# Stratified Bounded Affine Logic for Logarithmic Space

Ulrich Schöpp
Ludwig-Maximilians-Universität München
Oettingenstraße 67, D-80538 München, Germany

## Abstract

*A number of complexity classes, most notably* PTIME*, have been characterised by sub-systems of linear logic. In this paper we show that the functions computable in logarithmic space can also be characterised by a restricted version of linear logic. We introduce Stratified Bounded Affine Logic (*SBAL*), a restricted version of Bounded Linear Logic, in which not only the modality* ! *but also the universal quantifier is bounded by a resource polynomial. We show that the proofs of certain sequents in* SBAL *represent exactly the functions computable logarithmic space. The proof that* SBAL*-proofs can be compiled to* LOGSPACE *functions rests on modelling computation by interaction dialogues in the style of game semantics. We formulate the compilation of* SBAL*-proofs to space-efficient programs as an interpretation in a realisability model, in which realisers are taken from a Geometry of Interaction situation.*

## 1. Introduction

Sub-systems of Linear Logic [4] can be used to characterise complexity classes. The most prominent example of a complexity class captured by a sub-system of linear logic is PTIME, being characterised by many versions of linear logic such as Bounded Linear Logic [7], Light Linear Logic [6] and Soft Linear Logic [12]. In this paper we contribute a version of linear logic that characterises logarithmic space. We introduce Stratified Bounded Affine Logic (SBAL), which is strongly based on Bounded Linear Logic (BLL) [7]. Being a sub-system of linear logic, SBAL can naturally be viewed as a type system for the second order $\lambda$-calculus. We show that SBAL-typeable $\lambda$-terms capture the functions computable in logarithmic space.

Functions computable in logarithmic space are typically thought of as being implemented by algorithms that have access to a constant number of pointers into the input. In a functional programming setting, the use of pointers can be naturally represented by using higher-order functions. Suppose, for example, that the input to a LOGSPACE function is presented as a string. A pointer into a string is a natural number $i$ that points to the $i$th character in the string. One can then represent a string as a function $s\colon \mathbb{N} \to \Sigma$, where $\Sigma$ is the alphabet, and where $s(i)$ is either the $i$th character of the string or a blank symbol if $i$ points beyond the end of the string. With this representation of strings, a function from strings to strings is represented by a function of type $(\mathbb{N} \to \Sigma) \to (\mathbb{N} \to \Sigma)$. Algorithms that use a constant number of pointers into the input can be naturally represented as functions of this type. The pointers are represented by natural numbers and pointer lookup is given by function application. In this way, LOGSPACE algorithms can be naturally represented as functions of higher-order type. Stratified Bounded Affine Logic supports higher-order functions and the representation of LOGSPACE algorithms by higher-order functions.

Besides higher order functions, Stratified Bounded Affine Logic also supports a form polymorphism that can be used for the representation of inductive data types. Basic data types such as $\mathbb{N}$ and $\Sigma$ can be represented in the second-order $\lambda$-calculus by impredicative encodings. The natural numbers, for example, can be represented as the elements of the type $\forall\alpha.\,(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$. In most of the linear-logic-based type systems for capturing complexity classes, inductive data types are represented in this fashion. In these type systems, the function space is decomposed in a modality and a linear function space, but universal quantification remains unchanged. For example, in Bounded Linear Logic, natural numbers can be represented as elements of type $\forall\alpha.\,!_{y<p}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(p)$, where $p$ is a resource polynomial. In Stratified Bounded Affine Logic too, data types can be represented by an encoding in this style. However, to remain in logarithmic space, we need to impose a restriction on universal quantification, leading to a stratified form of universal quantification. Natural numbers can then be represented as elements of type $\forall\alpha \le q.\,!_{y<p}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(p)$. In this formula, both $p$ and $q$ are resource polynomials as in Bounded Linear Logic.

Stratified Bounded Affine Logic is strongly based on Bounded Linear Logic. Both logics contain the connec-

22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)
0-7695-2908-9/07 $25.00 © 2007 IEEE

tives $\otimes$ and $\multimap$ from intuitionistic linear logic and they both contain a restricted modality $!_{x<p}$ for the fine-grained control of duplication. The restricted modality $!_{x<p}$ is used in place of the standard modality ! from linear logic in order to restrict the uses of the duplication map $!A \multimap !A \otimes !A$. The intuition is that while $!A$ contains an unbounded number of copies of $A$, the formula $!_{x<p}A$ represents only $p$-many copies. For instance, $!_{x<2}A$ is isomorphic to $A[0/x] \otimes A[1/x]$. Thus, ! is replaced with a family of modalities $!_{x<p}$, one for each polynomial $p$. Then, the duplication map $!A \multimap !A \otimes !A$ becomes $!_{x<p+q}A \multimap !_{x<p}A \otimes !_{x<q}A$ and the bounds on the modalities restrict how often a datum can be duplicated in the course of an iteration. The main difference between BLL and SBAL lies in the rules for universal quantification. While in BLL the rules for the universal quantifiers are the same as in intuitionistic linear logic, SBAL only contains bounded universal quantification $\forall\alpha \leq p.A$.

While we view SBAL as a type system that identifies LOGSPACE-computable functions in the second-order $\lambda$-calculus, we cannot use a standard functional evaluation strategy to compute the outputs of functions. Above, we have argued that it is natural to represent LOGSPACE-algorithms by higher-order functions. With a standard evaluation strategy, like the ones in the PTIME-logics of loc. cit., the evaluation of such functions typically requires linear space. This is because we represent input and output words as functions, and linear space is needed to store them. In this paper we introduce an evaluation strategy using which higher-order representations of LOGSPACE-algorithms can be evaluated in logarithmic space.

We use ideas from game semantics to compile SBAL-typed higher-order programs to space-efficient programs. The central idea of game semantics is to represent computation by question/answer-dialogues between a number of entities. The compilation of SBAL takes the form of an interpretation in a model in which the computation is represented by dialogues that can be played in a space-efficient way. For the present purposes, one may think of a program as being modelled by a fixed message-passing network, in which questions and answers are being passed around as messages. One should think of the questions and answers as being of a basic type, such as the natural numbers. The result of a computation is determined by sending a number of questions to the network and interpreting the answers. In this paper we only consider networks that can be simulated by algorithms that store in memory only a constant number of questions and answers. Thus, an interpretation of SBAL in such a model amounts to a compilation to programs whose memory requirements are linear in size of questions and answers that may appear in the course of a computation. Given this translation, it is clear that when all questions and answers have logarithmic size in the size of the input, then the whole program can be evaluated in logarithmic space.

## 2. Stratified bounded affine logic

**Definition 1.** A *monomial* over a finite set $X$ of variables is a finite product $\prod_i \binom{x_i}{n_i}$, where each $x_i$ is a variable in $X$, where no two variables $x_i$ and $x_j$ for $i \neq j$ are the same, and where each $n_i$ is a non-negative integer.

$$\binom{x}{0} = 1 \qquad \binom{x}{n+1} = \binom{x}{n} \cdot \frac{x-n}{n+1}$$

A *resource polynomial* over $X$ is a polynomial with variables in $X$ and rational coefficients, which arises as a finite sum of monomials over $X$. We write $P(X)$ for the set of resource polynomials over $X$. We refer to the finite set $X$ as the set of resource variables.

For $p,q \in P(X)$ write $p \leq q$ if the polynomial $q-p$ is a resource polynomial. It is the case that if $p$ and $q$ are resource polynomials over $X$, then so are $p+q$, $p \cdot q$, $p[q/x]$ and $\sum_{x<q} p$. The last point, that resource polynomials are closed under bounded summation, is the reason for using polynomials with binomial coefficients, as opposed to, say, polynomials with natural numbers as coefficients. Notice, however, that each polynomial with natural coefficients and variables in $X$ is a resource polynomial over $X$.

An *environment* $\eta$ *on* $X$ is a function that maps each $x \in X$ to a natural number. We write $V(X)$ for the set of environments on $X$. For $p \in P(X)$ we write $p[\eta]$ for the natural number obtained by evaluating $p$ at $\eta$.

In addition to ordinary resource polynomials we also use resource polynomials with a number of bound variables, such as e.g. $p = \lambda x.x \cdot x + y$.

**Definition 2.** Define the set $P_k(X)$ of polynomials with $k$ bound variables inductively by $P_0(X) = P(X)$ and

$$P_{k+1}(X) = \{\lambda x.\, p \mid p \in P_k(X \cup \{x\}), x \notin X\}.$$

For $p \in P_{k+1}(X)$ and $q \in P(X)$, we write $p(q)$ for the polynomial $p'[q/x] \in P_k(X)$, where $p = \lambda x.p'$. An ordering on the elements of $P_k(X)$ is given by

$$p \leq q \iff \forall r_1,\ldots,r_k \in P(X).\, p(r_1,\ldots,r_k) \leq q(r_1,\ldots,r_k).$$

The syntax for the formulae of SBAL is defined by

$$A ::= \alpha(\vec{p}) \mid A \otimes A \mid A \multimap A \mid !_{x<p}A \mid \forall\alpha \leq p\colon \mathsf{Fm}_k.A,$$

where the resource variable $x$ in $!_{x<p}A$ binds any occurrence of $x$ in $A$ and the second-order variable $\alpha$ in $\forall\alpha \leq p\colon \mathsf{Fm}_k.A$ binds any occurrence of $\alpha$ in $A$. With the exception of the universal quantifier, the formulae are as in Bounded Linear Logic. In particular, each second-order variable $\alpha$ is applied to a number of resource polynomials $\vec{p}$. The logic is set up so that each second-order $\alpha$ variable has a fixed

22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)
0-7695-2908-9/07 $25.00 © 2007 IEEE

arity, meaning that $\alpha$ must be applied to previously specified, fixed number of resource polynomials. Compared to Bounded Linear Logic, the formula $\forall\alpha \leq p\colon \mathsf{Fm}_k.A$ for universal quantification has two additional parameters $k$ and $p$. The parameter $k$ is there for technical convenience only. It represents the arity of the variable $\alpha$. More important is the presence of the polynomial $p$ with $k$ bound variables. It serves as a size bound on $\alpha$.

Next we define the judgement $\Sigma \vdash A \leq p\colon \mathsf{Fm}$, which declares $A$ to be a well-formed formula of size $p$. In this judgement, $\Sigma$ is a *second-order context* of declarations $\alpha \leq q\colon \mathsf{Fm}_k$, declaring $\alpha$ to be a second-order variable of arity $k$ with size-bound $q \in P_k(X)$. In the logic we allow only well-formed formulae. In this way, we enforce that second-order variables are used according to the arities declared in $\Sigma$. The rules for the judgement $\Sigma \vdash A \leq p\colon \mathsf{Fm}$ appear in Figure 1. The meaning of the size polynomials in this figure will become clear when we give a semantic interpretation of SBAL in Section 3. For the time being, it should suffice to say that the interpretation of a formula will comprise a set of questions and a set of answers and the size-polynomial bounds the size of the elements of these two sets.

We write $\Sigma \vdash A\colon \mathsf{Fm}$ instead of $\Sigma \vdash A \leq p\colon \mathsf{Fm}$ if we are not interested in the polynomial $p$. Also, since in a declaration $\alpha \leq p\colon \mathsf{Fm}_k$ the number $k$ can be inferred from the polynomial $p$, we often write just $A \leq p$ for it. In particular, we write $\forall\alpha \leq p.A$ instead of $\forall\alpha \leq p\colon \mathsf{Fm}_k.A$. Moreover, we write $!_p A$ for $!_{x<p}A$ if $x$ does not appear free in $A$.

Like Bounded Linear Logic, Stratified Bounded Affine Logic allows for monotonicity reasoning on resource polynomials. For example, if $p \leq q$ holds then the implications $\alpha(p) \multimap \alpha(q)$ and $!_q A \multimap !_p A$ are derivable. For the formulation of such monotonicity reasoning, we define notions of *positive* and *negative* occurrences of a resource variable in a formula as follows. A resource variable $x$ occurs positively in $\alpha(p_1,\ldots,p_n)$ if it occurs in one or more of the polynomials $p_1,\ldots,p_n$. If $x$ occurs positively (resp. negatively) in $A$ or $B$ then it occurs positively (resp. negatively) in $A \otimes B$. If $x$ occurs positively (resp. negatively) in $A$ or negatively (resp. positively) in $B$ then it occurs negatively (resp. positively) in $A \multimap B$. Any resource variable $x$ that occurs in $A$ occurs in $!_{x<p}A$ with the same polarity as in $A$. Any resource variable $x$ that occurs in $p$ occurs negatively in $!_{x<p}A$. Finally, any variable $x$ that occurs in $A$ occurs in $\forall\alpha \leq p\colon \mathsf{Fm}_k.A$ with the same polarity as in $A$. Note that a variable $x$ can occur both positively and negatively in a formula. We say that a set $X$ *is positive for a formula* $A$ if each $x \in X$ occurs only positively in $A$.

With these definitions, we are ready to define the inference rules of SBAL, which appear in Figure 2. The judgement $\Sigma \vdash A \leq B$ expresses that the formulae $A$ and $B$ differ only in their resource polynomials and that $B$ subsumes $A$. The judgement $\Sigma \mid \Gamma \vdash B$, in which $\Gamma$ is a list of formulae and $B$ is a formula, expresses logical entailment of $B$ from $\Gamma$.

We note that all the modality rules of Bounded Linear Logic are available in Stratified Bounded Affine Logic. In addition, SBAL contains the rule (FUNCTORIALITY). This is not an essential extension to the modality rules, as Hofmann & Scott [9] have shown that this rule can be added without harm to BLL as well. Furthermore, SBAL contains an unrestricted weakening rule, which, for example, is useful to prove $!_{2p}(A \otimes B) \multimap !_p A \otimes !_p B$. Again, this is not an essential difference to BLL, as unrestricted weakening can be added to BLL without affecting PTIME-characterisation results [9]. The main essential difference between SBAL and BLL lies in the rules for universal quantification. In particular, rule ($\forall$-L) allows us to instantiate a universal quantifier $\forall\alpha \leq p.A$ only with formulae whose size polynomial does not exceed $p$. In this way, universal quantification SBAL is stratified by size polynomials.

The inference rules of SBAL can be viewed as typing rules for the second-order $\lambda$-calculus (System F). Each SBAL formula can be translated to a formula of System F by removing the modality $!_{x<p}$, by replacing $\multimap$ with $\Rightarrow$ and $\forall\alpha \leq p$ with $\forall\alpha$. Thus, the rules in Figure 2 translate to typing rules in System F. We refer to the System F type obtained from a SBAL formula $A$ as the *underlying set* of $A$. By translating a derivation in SBAL to a derivation in System F, we can assign a $\lambda$-term to each derivable sequent. We call this $\lambda$-term the *underlying function* of a sequent.

## 2.1. Examples

**2.1.1. Representing inductive data types.** We give a few examples to show that while universal quantification in SBAL is stratified, it can nevertheless be used to represent inductive data types in the style of impredicative encodings. A few examples appear in Figure 3. The formula $\mathbf{F}_k^p$ represents a finite set with $k$ elements; the formula $\mathbf{N}_x^p$ represents natural numbers not larger than $x$; and $\mathbf{S}_x^p$ represents binary words of length at most $x$. The definitions of $\mathbf{F}_k^p$, $\mathbf{N}_x^p$ and $\mathbf{S}_x^p$ differ from corresponding impredicative encodings in BLL only in the presence of a bound on the universal quantifier.

To give an example of how inductive data types can be used in SBAL, we define a few functions for the type of natural numbers $\mathbf{N}_x^p$. The constant zero of type $\mathbf{N}_x^p$ and the successor function of type $\mathbf{N}_x^p \multimap \mathbf{N}_{x+1}^p$, where $x$ and $p$ are arbitrary, can be defined as in BLL.

**Addition.** We show that addition on $\mathbf{N}_x^p$ can be defined in SBAL. For the definition we need the size polynomial of $\mathbf{N}_x^p$, which is easily derived using the rules from Figure 1:

$$\vdash \mathbf{N}_x^p \leq 2(2((x+2(p(x)+p(x+1)))^2+p(0))+p(x))$$

Write $n(p,x)$ for the polynomial in this sequent. With this notation, addition is defined by the derivation in Figure 4.

$$\frac{\forall 1 \le i \le k.\, q_i \in P(X)}{\Sigma,\, \alpha \le p : \mathsf{Fm}_k \vdash \alpha(q_1,\ldots,q_k) \le p(q_1,\ldots,q_k) : \mathsf{Fm}}$$

$$\frac{\Sigma \vdash A \le p : \mathsf{Fm} \qquad \Sigma \vdash B \le q : \mathsf{Fm}}{\Sigma \vdash (A \otimes B) \le 2\cdot(p+q) : \mathsf{Fm}} \qquad \frac{\Sigma \vdash A \le p : \mathsf{Fm} \qquad \Sigma \vdash B \le q : \mathsf{Fm}}{\Sigma \vdash (A \multimap B) \le 2\cdot(p+q) : \mathsf{Fm}}$$

$$\frac{\Sigma,\, \alpha \le p : \mathsf{Fm}_k \vdash A \le q : \mathsf{Fm}}{\Sigma \vdash (\forall \alpha \le p : \mathsf{Fm}_k.A) \le q : \mathsf{Fm}} \qquad \frac{\Sigma \vdash A \le q : \mathsf{Fm}}{\Sigma \vdash (!_{x<p}A) \le (p + q[p/x] + 1)^2 : \mathsf{Fm}}$$

$$\frac{\Sigma \vdash A \le p : \mathsf{Fm} \qquad p \le q}{\Sigma \vdash A \le q : \mathsf{Fm}}$$

**Figure 1. Formulae and their size polynomials**

**Monotonicity Rules.**

$$\frac{\Sigma \vdash \alpha(p_1,\ldots,p_k) : \mathsf{Fm} \qquad \Sigma \vdash \alpha(q_1,\ldots,q_k) : \mathsf{Fm} \qquad \forall 1 \le i \le k.\, p_i \le q_i}{\Sigma \vdash \alpha(p_1,\ldots,p_k) \le \alpha(q_1,\ldots,q_k)}$$

$$\frac{\Sigma \vdash A \le A' \qquad \Sigma \vdash B \le B'}{\Sigma \vdash A \otimes B \le A' \otimes B'} \qquad \frac{\Sigma \vdash A \le A' \qquad \Sigma \vdash B' \le B}{\Sigma \vdash A \multimap B \le A' \multimap B'}$$

$$\frac{\Sigma \vdash A \le A' \qquad p \le q}{\Sigma \vdash\ !_{x<q}A \le\ !_{x<p}A'} \qquad \frac{\Sigma,\, \alpha \le p : \mathsf{Fm}_k \vdash A \le A'}{\Sigma \vdash (\forall \alpha \le p : \mathsf{Fm}_k.A) \le (\forall \alpha \le p : \mathsf{Fm}_k.A')}$$

**Logical Rules.**

$$(\textsc{Axiom})\ \frac{\Sigma \vdash A \le A'}{\Sigma \mid A \vdash A'} \qquad (\textsc{Cut})\ \frac{\Sigma \mid \Gamma \vdash A \qquad \Sigma \mid \Delta, A \vdash B}{\Sigma \mid \Gamma, \Delta \vdash B}$$

$$(\otimes\text{-L})\ \frac{\Sigma \mid \Gamma, A, B \vdash C}{\Sigma \mid \Gamma, A \otimes B \vdash C} \qquad (\otimes\text{-R})\ \frac{\Sigma \mid \Gamma \vdash A \qquad \Sigma \mid \Delta \vdash B}{\Sigma \mid \Gamma, \Delta \vdash A \otimes B}$$

$$(\multimap\text{-L})\ \frac{\Sigma \mid \Gamma \vdash A \qquad \Sigma \mid \Delta, B \vdash C}{\Sigma \mid \Gamma, \Delta, A \multimap B \vdash C} \qquad (\multimap\text{-R})\ \frac{\Sigma \mid \Gamma, A \vdash B}{\Sigma \mid \Gamma \vdash A \multimap B}$$

$$(\forall\text{-L})\ \frac{\Sigma \vdash B \le p(\vec{x}) \qquad \Sigma \mid \Gamma, A[\lambda \vec{x}.B/\alpha] \vdash C}{\Sigma \mid \Gamma, \forall \alpha \le p : \mathsf{Fm}_k.A \vdash C}\ \vec{x} \text{ fresh for } p \text{ and } \vec{x} \text{ positive for } B$$

$$(\forall\text{-R})\ \frac{\Sigma,\, \alpha \le p \mid \Gamma \vdash A}{\Sigma \mid \Gamma \vdash \forall \alpha \le p : \mathsf{Fm}_k.A}\ \alpha \notin \Gamma$$

$$(\textsc{Weakening})\ \frac{\Sigma \vdash A : \mathsf{Fm} \qquad \Sigma \mid \Gamma \vdash B}{\Sigma \mid \Gamma, A \vdash B} \qquad (\textsc{Dereliction})\ \frac{\Sigma \mid \Gamma, A[0/x] \vdash B}{\Sigma \mid \Gamma, !_{x<1+w}A \vdash B}$$

$$(\textsc{Contraction})\ \frac{\Sigma \mid \Gamma, !_{x<p}A, !_{y<q}A[p+y/x] \vdash B}{\Sigma \mid \Gamma, !_{x<p+q+w}A \vdash B} \qquad (\textsc{Functoriality})\ \frac{\Sigma \mid \Gamma \vdash A}{\Sigma \mid\ !_{x<p}\Gamma \vdash\ !_{x<p}A}$$

$$(\textsc{Storage})\ \frac{\Sigma \mid \Gamma,\ !_{x<p}!_{z<q(x)}A[z + \sum_{u<x} q(u)/y] \vdash B}{\Sigma \mid \Gamma,\ !_{y<\sum_{x<p} q(x)}A \vdash B}$$

**Figure 2. Inference rules of** SBAL

22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)
0-7695-2908-9/07 $25.00 © 2007 IEEE

$$\mathbf{F}^p_k := \forall \alpha \le p.\, \overbrace{\alpha \multimap \ldots \multimap \alpha}^{k \text{ times}} \multimap \alpha, \qquad \text{where } k \in \mathbb{N}$$
$$\mathbf{N}^p_x := \forall \alpha \le p\colon \mathsf{Fm}_1.\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x)$$
$$\mathbf{S}^p_x := \forall \alpha \le p\colon \mathsf{Fm}_1.\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x)$$

**Figure 3. Some inductive data types**

$$(\forall\text{-L})\ \frac{\dfrac{\vdots}{\vdash \mathbf{N}^p_{x+z} \le n(p,x+z)} \qquad (\text{FUNCTORIALITY})\ \dfrac{\dfrac{\text{successor}}{\vdash \mathbf{N}^p_{x+z} \multimap \mathbf{N}^p_{x+z+1}}}{\vdash\ !_{z<y}(\mathbf{N}^p_{x+z} \multimap \mathbf{N}^p_{x+z+1})}}{\dfrac{\text{evident applications}}{\mathbf{N}^p_x,\ !_{z<y}(\mathbf{N}^p_{x+z} \multimap \mathbf{N}^p_{x+z+1}) \multimap \mathbf{N}^p_x \multimap \mathbf{N}^p_{x+y} \vdash \mathbf{N}^p_{x+y}}}{\mathbf{N}^p_x,\ \mathbf{N}^{\lambda z.n(p,x+z)}_y \vdash \mathbf{N}^p_{x+y}}$$

**Figure 4. Addition on $\mathbf{N}^p_x$**

**Coercion.** In Bounded Linear Logic there exists a coercion from $\mathbf{N}_x$ to $!_p\mathbf{N}_x$. A similar coercion also exists in SBAL, where we can go from $\mathbf{N}^{\lambda x.(p+n(q,x)+1)^2}_x$ to $!_p\mathbf{N}^q_x$. This coercion is defined by instantiating the universal quantifier in $\mathbf{N}^{\lambda x.(p+n(q,x)+1)^2}_x$ with $!_p\mathbf{N}^q_x$ and then applying successor and zero to the result. The universal quantifier can be instantiated thus because $\vdash\ !_p\mathbf{N}^q_x \le (p+n(q,x)+1)^2$ is derivable.

**Monotonicity Reasoning.** If $q \ge p$ holds then there exists a proof of $\mathbf{N}^q_x \vdash \mathbf{N}^p_x$, whose underlying function is the identity function. The derivation is shown below. In it we write $B$ for $!_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x)$.

$$(\forall\text{-R})\ \frac{(\forall\text{-L})\ \dfrac{\dfrac{\dfrac{}{\alpha \le p \vdash \alpha \le p}}{\alpha \le p \vdash \alpha \le q}\ p \le q \qquad (\text{AX})\ \dfrac{}{\alpha \le p \mid B \vdash B}}{\alpha \le p \mid \mathbf{N}^q_x \vdash B}}{\mathbf{N}^q_x \vdash \mathbf{N}^p_x}$$

We believe that the other direction $\mathbf{N}^p_x \vdash \mathbf{N}^q_x$ is *not* derivable. However, as in BLL, the sequent $\mathbf{N}^r_p \vdash \mathbf{N}^r_q$ has a derivation with the identity as the underlying function.

**2.1.2. A simple logspace function.** We give a very simple example to show how we represent LOGSPACE algorithms by higher-order functions in SBAL. In this example, we consider LOGSPACE algorithms whose input and output are binary strings. Binary strings of length at most $x$ can be represented by the formula $\mathbf{N}^p_x \multimap \mathbf{F}^q_3$ as follows. The underlying set of this formula is $\mathbb{N} \to \{0,1,*\}$. A proof of $\mathbf{N}^p_x \multimap \mathbf{F}^q_3$ represents the word $b_1 \ldots b_n \in \{0,1\}^*$ if its underlying function $f\colon \mathbb{N} \to \{0,1,*\}$ satisfies $f(i) = b_i$ for $1 \le i \le n$ and $f(i) = *$ for $i > n$. With this representation of words, a LOGSPACE-algorithm $l\colon \{0,1\}^* \to \{0,1\}^*$ can be represented as a proof of a formula $(\mathbf{N}_{p(x)} \multimap \mathbf{F}_3) \multimap (\mathbf{N}_{q(x)} \multimap \mathbf{F}_3)$ with appropriate superscripts, such that, for all $b \in \{0,1\}^*$, the underlying function of this proof maps any function representing $b$ to a function representing $l(b)$.

A very simple example for such a function is the shift function mapping $b_1 \ldots b_n$ to $b_2 \ldots b_n$. It is represented by the proof below.

$$(\multimap\text{-R})\ \frac{(\multimap\text{-L})\ \dfrac{\dfrac{\text{successor}}{\mathbf{N}^q_x \vdash \mathbf{N}^q_{x+1}} \qquad (\text{AXIOM})\ \dfrac{}{\mathbf{F}^q_3 \vdash \mathbf{F}^q_3}}{\mathbf{N}^q_{x+1} \multimap \mathbf{F}^q_3,\ \mathbf{N}^s_x \vdash \mathbf{F}^q_3}}{\vdash (\mathbf{N}^q_{x+1} \multimap \mathbf{F}^q_3) \multimap (\mathbf{N}^q_x \multimap \mathbf{F}^q_3)}$$

For the analysis of the space-usage of a function $l\colon \{0,1\}^* \to \{0,1\}^*$ represented by a proof of type $(\mathbf{N}_{p(x)} \multimap \mathbf{F}_3) \multimap (\mathbf{N}_{q(x)} \multimap \mathbf{F}_3)$, notice that $p(x)$ is an upper bound on the length of the input string. Hence, for a LOGSPACE-evaluation of the representation of $l$ we can use space $O(\log(p(x))) = O(\log(x))$. Therefore, the reader should think of space available to the computations as being logarithmic in the values of the resource variables. In particular, the elements of $\mathbf{N}_x$, i.e. the natural numbers less than $x$, can be stored in memory by using binary encoding.

## 2.2. Skewed iteration

Above we have shown that addition on $\mathbf{N}^p_x$ can be defined in SBAL in the usual manner. Other functions, such as multiplication can be defined analogously. However, the iteration principle embodied by $\mathbf{N}^p_x$ is not quite sufficient to define all the usual basic functions on natural numbers in the normal way. The subtraction function, for example, would usually (e.g. in BLL) be defined by iterating the predecessor function. In terms of the underlying $\lambda$-term, subtraction

would be defined by $minus{:=}\lambda a:\mathbf{N}.\lambda b:\mathbf{N}.(b\ pred\ a)$. To type this definition in SBAL, we must find a formula $A(x)$ such that the predecessor function can be given the type $!_{z<y}(A(z) \multimap A(z+1))$. If the predecessor function is defined in the usual way in SBAL, it will have type $\mathbf{N}_x^q \multimap \mathbf{N}_x^p$ for $q > p$. Using rule (FUNCTORIALITY) we can give it type $!_{z<y}(\mathbf{N}_x^q \multimap \mathbf{N}_x^p)$. However, due to $p$ and $q$ being different, this does not quite have the required form, and there appears to be no way to bring this type into the required form. So, it does not appear that subtraction can be defined in SBAL in the usual way. In BLL, on the other hand, no polynomials appear in the superscript, and the type of the predecessor function does have the required form.

For explaining the reason for the weakness of the iteration principle in SBAL, let us consider the space needed to evaluate an iteration. Suppose we want to represent a LOGSPACE algorithm in SBAL. Let the resource variable $x$ denote the length of the input to this algorithm. The formula $\mathbf{N}_{p(x)}^q$ represents natural numbers not greater than $p(x)$. Using a binary encoding, a LOGSPACE algorithm can store such numbers in memory. The formula $\mathbf{S}_{p(x)}^q$, on the other hand, represents binary words of length up to $p(x)$. A LOGSPACE algorithm does not have enough space to store such words completely. Yet, the iteration principle for natural numbers allows us to derive $\mathbf{N}_x^q \multimap !_x(A \multimap A) \multimap A \multimap A$ for both $A = \mathbf{N}_x^p$ and $A = \mathbf{S}_x^p$. Since values of type $\mathbf{S}_x^p$ cannot be wholly stored in memory, iteration must be implemented without storing intermediate values in memory.

One consequence of the fact that in the implementation of iteration we cannot, in general, store intermediate values is that we cannot apply the standard tail-call optimisation. It is standard to implement an iteration by a while loop of the following form.

```
v := g;
while n > 0 do
  v := f(v);
  n := n − 1
done;
return(v)
```

This implementation of iteration by a while loop depends on being able to store the value $v$ in memory. Hence, in general, iteration in SBAL cannot be implemented using tail-call optimisation. For this reason, iteration of the predecessor function above cannot be allowed. Indeed, iteration of the predecessor function without tail-call optimisation would lead beyond logarithmic space. Nevertheless, there are instances of the iteration principle that can be implemented by a while loop, namely when the result type contains only small values that fit into memory, e.g. $A = \mathbf{N}_{p(x)}^q$.

We now introduce the principle of *skewed iteration*, which makes available in the logic that for certain formulae that represent small values, iteration can be implemented using a while loop as above. This iteration principle allows one to iterate the predecessor function above, whose type $\mathbf{N}_x^q \multimap \mathbf{N}_x^p$ is skewed in its superscripts.

**Definition 3.** The set of *small data types* is defined inductively by: if $p \geq k$, then $\mathbf{F}_k^p$ is a small data type; if $q \geq \lambda x.x$, then $\mathbf{N}_p^q$ is a small data type; and if both $A$ and $B$ are small data types, then so is $A \otimes B$.

The rule for skewed iteration, which in essence allows one to ignore the superscripts of small data types in an iteration, is defined below. In it we write $\mathscr{I}$ for the canonical translation of SBAL-formulae to formulae in BLL.

$$(\text{SKEW})\ \frac{\begin{array}{l} A, B, C \text{ and } D \text{ small data types,} \\ p \text{ and } q \text{ are the polynomials from Lemma 10,} \\ \mathscr{I}(A) = E[y/x] \qquad \mathscr{I}(B) = E[y+1/x] \\ \mathscr{I}(C) = E[0/x] \qquad \mathscr{I}(D) = E \end{array}}{\Sigma \mid \Gamma \vdash \mathbf{N}_x^p \multimap !_q !_{y<x}(A \multimap B) \multimap !_q C \multimap D}$$

We write SBAL+(SKEW) for SBAL extended with this rule.

## 2.3. Complexity

In this section we state the main results of this paper. The first result is that functions from $\mathbf{N}_x$ to $\mathbf{N}_{p(x)}$ are computable in linear space. This corresponds to the intuition that values of type $\mathbf{N}_x$ represent pointers. Since pointers typically have logarithmic size, we expect the space usage of our programs to be linear in the size of pointers.

**Theorem 1.** *If $\vdash \mathbf{N}_x^q \multimap \mathbf{N}_{p(x)}^r$ is derivable and $r \geq \lambda x.x$ holds, then the underlying function of this proof is computable in linear space, when viewed as a function on natural numbers in binary representation.*

To formulate the result that all LOGSPACE-computable functions can be represented in SBAL, we make precise the representation of LOGSPACE-functions by higher-order functions, as outlined in the Introduction. We do this for functions on binary strings in the following definition.

**Definition 4.** Let $p \in P_1(X)$, $q \in P(X)$ and $r \in P_1(X)$ be resource polynomials satisfying $p \geq \lambda x.x$, $r \geq \lambda x.x$ and $q \geq 3$. For such polynomials we define a formula $\mathbf{W}_x^{p,q,r}$ by $(!_q \mathbf{N}_x^p \multimap \mathbf{F}_3^q) \otimes \mathbf{N}_x^r$. The underlying set of this formula is $(\mathbb{N} \to \{0,1,*\}) \times \mathbb{N}$. We say that a pair $\langle f, n\rangle$ in this set *represents the word* $w_0 \ldots w_k \in \{0,1\}^*$ if $n \geq k$ holds and $f$ satisfies

$$f(i) = \begin{cases} w_i & \text{if } i \leq k, \\ * & \text{otherwise.} \end{cases}$$

We say that $g\colon (\mathbb{N} \to \{0,1,*\}) \times \mathbb{N} \to (\mathbb{N} \to \{0,1,*\}) \times \mathbb{N}$ *represents a function* $h\colon \{0,1\}^* \to \{0,1\}^*$ if, for all $w \in \{0,1\}^*$, $g$ maps any pair that represents $w$ to a pair that represents $h(w)$.

**Theorem 2** (Soundness)**.** *If* $\vdash \left(!_{y<t}\mathbf{W}_x^{q,r,s}\right) \multimap \mathbf{W}_{p(x)}^{u,v,w}$ *is derivable in* SBAL+(SKEW) *and the underlying function of this proof represents a function* $f\colon \{0,1\}^* \to \{0,1\}^*$ *then* $f$ *is computable in logarithmic space.*

**Theorem 3** (Completeness)**.** *For each* LOGSPACE *function* $f\colon \{0,1\}^* \to \{0,1\}^*$*, there exists in* SBAL+(SKEW) *a derivation of a sequent* $\vdash \left(!_{y<t}\mathbf{W}_x^{q,r,s}\right) \multimap \mathbf{W}_{p(x)}^{u,v,w}$ *whose underlying function represents* $f$*.*

We outline the proof of soundness in the next section. The proof of completeness goes by encoding a Turing Machine; we omit the details for space-reasons.

Theorem 2 can also be formulated with $\mathbf{S}_x$ instead of $\mathbf{W}_x$. To do this we need to strengthen the iteration principled embodied by $\mathbf{N}_x$ and $\mathbf{S}_x$ to the recursion principles

$$\mathbf{N}_x^q \vdash \forall \alpha \leq p.\, !_{y<x}B \multimap \alpha(0) \multimap \alpha(x), \qquad (\text{REC}\mathbf{N})$$

$$\mathbf{S}_x^q \vdash \forall \alpha \leq p.\, !_{y<x}B \multimap !_{y<x}B \multimap \alpha(0) \multimap \alpha(x), \qquad (\text{REC}\mathbf{S})$$

in which we write $B$ for $(\mathbf{N}_{y+w}^v \multimap \alpha(y) \multimap \alpha(y+1))$. The underlying functions of these sequents are such that the additional argument $\mathbf{N}_{y+w}^v$ supplies the value of $y$ to the step-function. We do not know if the recursion principles are derivable. However, since they can be interpreted in the semantics to be introduced in the next section, we can assume them without harming soundness.

**Lemma 4.** *For all* $p$ *there exist* $u$*,* $v$*,* $w$ *and* $q$ *and a proof in* SBAL+(REC$\boldsymbol{N}$) *of* $\vdash !_q\mathbf{W}_x^{u,v,w} \multimap \mathbf{S}_x^p$ *whose underlying function maps any pair* $\langle f,n\rangle$ *that represents a word* $b$ *to the impredicative representation of* $b$*.*

**Lemma 5.** *There exist polynomials* $u$*,* $v$*,* $w$ *and* $p$ *and a proof in* SBAL+(REC$\boldsymbol{S}$) *of* $\vdash \mathbf{S}_x^p \multimap \mathbf{W}_{x+1}^{u,v,w}$ *whose underlying function maps any word* $b$ *to a pair* $\langle f,n\rangle$ *representing* $b$*.*

By these lemmas, Theorems 2 implies a statement to effect that the underlying function $f\colon \{0,1\}^* \to \{0,1\}^*$ of each derivation $\vdash \mathbf{S}_x^r \multimap \mathbf{S}_{p(x)}^q$ for sufficiently large $q$ is computable in logarithmic space. We leave it for further work to ascertain whether a completeness result of this form holds.

## 3. An interaction model for SBAL

In this section we outline how SBAL proofs can be compiled to space-efficient functions. The definition of the compilation takes the form of an interpretation in a semantic model. Owing to the fact that the underlying functions of SBAL proofs are naturally represented in System F, the semantic model is built from types and terms in System F, which are equipped with realisers that allow the space-efficient evaluation of functions by interaction dialogues.

We assume a standard formulation of System F, e.g. [3], with explicit type abstraction and $\beta\eta$-equality, and write $\vdash_{\mathbb{F}}$ for the typing judgements in this system.

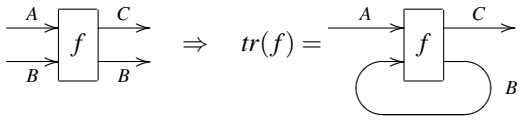
For each type context $\Sigma = (\alpha_1\colon \mathsf{Type}, \ldots, \alpha_n\colon \mathsf{Type})$ we define a syntactic category $\mathbb{F}(\Sigma)$ of types and terms in context $\Sigma$. The objects of $\mathbb{F}(\Sigma)$ are types $\Sigma \vdash_{\mathbb{F}} A\colon \mathsf{Type}$ and the morphisms from $\Sigma \vdash_{\mathbb{F}} A\colon \mathsf{Type}$ to $\Sigma \vdash_{\mathbb{F}} B\colon \mathsf{Type}$ are equivalence classes of terms $\Sigma \mid \cdot \vdash_{\mathbb{F}} M\colon A \Rightarrow B$ under $\beta\eta$-equality. Each category $\mathbb{F}(\Sigma)$ is cartesian closed, has finite products and a natural number object $\mathbb{N}$. Write $\mathbb{F}$ for $\mathbb{F}(\emptyset)$. We regard $\mathbb{F}$ as a 'set theory' and call its objects and morphisms also sets and functions.

### 3.1. Geometry of Interaction situation

The semantic model for the compilation of SBAL is based on a Geometry of Interaction (GoI) situation [1] that captures a way of modelling computation as question/answer dialogues. We introduce the relevant structure of this GoI situation in this section and refer to [1, 8] for further information. In particular, a discussion on the relation of this GoI situation to Girard's original Geometry of Interaction [5] can be found in Haghverdi's thesis [8].

**3.1.1. Sets and partial functions.** Let $\mathbb{B}$ be the category of sets and partial functions, in which each object $A$ is furthermore equipped with a total coding function $c\colon A \to 2^*$. For $a \in A$, we write $|a|$ for the length of the word $c(a)$. We say that a morphism $f\colon A \to B$ in $\mathbb{B}$ is computable in space $g(n)$ if there exists a $DSpace(g(n))$-Turing Machine that maps the code of each $a \in dom(f)$ to the code for $f(a) \in B$.

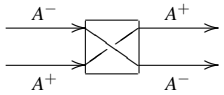
The category $\mathbb{B}$ has binary coproducts $+$ that are given by disjoint union with an evident coding function. Moreover, $\mathbb{B}$ is traced with respect to $+$, which means that, for each $f\colon A+B \to C+B$ there exists a morphism $tr(f)\colon A \to C$ satisfying the trace axioms [10]. Explicitly, $tr(f)$ is given by $tr_0 \circ inl$, where $tr_0\colon A+B \to C$ is the least function satisfying $tr_0(x) = c$ if $f(x) = inl(c)$ and $tr_0(inr(b))$ if $f(x) = inr(b)$. The trace operation can be depicted as follows.




Other objects of interest in $\mathbb{B}$ are: the natural numbers $\mathbb{N}$ with binary encoding, the set of pairs $A \times B$ with a reasonable encoding; and the set $V(X)$ with a coding function that has linear overhead, i.e. $|\eta| \in O(\sum_{x\in X} |\eta(x)|)$. We note that, for each resource polynomial $p \in P(X)$, the function $\eta \mapsto p[\eta]$ of type $V(X) \to \mathbb{N}$ is computable in linear space.

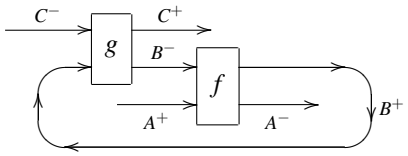
**3.1.2. GoI construction.** We now define a category $\mathbb{G}$ by using the GoI construction of [1] on $\mathbb{B}$. An object $A$ of $\mathbb{G}$ is a pair $(A^-, A^+)$ of two objects $A^-$ and $A^+$ of $\mathbb{B}$. The set $A^-$ is thought of as a set of questions and $A^+$ is thought of as a set of answers.

A morphism from $A$ to $B$ in $\mathbb{G}$ is a morphism of type $A^+ + B^- \to A^- + B^+$ in $\mathbb{B}$. Before we explain the intuition for this definition, let us define the identity map and composition. The identity morphism $id\colon A \to A$ is defined to be $[inr, inl]\colon A^+ + A^- \to A^- + A^+$.




The composition $g \cdot f\colon A \to C$ of $f\colon A \to B$ and $g\colon B \to C$ is the trace of $(A^- + g) \circ (f + C^-)$ with respect to $B^+$.

$g \cdot f =$




To understand the definition of morphisms in $\mathbb{G}$, consider the object $I = (\emptyset, \emptyset)$. Morphisms of type $I \to A$ in $\mathbb{G}$ are maps of type $A^- \to A^+$ in $\mathbb{B}$. Thus, a map $e\colon I \to A$ describes how to answer questions for $A$. A morphism $f\colon A \to B$ in $\mathbb{G}$ then describes how we can answer questions for $B$, given that we know how to answer questions for $A$. That is, if we have a map $e\colon I \to A$ describing how to answer questions for $A$ then we also have a map $f \cdot e\colon I \to B$ describing how to answer questions for $B$.

We confuse the morphisms of $\mathbb{G}$ with their defining morphisms in $\mathbb{B}$. For example, we say that a morphism $f\colon A \to B$ in $\mathbb{G}$ is in $DSpace(g(n))$ whenever this is the case for the corresponding $\mathbb{B}$-morphism $f\colon A^+ + B^- \to A^- + B^+$.

The category $\mathbb{G}$ has been studied thoroughly, see e.g. [8]. In this paper we use a monoidal closed structure $(I, \otimes, \multimap)$ and an endo-functor $!(-)$ on $\mathbb{G}$. On objects these are defined by $A \otimes B = (A^- + B^-, A^+ + B^+)$, $A \multimap B = (A^+ + B^-, A^- + B^+)$ and $!A = (A^- \times \mathbb{N}, A^+ \times \mathbb{N})$.

### 3.2. Realisability

We now introduce the realisability category $T(X)$, which is obtained by equipping $\mathbb{F}$ with realisers from $\mathbb{G}$. Its definition is inspired by the realisability model for BLL of Hofmann & Scott [9]. The category $T(X)$ has enough structure to interpret the multiplicative exponential fragment of SBAL and will be the basis of a model for full SBAL.
For any finite set $X$ of resource variables, define $T(X)$ by:

**Objects** An object $A$ of $T(X)$ is a triple $(|A|, \|A\|, \Vdash)$ of:

1. an object $|A|$ in $\mathbb{F}$ (the *underlying set*);
2. a mapping $\|A\|$ that assigns to each $\eta \in V(X)$ an object $\|A\|_\eta$ of $\mathbb{G}$ (the *realising object*);
3. a relation $\Vdash \subseteq (\Sigma_{\eta \in V(X)} \mathbb{G}(I, \|A\|_\eta)) \times \mathbb{F}(1, |A|)$ (the *realisation relation*).

This data must be such that there exist $c, d \in \mathbb{N}$ with

$$\forall \eta \in V(X).\ \forall x \in \|A\|^-_\eta \cup \|A\|^+_\eta.\ |x| \le c \cdot |\eta| + d.$$

**Morphisms** A morphism $f\colon A \to B$ in $T(X)$ is a morphism $f\colon |A| \to |B|$ in $\mathbb{F}$, for which there exists a realiser $r\colon \Pi_{\eta \in V(X)} \|A\|_\eta \to \|B\|_\eta$ satisfying:

1. For all $\eta \in V(X)$, $a \in |A|$ and $e\colon I \to \|A\|_\eta$, $\eta, e \Vdash_A a$ implies $\eta, r_\eta \cdot e \Vdash_B f(a)$.
2. The mapping $\langle \eta, q \rangle \mapsto r_\eta(q)$ is LINSPACE-computable.

The size restriction in the definition of objects is needed for defining composition. The realiser for a composition is defined by the evident composition of realisers, and the size restriction ensures LINSPACE computability of the result.

The category $T(X)$ has a symmetric monoidal structure

$$|A \otimes B| = |A| \times |B|, \qquad \|A \otimes B\|_\eta = \|A\|_\eta \otimes \|B\|_\eta,$$
$$\eta, e_a \otimes e_b \Vdash_{A \otimes B} \langle a, b \rangle \iff (\eta, e_a \Vdash_A a) \wedge (\eta, e_b \Vdash_B b),$$

whose unit is the terminal object of $T(X)$ defined by $|1| = \{*\}$ and $\|1\|_\eta = I$. It has a monoidal exponent defined by

$$|A \multimap B| = |A| \Rightarrow |B|, \qquad \|A \multimap B\|_\eta = \|A\|_\eta \multimap \|B\|_\eta,$$
$$\eta, r \Vdash_{A \multimap B} f \iff \text{if } (\eta, e \Vdash_A a) \text{ then } (\eta, ev \cdot (r \otimes e) \Vdash_B f(a)).$$

In this definition, $ev$ is the application morphism for the monoidal closed structure in $\mathbb{G}$.

To model the bounded modality, we have for each resource polynomial $p \in P(X)$ and each $x \notin X$ a functor $!_{x<p}\colon T(X \cup \{x\}) \to T(X)$, such that

$$|!_{x<p}A| = |A|,$$
$$\|!_{x<p}A\|_\eta = \Big( \sum_{n<p[\eta]} \|A\|^-_{\eta[n/x]}, \sum_{n<p[\eta]} \|A\|^+_{\eta[n/x]} \Big),$$

and $\eta, e \Vdash_{!_{x<p}A} a$ holds if and only if

$$\forall n < p[\eta].\ \exists e'\colon I \to \|A\|_{\eta[n/x]}.\ \Big(\forall q \in \|A\|^-_{\eta[n/x]}.\, e(n,q) = \langle n, e'(q) \rangle\Big) \wedge (\eta[n/x], e' \Vdash_A a).$$

The object $!_{x<p}A$ can be viewed as the object $A$ together with an additional memory cell of size $\log(p)$. A question to $!_{x<p}A$ consists of a question $q$ for $A$ and a number $n < p[\eta]$. We view $n$ as the data to be stored in the memory cell. An answer from $!_{x<p}A$ by definition comprises an answer to the question for $A$ together with the value $n$ that was put in the memory cell when the question was sent to $!_{x<p}A$.

For all objects $A$ and $B$ of $T(X \cup \{x\})$, there are maps

$$der\colon !_{x<1}A \longrightarrow A[0/x],$$
$$distr\colon !_{x<p}A \otimes !_{x<p}B \longrightarrow !_{x<p}(A \otimes B),$$
$$contr\colon !_{x<p+q}A \longrightarrow !_{x<p}A \otimes !_{y<q}A[p+y/x],$$
$$dig\colon !_{x<\sum_{y<p} q(y)}A \longrightarrow !_{y<p}!_{z<q(y)}A[z + \sum_{u<y} q(u)/x]$$

with the evident underlying functions. To give an example how these functions are realised, we spell out a realiser for *contr*. A realiser $r_\eta$ has domain

$$\sum_{n<p[\eta]+q[\eta]} \|A\|^+_{\eta[n/x]} + \left( \sum_{n<p[\eta]} \|A\|^-_{\eta[n/x]} + \sum_{n<q[\eta]} \|A\|^-_{\eta[n+p[\eta]/x]} \right).$$

Its range is the same set with inverted polarities. Concretely, $r_\eta$ can be defined by:

$$r_\eta inl(n,a) = \begin{cases} inr(inl(n,a)) & n < p[\eta] \\ inr(inr(n-p[\eta],a)) & \text{otherwise} \end{cases}$$
$$r_\eta inr(inl(n,q)) = inl(n,q)$$
$$r_\eta inr(inr(n,q)) = inl(n+p[\eta],q)$$

Since the polynomial $p$ is fixed and the function mapping $\eta \mapsto p[\eta]$ is computable in linear space, it follows that $\langle \eta, x\rangle \mapsto r_\eta(x)$ can be computed in linear space.

While $T(X)$ can interpret the multiplicative exponential fragment of SBAL, it does not model universal quantification. Hence, impredicative encodings of datatypes are not available in it. Nevertheless, we can prove complexity results for $T(X)$ by defining directly a type $S_x$ of small natural numbers. The object $S_x$ in $T(\{x\})$ is defined by

$$|S_x| = \mathbb{N}, \qquad \|S_x\|^-_\eta = \|S_x\|^+_\eta = \{n \in \mathbb{N} \mid n \le x[\eta]\},$$
$$(\eta, e \Vdash_{S_x} n) \iff (n \le \eta(x)) \wedge (e(0) = n).$$

**Lemma 6.** *The underlying function of each morphism $!_q S_x \to S_{p(x)}$ in $U(\{x\})$ is linear space computable.*

The proof of this lemma follows from the LINSPACE assumption in the definition of $T(X)$.

**Lemma 7.** *If the underlying function of a morphism $!_q(S_x \multimap S_3) \otimes !_q S_x \longrightarrow (!_q S_{p(x)} \multimap S_3) \otimes S_{p(x)}$ in $U(\{x\})$ represents a function $f\colon \{0,1\}^* \to \{0,1\}^*$ in the sense of Definition 4, then $f$ is computable in logarithmic space.*

This lemma is shown by observing that from a realiser for the morphism one obtains a function that computes the result of $f$ bit by bit. The space needed for this function is linear in the space necessary to store a counter over the bits of the output of $f$, i.e. $O(\log(p(x))) = O(\log(x))$.

**Lemma 8** (Skewed iteration on $S_x$)**.** *For each $p \in P_1(X)$ and each $i \in P(X)$, there exists $q \in P(X)$, such that the object $S_x \multimap !_q !_{y<x}(!_i S_{p(y)} \multimap S_{p(y+1)}) \multimap !_q S_{p(0)} \multimap S_{p(x)}$ has a global element, whose underlying function is the standard iteration combinator on natural numbers.*

Having defined a semantics for the multiplicative exponential fragment of SBAL + (SKEW), to define a model for full SBAL + (SKEW), it remains to account for formulae with free second-order variables and with bounded universal quantification.

Our approach to modelling universal quantification is similar to the approaches in many other realisability models in the literature, e.g. [2]. Informally, we let the underlying set $|\forall\alpha \le p.A|$ be $\forall\alpha.|A|$ and define the realisation relation on $\forall\alpha \le p.A$ uniformly, that is a realiser for $f\colon \forall\alpha.|A|$ is a realiser of *all* the instances $f(|B|)$ for appropriate objects $B$. Although the polynomial $p$ is not mentioned in this informal explanation, it is essential for finding a realising object for $\forall\alpha \le p.A$. Let us explain the role of $p$ using the example where $A$ is $\alpha \multimap \alpha$. Corresponding to rule (∀-L), there should be a morphism $\|\forall\alpha \le p.\,\alpha \multimap \alpha\|_\eta \to \|B \multimap B\|_\eta$ for all appropriate objects $B$. If we use uniform realisers for the universal quantifier, this map should be essentially the identity (perhaps up to some encoding). Hence, the set of questions $\|\forall\alpha \le p.\,\alpha \multimap \alpha\|^-_\eta$ should be such that it can embed the sets of questions $\|B \multimap B\|^-_\eta$ for all appropriate $B$, and likewise for the answers. Since the definition of objects in $T(X)$ contains a size-restriction on $\|\forall\alpha \le p.\,\alpha \multimap \alpha\|$, this implies that the object $B$ cannot range over all objects of $T(X)$. Instead, we use the bound $p$ to restrict the quantification to range only over objects $B$ whose realising object $\|B\|_\eta$ can be effectively encoded in the object $\|\mathbb{N}_p\|_\eta$ defined by $\|\mathbb{N}_p\|^-_\eta = \|\mathbb{N}_p\|^+_\eta = \{n \in \mathbb{N} \mid n < p[\eta]\}$. Then we can define $\|\forall\alpha \le p.\,\alpha \multimap \alpha\|_\eta$ to be $\|\mathbb{N}_p\|_\eta \multimap \|\mathbb{N}_p\|_\eta$. Thus, we model to quantification $\forall\alpha \le p.A$ such that it ranges only over objects that can be encoded in $\|\mathbb{N}_p\|_\eta$.

To model universal quantification as sketched above, we find it convenient to restrict our attention to the full subcategory of $T(X)$ consisting of objects whose realising object can be effectively encoded in $\|\mathbb{N}_p\|$ for some $p$. Formally, we consider the full subcategory of $T(X)$ with objects $A$ satisfying $\|A\| = \|\mathbb{N}_{p_A}\|$ for some $p_A \in P(X)$. We write $(|A|, p_A, \Vdash_A)$ for such an object and call $p_A$ its *size polynomial*. We write $U(X)$ for the resulting full subcategory of $T(X)$. It can be shown that the constructions of $\otimes$, $\multimap$ and $!_{x<p}$ in $T(X)$ restrict to $U(X)$, by encoding the realising objects appropriately. The size bounds in Figure 1 derive from the space requirements for these encodings.

In the rest of this section we outline how to model formulae with free variables as well as universal quantification. We model a formula $\Sigma \vdash A$ by a family $(A_\rho)_{\rho \in E(\Sigma,X)}$ of $U(X)$-objects $A_\rho$ indexed by environments $\rho$. An environment is a function that maps second-order variables to objects in $U(X)$. We model a proof $\Sigma \mid A \vdash B$ as a family $(f_\rho\colon A_\rho \to B_\rho)_{\rho \in E(\Sigma,X)}$ of morphisms in $U(X)$. Both the objects and the morphisms are subject to a uniformity condition. In particular, the morphisms must be uniformly realised, which means that all the maps $f_\rho$ share a single realiser. This uniformity condition is what makes the interpretation of the universal quantifier work.

**Definition 5.** Let $X$ be a finite set of resource variables, let $y_1, \ldots, y_n$ be variables not occurring in $X$ and let $p \in P_n(X)$. Define $\mathbf{Obj}^p_{\vec{y}}(X)$ to be the set of all objects in $U(X \cup \{\vec{y}\})$

that are positive in $\vec{y}$ and have size polynomial $p(\vec{y})$.

**Definition 6.** Let $\Sigma$ be a second-order context with free resource variables in $X$. A $\Sigma$-*environment on* $X$ is a function that assigns to each variable $\alpha$, for which there is a declaration $\alpha \leq p\colon \mathsf{Fm}_n$ in $\Sigma$, a pair $\langle \vec{y}, A\rangle$ such that the variables in $\{\vec{y}\}$ do not appear in $X$ and $A$ is an object in $\mathbf{Obj}^p_{\vec{y}}(X)$.

We write $E(\Sigma, X)$ for the set of $\Sigma$-environments on $X$.

For a context $\Sigma$, we write $\Sigma_{\mathbb{F}}$ for the System F context $\{\alpha\colon \mathsf{Type} \mid (\alpha \leq p\colon \mathsf{Fm}_n) \in \Sigma\}$. For any $\Sigma$-environment $\rho$ we write $\rho_{\mathbb{F}}$ for the System F-substitution $\alpha \mapsto |\pi_2(\rho(\alpha))|$.

Define a category $\mathrm{UFam}(\Sigma, X)$ of uniform families over $\Sigma$ and $X$ as follows.

**Objects** An object is a triple $(|A|, p, \Vdash)$, where $|A|$ is an object of $\mathbb{F}(\Sigma_{\mathbb{F}})$ and $p$ is a polynomial in $P(X)$. Finally, $\Vdash$ is a family of relations $(\Vdash_\rho)_{\rho \in E(\Sigma,X)}$, such that for all $\rho \in E(\Sigma, X)$ the definition $A_\rho := (|A|[\rho_{\mathbb{F}}], p, \Vdash_\rho)$ yields an object of $U(X)$.

**Morphisms** A morphism $f\colon A \to B$ is a map $f\colon |A| \to |B|$ in $\mathbb{F}(\Sigma_{\mathbb{F}})$ that has a uniform realiser. This means that there exists $r\colon \Pi\eta \in V(X).\, \|A_\rho\|_\eta \to \|B_\rho\|_\eta$ such that $r$ realises $f[\rho_{\mathbb{F}}]$ for all $\rho \in E(\Sigma, X)$.

The category $\mathrm{UFam}(\cdot, X)$ is equivalent to $U(X)$.

For each object $A$ of $\mathrm{UFam}((\Sigma, \alpha \leq p\colon \mathsf{Fm}_n), X)$, there exists an object $\forall \alpha \leq p.A$ in $\mathrm{UFam}(\Sigma, X)$ satisfying

$$|\forall \alpha \leq p.A| = \forall \alpha.\, |A|, \qquad \|\forall \alpha \leq p.A\|_\eta = \|A\|_\eta,$$
$$\eta, e \Vdash_{(\forall \alpha \leq p.A)_\rho} f \iff \forall B \in \mathbf{Obj}^p_{\vec{y}}(X).\, \eta, e \Vdash_{A_{\rho[\alpha \mapsto \langle \vec{y}, B\rangle]}} f(|B|),$$

where $\vec{y} = y_1, \ldots, y_n$ is a vector of variables that do not occur in $X$. This definition can be extended to a functor $\mathrm{UFam}((\Sigma, \alpha \leq p\colon \mathsf{Fm}_n), X) \to \mathrm{UFam}(\Sigma, X)$, using which UFam can interpret the whole of SBAL.

The complexity results from Section 2.3 are a consequence of Lemmas 6 and 7. This is the case because impredicative data types can be encoded in the object $S_x$ of small numbers, in the following sense.

**Lemma 9.** *For each $p \in P_1(X)$, there are in $U(X \cup \{x\})$ morphisms $c\colon \mathbf{N}^{\lambda x.x}_x \to S_x$ and $d\colon !_{3x \cdot p(x)} S_x \to \mathbf{N}^p_x$, whose underlying functions are an isomorphism.*

A similar result holds for $\mathbf{F}^p_k$. We note that the underlying sets of $\mathbf{N}^p_x$ and $S_x$ are isomorphic because we use $\mathbb{F}$ to represent the underlying functions. Had we use standard set theory then the underlying set of $\mathbf{N}^p_x$ could have contained additional elements.

The next lemma follows from Lemma 8 using Lemma 9.

**Lemma 10** (Correctness of skewed iteration)**.** *Let A, B, C and D be small data types satisfying the assumptions of rule (*SKEW*). Then there are polynomials p and q, for which the object $\mathbf{N}^p_x \multimap !_q !_{y<x}(A \multimap B) \multimap !_q C \multimap D$ in $\mathrm{UFam}(\Sigma, X)$ has a global element, whose underlying function is the standard iteration combinator.*

### 3.3. Conclusion and further work

We believe that SBAL improves on existing type systems for logarithmic space, such as [14, 13, 11, 15]. With higher-order functions, SBAL allows for a natural representation of LOGSPACE algorithms, and by supporting impredicative-style datatype encodings, SBAL offers flexible data representation. The model construction for SBAL is based on ideas from [15], but improves on [15] in simplicity and generality.

Interesting questions for further work include studying the possibilities for simplifications of SBAL, perhaps using ideas from LLL [6], finding a characterisation of skewed iteration in terms of logical concepts, and studying to what extent LOGSPACE-algorithms on structured data, such as graphs, can be represented naturally in SBAL.

**Acknowledgments.** I wish to thank Martin Hofmann for interesting discussions and feedback. This work has been supported by the DFG as part of the project Pro.Platz.

## References

[1] S. Abramsky, E. Haghverdi, and P. Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

[2] U. Dal Lago and M. Hofmann. Quantitative models and implicit complexity. In *FSTTCS05*, pages 189–200, 2005.

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

[4] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[5] J.-Y. Girard. Geometry of interaction I: interpretation of system F. In *Logic Colloquium 88*, pages 221–260, 1989.

[6] J.-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

[7] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97:1–66, 1992.

[8] E. Haghverdi. *A Categorical Approach to Linear Logic, Geometry of Proofs and Full Completeness*. PhD thesis, University of Ottawa, 2000.

[9] M. Hofmann and P. Scott. Realizability models for BLL-like languages. *Theoretical Computer Science*, 318(1–2):121–137, 2004.

[10] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[11] L. Kristiansen. Neat function algebraic characterizations of LOGSPACE and LINSPACE. *Computational Complexity*, 14:72–88, 2005.

[12] Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318:163–180, 2004.

[13] P. Møller-Neegaard. *Complexity Aspects of Programming Language Design*. PhD thesis, Brandeis University, 2004.

[14] P. Møller-Neergaard. A functional language for logarithmic space. In *APLAS*, pages 311–326, 2004.

[15] U. Schöpp. Space-efficient computation by interaction – a type system for logarithmic space. In *Computer Science Logic 2006*, volume 4207 of *LNCS*. Springer-Verlag, 2006.