \text{ where } C=\{\vec{b}_8^+\geq\vec{b}_4^+,\vec{b}_4^-\geq\vec{b}_8^-,\vec{b}_8^+\geq\vec{b}_6^+,\vec{b}_6^-\geq\vec{b}_8^-,\vec{b}_9^+\geq\vec{b}_3^+\sqcap(\vec{b}_5^+\sqcup\vec{b}_7^+)\}}$$

$$\frac{((f:u[\vec{b}_1^-,\vec{b}_1^+]),\Gamma[\vec{b}^-,\vec{b}^+])\vdash e : u[\vec{b}_2^+,\vec{b}_2^-],(b_3^+,\vec{b}_4^+),C}{\Gamma[\vec{b}^-,\vec{b}^+]\vdash(\mathsf{rec}\ f\ e) : u[\vec{b}_5^+,\vec{b}_5^-],\vec{b}_4^+,C\cup C' \text{ where } C'=\{\vec{b}_5^+\geq\vec{b}_2^+,\vec{b}_2^-\geq\vec{b}_5^-,\vec{b}_2^+=\vec{b}_1^-,\vec{b}_2^-=\vec{b}_1^+\}}$$

Fig. 4. An inference system collecting constraints.

Now let a weak solution to N be given. Wrt. this solution, we have $\vec{b}_1\geq g_1(\vec{b}_2)$ and $\vec{b}_2\geq g_2(\vec{b}_3)$. Due to the monotonicity of g_1, we also have $\vec{b}_1\geq g_1(g_2(\vec{b}_3))$ showing that this solution is also a weak solution to N'. □

Lemma 7. *Suppose N contains the constraints $\vec{b}_1\geqq g_1(\vec{b}_2)$ and $\vec{b}_2\geqq g_2(\vec{b}_3)$. Then the former constraint can be replaced by the constraint $\vec{b}_1\geqq g_1(g_2(\vec{b}_3))$, yielding a new constraint system N'.*

(In other words, if we have the constraints $\vec{b}\geqq\vec{s}$ and $\vec{b}_i\geqq\vec{s}_i$ it is safe to replace the former constraint by $\vec{b}\geqq\vec{s}[\vec{s}_i/\vec{b}_i]$.)

Proof. Let a strong solution to N' be given. Wrt. this solution, we have $\vec{b}_2=g_2(\vec{b}_3)$, $\vec{b}_1=g_1(g_2(\vec{b}_3))$ and hence also $\vec{b}_1=g_1(\vec{b}_2)$ – thus this solution is also a strong solution to N.

Now let a weak solution to N be given. Wrt. this solution, we have $\vec{b}_1\geq g_1(\vec{b}_2)$ and $\vec{b}_2\geq g_2(\vec{b}_3)$. Due to the monotonicity of g_1, we also have $\vec{b}_1\geq g_1(g_2(\vec{b}_3))$ showing that this solution is also a weak solution to N'. □

Lemma 8. *Suppose N contains the constraint $\vec{b}_1\geq g(\vec{b}_1,\vec{b}_2)$. Then this can be replaced by the constraint $\vec{b}_1\geqq g'(\vec{b}_2)$ (yielding N'), where*

$$g'(\vec{b}_2)=\sqcup_k h^k(\vec{0})\ \textit{with}\ h(\vec{b})=g(\vec{b},\vec{b}_2)$$

(this just amounts to Tarski's theorem – notice that it will actually suffice with $|\vec{b}_1|$ *iterations).*

Proof. First suppose that we have a strong solution to N', i.e. $\vec{b}_1 = g'(\vec{b}_2)$. Since $\vec{b}_1 = \sqcup_k h^k(\vec{0})$, standard reasoning on the monotone and hence (as everything is finite) continuous function h tells us that $\vec{b}_1 = h(\vec{b}_1)$, i.e. $\vec{b}_1 = g(\vec{b}_1, \vec{b}_2)$. This shows that we have a strong solution to N.

Now suppose that we have a weak solution to N, i.e. $\vec{b}_1 \geq g(\vec{b}_1, \vec{b}_2)$. In order to show that this also is a weak solution to N', we must show that $\vec{b}_1 \geq g'(\vec{b}_2)$. This can be done by showing that $\vec{b}_1 \geq \vec{b}$ implies $\vec{b}_1 \geq h(\vec{b})$. But if $\vec{b}_1 \geq \vec{b}$ we have $\vec{b}_1 \geq g(\vec{b}_1, \vec{b}_2) \geq g(\vec{b}, \vec{b}_2) = h(\vec{b})$.

It is easy to see that g' is monotone. □

5 The Normalization Process

We shall examine the various constructs: for constants and variables the normalization process is trivial, as the constraints generated are of the required form. Neither does the rule for abstractions cause any trouble, since no new constraints are generated and since a strictness variable appears in the premise of the rule iff it appears in the conclusion (and with the same polarity). Now let us focus upon the remaining constructs, where for space reasons we omit conditionals (can be found in [Amt93b]).

Normalizing the Rule for Application. Recall the rule

$$\frac{\Gamma[\vec{b}^-, \vec{b}^+] \vdash e_1 : u_2[\vec{b}_2^-, \vec{b}_2^+] \rightarrow_{b_3^+} u_1[\vec{b}_4^+, \vec{b}_4^-], \vec{b}_5^+, C_1 \text{ and } e_2 : u_2[\vec{b}_6^+, \vec{b}_6^-], \vec{b}_7^+, C_2}{\Gamma[\vec{b}^-, \vec{b}^+] \vdash e_1 e_2 : u_1[\vec{b}_4^+, \vec{b}_4^-], \vec{b}_8^+, C_1 \cup C_2 \cup C}$$

where $C = \{\vec{b}_6^+ = \vec{b}_2^-, \vec{b}_2^+ = \vec{b}_6^-, \vec{b}_8^+ \geq \vec{b}_5^+ \sqcap (b_3^+ \sqcup \vec{b}_7^+)\}$.

Let $\vec{b}_0$ be the "extra" strictness variables of C_1 and $\vec{b}_1$ the extra strictness variables of C_2. Then we are inductively entitled to assume that there exist N_1,N_2 with $C_1 \lhd N_1$, $C_2 \lhd N_2$ such that N_1 takes the form

$$\vec{b}^+ \geq \vec{s}_a \; \vec{b}_2^+ \geq \vec{s}_2 \; b_3^+ \geq s_3 \; \vec{b}_4^+ \geq \vec{s}_4 \; \vec{b}_5^+ \geq \vec{s}_5 \; \vec{b}_0 \geqq \vec{s}_0$$

(where the free variables of the strictness expressions above belong to $\{\vec{b}^-, \vec{b}_2^-, \vec{b}_4^-\}$) and such that N_2 takes the form

$$\vec{b}^+ \geq \vec{s}_b \; \vec{b}_6^+ \geq \vec{s}_6 \; \vec{b}_7^+ \geq \vec{s}_7 \; \vec{b}_1 \geqq \vec{s}_1$$

(where the free variables of the strictness expressions above belong to $\{\vec{b}^-, \vec{b}_6^-\}$.)

Clearly $C_1 \cup C_2 \cup C \lhd N_1 \cup N_2 \cup C$. We shall now manipulate $N_1 \cup N_2 \cup C$ with the aim of getting something of the desired form.

The first step is to use Fact 4 to replace the two inequalities for $\vec{b}^+$ by one, and at the same time exploit that $\vec{b}_2^+ = \vec{b}_6^-$ and $\vec{b}_6^+ = \vec{b}_2^-$. As a result, we arrive at

$$\begin{array}{llll} \vec{b}^+ \geq \vec{s}_a \sqcup \vec{s}_b & \vec{b}_6^- \geq \vec{s}_2 & b_3^+ \geq s_3 & \vec{b}_4^+ \geq \vec{s}_4 \\ \vec{b}_5^+ \geq \vec{s}_5 & \vec{b}_0 \geqq \vec{s}_0 & \vec{b}_2^- \geq \vec{s}_6 & \vec{b}_7^+ \geq \vec{s}_7 \\ \vec{b}_1 \geqq \vec{s}_1 & \vec{b}_6^+ = \vec{b}_2^- & \vec{b}_2^+ = \vec{b}_6^- & \vec{b}_8^+ \geq \vec{b}_5^+ \sqcap (b_3^+ \sqcup \vec{b}_7^+) \end{array}$$

We now focus upon the pair of constraints

$$(\vec{b}_6^-, \vec{b}_2^-) \geq (\vec{s}_2, \vec{s}_6) \ .$$

According to Lemma 8, these can be replaced by the constraints

$$(\vec{b}_6^-, \vec{b}_2^-) \geqq (\vec{S}_2, \vec{S}_6)$$

where $(\vec{S}_2, \vec{S}_6)$ is given as the "limit" of the chain with elements $(\vec{s}_{2n}, \vec{s}_{6n})$ given by

$$\begin{aligned} (\vec{s}_{20}, \vec{s}_{60}) &= \vec{0} \\ (\vec{s}_{2(n+1)}, \vec{s}_{6(n+1)}) &= (\vec{s}_2, \vec{s}_6)[(\vec{s}_{2n}, \vec{s}_{6n})/(\vec{b}_6^-, \vec{b}_2^-)] \end{aligned}$$

This limit can be found as the k'th element, where $k = |(\vec{b}_2^-, \vec{b}_6^-)|$.

As $\vec{s}_2$ does not contain $\vec{b}_6^-$ and $\vec{s}_6$ does not contain $\vec{b}_2^-$, the above can be simplified into

$$\begin{array}{ll} \vec{s}_{20} = \vec{0} & \vec{s}_{2(n+1)} = \vec{s}_2[\vec{s}_{6n}/\vec{b}_2^-] \\ \vec{s}_{60} = \vec{0} & \vec{s}_{6(n+1)} = \vec{s}_6[\vec{s}_{2n}/\vec{b}_6^-] \end{array}$$

Our next step is to substitute in the new constraints for $\vec{b}_2^-$ and $\vec{b}_6^-$, using Lemma 6 and Lemma 7. We arrive at (after also having used Fact 5)

$$\begin{array}{lll} \vec{b}^+ \geq \vec{s}_a[\vec{S}_6/\vec{b}_2^-] \sqcup \vec{s}_b[\vec{S}_2/\vec{b}_6^-] & \vec{b}_6^- \geqq \vec{S}_2 & b_3^+ \geqq s_3[\vec{S}_6/\vec{b}_2^-] \\ \vec{b}_4^+ \geq \vec{s}_4[\vec{S}_6/\vec{b}_2^-] & \vec{b}_5^+ \geqq \vec{s}_5[\vec{S}_6/\vec{b}_2^-] & \vec{b}_0 \geqq \vec{s}_0[\vec{S}_6/\vec{b}_2^-] \\ \vec{b}_2^- \geqq \vec{S}_6 & \vec{b}_7^+ \geqq \vec{s}_7[\vec{S}_2/\vec{b}_6^-] & \vec{b}_1 \geqq \vec{s}_1[\vec{S}_2/\vec{b}_6^-] \\ \vec{b}_6^+ \geqq \vec{S}_6 & \vec{b}_2^+ \geqq \vec{S}_2 & \vec{b}_8^+ \geq \vec{b}_5^+ \sqcap (b_3^+ \sqcup \vec{b}_7^+) \end{array}$$

Finally we use Lemma 6 on the constraint for $\vec{b}_8^+$, arriving at

$$\begin{array}{lll} \vec{b}^+ \geq \vec{s}_a[\vec{S}_6/\vec{b}_2^-] \sqcup \vec{s}_b[\vec{S}_2/\vec{b}_6^-] & \vec{b}_6^- \geqq \vec{S}_2 & b_3^+ \geqq s_3[\vec{S}_6/\vec{b}_2^-] \\ \vec{b}_4^+ \geq \vec{s}_4[\vec{S}_6/\vec{b}_2^-] & \vec{b}_5^+ \geqq \vec{s}_5[\vec{S}_6/\vec{b}_2^-] & \vec{b}_0 \geqq \vec{s}_0[\vec{S}_6/\vec{b}_2^-] \\ \vec{b}_2^- \geqq \vec{S}_6 & \vec{b}_7^+ \geqq \vec{s}_7[\vec{S}_2/\vec{b}_6^-] & \vec{b}_1 \geqq \vec{s}_1[\vec{S}_2/\vec{b}_6^-] \\ \vec{b}_6^+ \geqq \vec{S}_6 & \vec{b}_2^+ \geqq \vec{S}_2 & \vec{b}_8^+ \geq \vec{s}_5[\vec{S}_6/\vec{b}_2^-] \sqcap (s_3[\vec{S}_6/\vec{b}_2^-] \sqcup \vec{s}_7[\vec{S}_2/\vec{b}_6^-]) \end{array}$$

This is of the desired form, as it is quite easy to check that the only strictness variables occurring in the expressions on the right hand sides are those occurring in $\vec{b}^-$ and in $\vec{b}_4^-$.

Normalizing the Rule for Recursion. Recall the rule

$$\frac{((f : u[\vec{b}_1^-, \vec{b}_1^+]), \Gamma[\vec{b}^-, \vec{b}^+]) \vdash e : u[\vec{b}_2^+, \vec{b}_2^-], (b_3^+, \vec{b}_4^+), C}{\Gamma[\vec{b}^-, \vec{b}^+] \vdash \mathsf{rec}\ f\ e : u[\vec{b}_5^+, \vec{b}_5^-], \vec{b}_4^+, C \cup C'}$$

where $C' = \{\vec{b}_1^- = \vec{b}_2^+, \vec{b}_2^- = \vec{b}_1^+, \vec{b}_5^+ \geq \vec{b}_2^+, \vec{b}_2^- \geq \vec{b}_5^-\}$.

Let $\vec{b}_0$ be the "extra" strictness variables of C. We are inductively entitled to assume that there exist N with $C \lhd N$ such that N takes the form

$$\vec{b}^+ \geq \vec{s}\ \vec{b}_1^+ \geq \vec{s}_1\ \vec{b}_2^+ \geq \vec{s}_2\ b_3^+ \geq \vec{s}_3\ \vec{b}_4^+ \geq \vec{s}_4\ \vec{b}_0 \geqq \vec{s}_0$$

(where the free variables of the strictness expressions above belong to $(\vec{b}^-, \vec{b}_1^-, \vec{b}_2^-)$.)

Clearly $C \cup C' \lhd N \cup C'$. We shall now manipulate $N \cup C'$ with the aim of getting something of the desired form.

The first step is to exploit that $\vec{b}_2^- = \vec{b}_1^+$ and $\vec{b}_1^- = \vec{b}_2^+$, and afterwards exploit Fact 4 to replace the two inequalities for $\vec{b}_2^-$ by one. We arrive at

$$\begin{array}{lll} \vec{b}^+ \geq \vec{s} & \vec{b}_2^- \geq \vec{s}_1 \sqcup \vec{b}_5^- & \vec{b}_1^- \geq \vec{s}_2 \\ b_3^+ \geq \vec{s}_3 & \vec{b}_4^+ \geq \vec{s}_4 & \vec{b}_0 \geqq \vec{s}_0 \\ \vec{b}_1^- = \vec{b}_2^+ & \vec{b}_2^- = \vec{b}_1^+ & \vec{b}_5^+ \geq \vec{b}_2^+ \end{array}$$

We now focus upon the pair of constraints

$$(\vec{b}_2^-, \vec{b}_1^-) \geq (\vec{s}_1 \sqcup \vec{b}_5^-, \vec{s}_2)\ .$$

According to Lemma 8, these can be replaced by the constraints

$$(\vec{b}_2^-, \vec{b}_1^-) \geqq (\vec{S}_1, \vec{S}_2)$$

where $(\vec{S}_1, \vec{S}_2)$ is given as the "limit" of the chain with elements $(\vec{s}_{1n}, \vec{s}_{2n})$ given by

$$\begin{aligned} (\vec{s}_{10}, \vec{s}_{20}) &= \vec{0} \\ (\vec{s}_{1(n+1)}, \vec{s}_{2(n+1)}) &= (\vec{s}_1 \sqcup \vec{b}_5^-, \vec{s}_2)[(\vec{s}_{1n}, \vec{s}_{2n})/(\vec{b}_2^-, \vec{b}_1^-)] \end{aligned}$$

This limit can be found as the k'th element, where $k = |(\vec{b}_1^-, \vec{b}_2^-)|$.

Our next step is to substitute in the new constraints for $\vec{b}_1^-$ and $\vec{b}_2^-$, using Lemma 6 and Lemma 7. We arrive at (after also having used Fact 5 to replace $=$ by $\geqq$)

$$\begin{array}{lll} \vec{b}^+ \geq \vec{s}[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] & \vec{b}_2^- \geqq \vec{S}_1 & \vec{b}_1^- \geqq \vec{S}_2 \\ b_3^+ \geq \vec{s}_3[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] & \vec{b}_4^+ \geq \vec{s}_4[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] & \vec{b}_0 \geqq \vec{s}_0[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] \\ \vec{b}_2^+ \geqq \vec{b}_1^- & \vec{b}_1^+ \geqq \vec{b}_2^- & \vec{b}_5^+ \geq \vec{b}_2^+ \end{array}$$

Finally we use Lemma 7 on the inequalities for $\vec{b}_2^+$ and $\vec{b}_1^+$, and subsequently use Lemma 6 on the inequality for $\vec{b}_5^+$. At the same time we replace some $\geq$ by $\geqq$, and arrive at

$$\begin{array}{lll}
\vec{b}^+ \geq \vec{s}[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] & \vec{b}_2^- \geqq \vec{S}_1 & \vec{b}_1^- \geqq \vec{S}_2 \\
b_3^+ \geqq \vec{s}_3[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] & \vec{b}_4^+ \geq \vec{s}_4[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] & \vec{b}_0 \geqq \vec{s}_0[(\vec{S}_1, \vec{S}_2)/(\vec{b}_2^-, \vec{b}_1^-)] \\
\vec{b}_2^+ \geqq \vec{S}_2 & \vec{b}_1^+ \geqq \vec{S}_1 & \vec{b}_5^+ \geq \vec{S}_2
\end{array}$$

This is of the desired form, as it is quite easy to check that the only strictness variables occurring in the expressions on the right hand sides are those occurring in $\vec{b}^-$ and in $\vec{b}_5^-$.

6 Concluding Remarks

We have shown how to convert an inference system for strictness analysis into an algorithm that works by manipulating symbolic expressions. It would be interesting to see if the method described in this paper could be used on other kinds of inference systems.

A work bearing strong similarities to ours is described in [CJ93]. Here a type inference system for binding time analysis is presented. The idea is to annotate the function arrows with the *program* variables that *perhaps* are needed to evaluate the function (by replacing "perhaps" by "surely" one could obtain a strictness analysis); hence this analysis is stronger than the one presented in this paper. The price to be paid for the increased precision is that the type reconstruction algorithm presented in [CJ93] requires the user to supply the *full* type for bound variables – in our framework, this essentially means that the user has to supply the values of the negative variables (and some of the positive ones too). A remark is made in the paper that to automatize this seems to require some sort of second-order unification (undecidable); at the cost of performing fixed point iteration we achieve the desired effect (but, it should be emphasized, for a less precise analysis).

The type inference algorithm has been implemented in Miranda[3]. The user interface is as follows: first the user writes a λ-expression, and provides the underlying type of the bound variables; then the system returns an inference tree where all arrows are annotated with strictness variables, together with a normalized set of constraints among those variables. Next the user provides the values of the constraint variables occurring negatively in the overall type; and finally the system produces an inference tree where all subexpressions are assigned the least possible strictness type. For a full documentation see [Amt93b].

Future work includes investigating the complexity of the algorithm just developed. This involves choosing an appropriate input size parameter (could be the size of the expression, the size of its type, the maximal size of a subexpression's type etc.). Also we have to think more carefully about a suitable representation of strictness expressions (in the implementation, a very naive representation was chosen).

[3] Miranda is a trademark of Research Software Limited.

Acknowledgements: The author is supported by the DART-project funded by the Danish Research Councils and by the LOMAPS-project funded by ESPRIT. The work reported here evolved from numerous discussions with Hanne Riis Nielson and Flemming Nielson. Also thanks to Jens Palsberg for useful feedback.

References

[Amt93a] Torben Amtoft. Minimal thunkification. In *3rd International Workshop on Static Analysis (WSA '93), September 1993, Padova, Italy*, number 724 in LNCS, pages 218–229. Springer-Verlag, 1993.

[Amt93b] Torben Amtoft. Strictness types: An inference algorithm and an application. Technical Report PB-448, DAIMI, University of Aarhus, Denmark, August 1993.

[BHA86] Geoffrey L. Burn, Chris Hankin, and Samson Abramsky. Strictness analysis for higher-order functions. *Science of Computer Programming*, 7:249–278, 1986.

[CJ93] Charles Consel and Pierre Jouvelot. Separate polyvariant binding-time analysis. Technical Report CS/E 93-006, Oregon Graduate Institute, Department of Computer Science and Engineering, 1993.

[Con91] Charles Consel. Fast strictness analysis via symbolic fixpoint iteration. Technical Report YALEU/DCS/RR-867, Yale University, September 1991.

[Hen91] Fritz Henglein. Efficient type inference for higher-order binding-time analysis. In John Hughes, editor, *International Conference on Functional Programming Languages and Computer Architecture*, number 523 in LNCS, pages 448–472. Springer-Verlag, August 1991.

[Jen91] Thomas P. Jensen. Strictness analysis in logical form. In John Hughes, editor, *International Conference on Functional Programming Languages and Computer Architecture*, number 523 in LNCS, pages 352–366. Springer-Verlag, August 1991.

[KM89] Tsung-Min Kuo and Prateek Mishra. Strictness analysis: A new perspective based on type inference. In *International Conference on Functional Programming Languages and Computer Architecture '89*, pages 260–272. ACM Press, September 1989.

[Myc80] Alan Mycroft. The theory of transforming call-by-need to call-by-value. In B. Robinet, editor, *International Symposium on Programming, Paris*, number 83 in LNCS, pages 269–281. Springer-Verlag, April 1980.

[Ros79] Barry K. Rosen. Data flow analysis for procedural languages. *Journal of the ACM*, 26(2):322–344, April 1979.

[SNN92] Kirsten Lackner Solberg, Hanne Riis Nielson, and Flemming Nielson. Inference systems for binding time analysis. In M. Billaud et al., editor, *Analyse statique, Bordeaux 92 (WSA '92)*, pages 247–254, September 1992.

[Wri91] David A. Wright. A new technique for strictness analysis. In *TAPSOFT '91*, number 494 in LNCS, pages 235–258. Springer-Verlag, April 1991.