Stratified Bounded Affine Logic for Logarithmic Space (Draft)

Ulrich Schöpp

April 20, 2007

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 [Girard, 1987] 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 [Girard et al., 1992], Light Linear Logic [Girard, 1998] and Soft Linear Logic [Lafont, 2004]. 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) [Girard et al., 1992]. Being a sub-system of linear logic, SBAL can naturally be viewed as a type system for the second order λ-calculus. We show that SBAL-typeable λ-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 ith character in the string. One can then represent a string as a function $s\colon \mathbb{N} \to \Sigma$, where Σ is

the alphabet, and where $s(i)$ is either the ith 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 Σ can be represented in the second-order λ-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 can be 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 connectives $\otimes$ 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 $!_{x<p+q}A \multimap !_{x<p}A \otimes !_{x<q}A$. 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 $\forall\alpha \le p.\,A$.

While we view SBAL as a type system that identifies LOGSPACE-computable functions in the second-order λ-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, such as the ones in the PTIME-logics of loc. cit., the evaluation of such functions typically requires linear space or more. 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. We formulate the compilation as an interpretation of Stratified Bounded Affine Logic in a game semantics model. The central idea of game semantics is to model computation as a question and answer dialogue between a number of entities. 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 Stratified Bounded Affine Logic in such a game semantics game semantics 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 input, then the whole program can be evaluated in logarithmic space. The construction in Section 3.2 is based on this observation.

2 Stratified Bounded Affine Logic

In this section we define the logic SBAL. As in Bounded Linear Logic, resource polynomials appear explicitly in the formulae of SBAL.

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.

$$\binom{x}{n} = \frac{x \cdot (x-1) \cdots (x-n+1)}{n!}$$

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 will 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 by the closure properties of resource polynomials, each polynomial with natural coefficients and variables in X is a resource polynomial on X.

A *substitution* $\sigma \colon X \to Y$ is a function from Y to $P(X)$. For each $\sigma \colon Y \to X$ and each $p \in P(X)$ define $p[\sigma] \in P(Y)$ to be the polynomial obtained by substituting $\sigma(x)$ for x in p for each $x \in X$. For two substitutions $\sigma \colon X \to Y$ and $\tau \colon Y \to Z$ their composition $\tau \circ \sigma \colon X \to Y$ is defined by $(\tau \circ \sigma)(z) = z[\tau][\sigma]$. The identity substitution $id \colon X \to X$ is given by $id(x) = x$.

Lemma 1. *The following are true.*

1. $id \circ \sigma = \sigma \circ id = \sigma$ *and* $(\sigma \circ \tau) \circ \theta = \sigma \circ (\tau \circ \theta)$.
2. $p[\sigma \circ \tau] = p[\sigma][\tau]$.

An *environment* η *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. Any environment $\eta \in V(X)$ can be viewed as a substitution $\eta \colon \emptyset \to X$.

Lemma 2. *The following are true.*

1. *For all* $p \in P(X)$ *and all* $\eta \in V(X)$, *we have* $0 \leq p[\eta]$.
2. *If* $p, q \in P(Y)$ *and* $\sigma \colon X \to Y$, *then* $p \leq q$ *implies* $p[\sigma] \leq q[\sigma]$.
3. *If* $p \in P(X)$, $q \in P(X + \{x\})$ *and* $\eta \in V(X)$, *then* $q[p/x][\eta] = q[\eta][p[\eta]/x]$.

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

Definition 2. Define the set $P_k(X)$ of polynomials with k bound variables by

$$\begin{aligned} P_0(X) &= P(X), \\ P_{k+1}(X) &= \{\lambda x.\, p \mid p \in P_k(X \cup \{x\}), x \notin X\}. \end{aligned}$$

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 given by the grammar

$$A ::= \alpha(p_1, \ldots, p_n) \mid A \otimes A \mid A \multimap A \mid !_{x<p} A \mid \forall \alpha \leq p \colon \mathsf{Fm}_n.\, A,$$

where the resource variable x in $!_{x<p}A$ binds any occurrence of x in A and the second-order variable α in $\forall \alpha \leq p \colon \mathsf{Fm}_k.\, A$ binds any occurrence of α in A. With the exception of the universal quantifier, the formulae are as in Bounded Linear Logic. In particular, each second-order variable α is applied to a number of resource polynomials $p_1, \ldots, p_n$. The logic is set up so that each second-order α variable has a fixed arity, meaning that α must be applied to previously specified, fixed number of resource polynomials. Compared to Bounded Linear Logic, the formula $\forall \alpha \leq p \colon \mathsf{Fm}_n.\, A$ for universal quantification has two additional parameters n and p. The parameter n is there for technical convenience only. It represents the arity of the variable α. More important is the presence of the polynomial p with n bound variables. It serves as a size bound on α.

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, Σ is a *second-order context* of declarations $\alpha \leq q \colon \mathsf{Fm}_k$, declaring α 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 Σ. The rules defining 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 the formulae in Section 3.2. For

$$\frac{\forall 1 \leq i \leq n.\, q_i \in P(X)}{\Sigma,\, \alpha \leq p\colon \mathsf{Fm}_n \vdash \alpha(q_1, \ldots, q_n) \leq p(q_1, \ldots, q_n)\colon \mathsf{Fm}}$$

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

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

$$\frac{\Sigma \vdash A \leq p\colon \mathsf{Fm} \qquad p \leq q}{\Sigma \vdash A \leq q\colon \mathsf{Fm}}$$

Figure 1: Formulae and their size polynomials

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 $A \leq p\colon \mathsf{Fm}_k$ the number k can be inferred from the polynomial p, we will often write just $A \leq p$ for it. In particular, we will 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 introduce a notion of *positive* and *negative* occurrences of a resource variable in a formula. These notions are defined inductively 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 then it occurs positively (resp. negatively) in $A \otimes B$. If x occurs positively (resp. negatively) in B then it occurs positively (resp. negatively) in $A \otimes B$. If x occurs positively (resp. negatively) in A then it occurs negatively (resp. positively) in $A \multimap B$. If x occurs positively (resp. negatively) in B then it occurs positively (resp. negatively) 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. 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 difference to the modality rules, as Hofmann & Scott [Hofmann and Scott, 2004] have shown that this rule can be added without harm to BLL as well. The main essential difference between SBAL and BLL lies in the rules for universal quantification. In particular, rule (∀-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 naturally be viewed as typing rules for the second-order λ-calculus, also known as System F. Each SBAL formula can be translated to a formula of System F by removing the modality $!_{x<p}$ and by replacing $\multimap$ with $\Rightarrow$ and $\forall\alpha \leq p$ with $\forall\alpha$. With this translation, 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 λ-term to each derivable sequent. We call this λ-term the *underlying function* of a sequent.

2.1 Examples

2.1.1 Data types

We give a few examples to show that, while SBAL is not fully impredicative, encodings of inductive data types are nevertheless available. For example, boolean values can be represented using the formula

$$\mathbf{B}^p \stackrel{\text{def}}{=} \forall\alpha \leq p\colon \mathsf{Fm}_0.\, \alpha \multimap \alpha \multimap \alpha.$$

This corresponds to the encoding of booleans by $\forall\alpha.\, \alpha \multimap \alpha \multimap \alpha$ in Bounded Linear Logic. Natural numbers smaller than x can be encoded by the formula

$$\mathbf{N}^p_x \stackrel{\text{def}}{=} \forall\alpha \leq p\colon \mathsf{Fm}_1.\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x).$$

This representation is very similar to that in Bounded Linear Logic, where natural numbers no larger than x would be represented using the formula

$$\forall\alpha.\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x).$$

For a final example of a data type, binary strings of length at most x can be represented in SBAL by the formula

$$\mathbf{S}^p_x \stackrel{\text{def}}{=} \forall\alpha \leq p\colon \mathsf{Fm}_1.\, !_{y<x}(\alpha(y) \multimap \alpha(y{+}1)) \multimap !_{y<x}(\alpha(y) \multimap \alpha(y{+}1)) \multimap \alpha(0) \multimap \alpha(x).$$

To give an example how these data types can be used, we define the addition function for the above encoding of natural numbers. First we define the constant zero and the successor functions.

Zero

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\alpha \leq p \mid \alpha(0) \vdash \alpha(0)}\,(\textsc{Axiom})}{\alpha \leq p \mid !_{y<0}(\alpha(y) \multimap \alpha(y+1)),\, \alpha(0) \vdash \alpha(0)}\,(\textsc{Weakening})}{\alpha \leq p \mid \cdot \vdash !_{y<0}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(0)}\,(\multimap\text{-R})}{\vdash \mathbf{N}^p_0}\,(\forall\text{-R})$$

Monotonicity Rules

$$\frac{\Sigma \vdash \alpha(p_1, \ldots, p_n) \colon \mathsf{Fm} \qquad \Sigma \vdash \alpha(q_1, \ldots, q_n) \colon \mathsf{Fm} \qquad \forall 1 \leq i \leq n.\, p_i \leq q_i}{\Sigma \vdash \alpha(p_1, \ldots, p_n) \leq \alpha(q_1, \ldots, q_n)}$$

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

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

Logical Rules

$$(\textsc{Axiom})\ \frac{\Sigma \vdash A \leq A'}{\Sigma \mid A \vdash A'}$$

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

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

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

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

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

$$(\textsc{Weakening})\ \frac{\Sigma \vdash A \colon \mathsf{Fm} \qquad \Sigma \mid \Gamma \vdash B}{\Sigma \mid \Gamma, A \vdash B}$$

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

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

$$(\textsc{Functoriality})\ \frac{\Sigma \mid \Gamma \vdash A}{\Sigma \mid !_{x<p} \Gamma \vdash !_{x<p} A}$$

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

Figure 2: Inference rules of SBAL

Successor Let us write $S(x)$ for the formula $(\alpha(x) \multimap \alpha(x+1))$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{evident applications}}{\alpha \le p \mid !_{z<y}S(z) \multimap \alpha(0) \multimap \alpha(y),\ \alpha(y) \multimap \alpha(y+1),\ !_{z<y}S(z),\ \alpha(0) \vdash \alpha(y+1)}}{\alpha \le p \mid !_{z<y}S(z) \multimap \alpha(0) \multimap \alpha(y),\ S(y),\ !_{z<y}S(z),\ \alpha(0) \vdash \alpha(y+1)}\,(\text{D})}{\alpha \le p \mid !_{z<y}S(z) \multimap \alpha(0) \multimap \alpha(y),\ !_{z<1}S(y+z),\ !_{z<y}S(z),\ \alpha(0) \vdash \alpha(y+1)}\,(\text{C})}{\alpha \le p \mid !_{z<y}S(z) \multimap \alpha(0) \multimap \alpha(y),\ !_{z<y+1}S(z),\ \alpha(0) \vdash \alpha(y+1)}\,(\multimap\text{-R})}{\alpha \le p \mid !_{z<y}S(z) \multimap \alpha(0) \multimap \alpha(y) \vdash !_{z<y+1}S(z) \multimap \alpha(0) \multimap \alpha(y+1)}\,(\forall\text{-L})}{\alpha \le p \mid \mathbf{N}^p_y \vdash !_{z<y+1}S(z) \multimap \alpha(0) \multimap \alpha(y+1)}\,(\forall\text{-R})}{\cdot \mid \mathbf{N}^p_y \vdash \mathbf{N}^p_{y+1}}\,(\multimap\text{-R})}{\vdash \mathbf{N}^p_y \multimap \mathbf{N}^p_{y+1}}$$

Addition For the definition of addition we need to use the size polynomial of the type $\mathbf{N}^p_x$. Using the rules from Figure 1 it is straightforward to show that

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

is derivable. We write $n(p, x)$ as an abbreviation for the polynomial

$$2(2((x + 2(p(x) + p(x+1)))^2 + p(0)) + p(x)).$$

With this definition, addition is defined by the derivation below.

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

Notice that the second argument of the addition function contains the polynomial $\lambda z.\, n(p, x+z)$ rather than p.

This completes the definition of the addition function. Other functions, such as multiplication can be defined analogously. We remark, however, that the iteration principle embodied in the definition of $\mathbf{N}^p_x$ is not quite sufficient for the to define all the usual basic functions on natural numbers in the normal way. This can be seen on the example of the subtraction function. The predecessor function can be defined with type $\mathbf{N}^q_x \multimap \mathbf{N}^p_x$, where q is some polynomial with $q > p$. To define subtraction, we would like to iterate the predecessor function. But because the type of the predecessor function contains different polynomials q and p, we cannot use it as a step function for an iteration. We discuss this problem and how we deal with it in the next section on skewed iteration.

Coercion In Bounded Linear Logic, there exists a coercion from $\mathbf{N}_x$ to $!_p\mathbf{N}_x$. Such a coercion also exists in SBAL, where we are allowed to go from $\mathbf{N}^{\lambda x.(p+n(q,x)+1)^2}_x$ to $!_p\mathbf{N}^q_x$. The derivation of the coercion sequent is as follows.

$$
(\forall\text{-L})\ \frac{\dfrac{\vdash \mathbf{N}_x^q \le n(q,x)}{\vdash\ !_p\mathbf{N}_x^q \le (p+n(q,x)+1)^2} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{successor}}{\mathbf{N}_y^p \vdash \mathbf{N}_{y+1}^p}}{!_q\mathbf{N}_y^p \vdash\ !_p\mathbf{N}_{y+1}^q}}{\vdash\ !_{y<x}(!_p\mathbf{N}_y^q \multimap\ !_p\mathbf{N}_{y+1}^q)} \qquad \dfrac{\dfrac{\text{zero}}{\vdash \mathbf{N}_0^q}}{\vdash\ !_p\mathbf{N}_0^q}}{!_{y<x}(!_p\mathbf{N}_y^q \multimap\ !_p\mathbf{N}_{y+1}^q) \multimap\ !_q\mathbf{N}_0^p \multimap\ !_q\mathbf{N}_x^p \vdash\ !_q\mathbf{N}_x^p}}{\dfrac{\mathbf{N}_x^{\lambda x.(p+n(q,x)+1)^2} \vdash\ !_p\mathbf{N}_x^q}{\vdash\ \mathbf{N}_x^{\lambda x.(p+n(q,x)+1)^2} \multimap\ !_p\mathbf{N}_x^q}}
$$

At this point, one should note the difference between the coercion in SBAL and the corresponding coercion in BLL. For instance, in BLL we can use the coercion to transform each function of type $!_p\mathbf{N}_x \multimap \mathbf{N}_x$ to a function of type $\mathbf{N}_x \multimap \mathbf{N}_x$. The resulting function can be used as a step function in an interaction. In SBAL, we can only transform a function of type $!_p\mathbf{N}_x^q \multimap \mathbf{N}_x^r$ to one of type $\mathbf{N}_x^{\lambda x.(p+n(q,x)+1)^2} \multimap \mathbf{N}_x^r$. Due to the different size polynomials, this function cannot be used as the step function in an iteration. We discuss this issue further in the next section on skewed iteration.

Decreasing the polynomial If $q \ge p$ holds then there exists a proof $\mathbf{N}_x^q \multimap \mathbf{N}_x^p$, whose underlying function is the identity. This proof is shown in the derivation below, in which we write B for $!_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x)$.

$$
\begin{array}{c}
(\forall\text{-L})\ \dfrac{\dfrac{\overline{\alpha \le p \vdash \alpha \le p}}{\alpha \le p \vdash \alpha \le q}\ p \le q \qquad (\text{Ax})\ \dfrac{}{\alpha \le p \mid B \vdash B}}{(\forall\text{-R})\ \dfrac{\alpha \le p \mid \mathbf{N}_x^q \vdash B}{(\multimap\text{-R})\ \dfrac{\mathbf{N}_x^q \vdash \mathbf{N}_x^p}{\vdash \mathbf{N}_x^q \multimap \mathbf{N}_x^p}}}
\end{array}
$$

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. To represent binary strings, we use formula $\mathbf{N}_x^p \multimap \mathbf{T}^q$, where $\mathbf{T}^q$ is defined by

$$\mathbf{T}^q \stackrel{\text{def}}{=} \forall\alpha \le q\colon \mathsf{Fm}_0.\ \alpha \multimap \alpha \multimap \alpha \multimap \alpha.$$

Since the formula $\mathbf{T}^p$ denotes a type whose underlying set has three elements, the underlying set of $\mathbf{N}_x^p \multimap \mathbf{T}^q$ is the set of functions $\mathbb{N} \to \{0,1,*\}$. A binary string $b = b_1 \dots b_n \in \{0,1\}^*$ can be represented by the following function f_b of type $\mathbb{N} \to \{0,1,*\}$.

$$f_b(i) = \begin{cases} b_i & \text{if } 1 \le i \le n, \\ * & \text{if } i > n. \end{cases}$$

With this encoding, a LOGSPACE-algorithm $l\colon \{0,1\}^* \to \{0,1\}^*$ can be represented as a proof of a formula $(\mathbf{N}_{p(x)} \multimap \mathbf{T}) \multimap (\mathbf{N}_{q(x)} \multimap \mathbf{T})$ with appropriate superscripts, such that, for all $b \in \{0,1\}^*$, the underlying function of this proof maps the function f_b to the function $f_{l(b)}$.

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

$$\dfrac{\dfrac{\dfrac{}{\mathbf{N}^q_x \vdash \mathbf{N}^q_{x+1}}\,\text{successor} \qquad \dfrac{}{\mathbf{T}^q \vdash \mathbf{T}^q}}{\mathbf{N}^q_{x+1} \multimap \mathbf{T}^q,\ \mathbf{N}^s_x \vdash \mathbf{T}^q}}{\vdash (\mathbf{N}^q_{x+1} \multimap \mathbf{T}^q) \multimap (\mathbf{N}^q_x \multimap \mathbf{T}^q)}$$

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, if l to be computable in 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. Notice in particular that the elements of $\mathbf{N}_x$, i.e. the natural numbers up to x, can be stored in memory by using binary encoding. The general representation of LOGSPACE algorithms in SBAL is made precise in Section 2.3 below.

2.2 Skewed Iteration

In the examples we have shown that the restricted universal quantifier SBAL is powerful enough for the representation of data types. Due to the restriction on the universal quantifier, the iteration schemes for these data type representations are less general than for Bounded Linear Logic. For example, in BLL one can use a function f of type $!_p\mathbf{N}_x \multimap \mathbf{N}_x$ as the step function in an iteration, since there exists a coercion $\mathbf{N}_x \multimap !_p\mathbf{N}_x$, by which f can be coerced to $\mathbf{N}_x \multimap \mathbf{N}_x$. This does not work in SBAL, since there the coercion has type $\mathbf{N}^r_x \multimap !_p\mathbf{N}^q_x$, where r is a polynomial with $r > q$. Hence, we can only coerce a function f of type $!_p\mathbf{N}^q_x \multimap \mathbf{N}^q_x$ to one of type $\mathbf{N}^r_x \multimap \mathbf{N}^q_x$, and, due to q and r being different, this function cannot be used as a step function for an iteration.

The fact that in SBAL functions of type $!_p\mathbf{N}^q_x \multimap \mathbf{N}^q_x$ cannot be iterated turns out to be a real restriction on the expressivity of the system. For instance, we do not know if subtraction on natural numbers can be expressed in SBAL. Clearly, however, the subtraction function on natural numbers can be allowed, while still remaining in logarithmic space. In this section we discuss the problem and solve it by adding a rule for *skewed iteration* to SBAL. We start by discussing why iteration on $!_p\mathbf{N}^q_x \multimap \mathbf{N}^q_x$ is not available in SBAL.

In the compilation of SBAL that we define in the next section, there are two irreconcilable ways of implementing iteration. One way is such that the n-fold iteration of a step-function f and a basis-function g is evaluated essentially in the same way the iterated application

$$\lambda x.\ \underbrace{f(f(\dots(f}_{n \text{ times}}\, x)\dots)). \tag{1}$$

The iteration schemes obtained from impredicative data type encodings are implemented in this way. The main advantage of this implementation strategy is that it allows us to iterate functions f with large return values, i.e. values that require more than logarithmic space, such as the values of type $\mathbf{S}^p_x$. Iteration on functions with large return values is possible because function application is evaluated piece-by-piece, so that even if f returns a large value, it is not necessary to store all of it in memory at once. A disadvantage of implementing iteration by repeated application is that we cannot expect to use it for the iteration of functions of type $!_pA \multimap A$ and still maintain a logarithmic space bound on the evaluation. The reason is that, for a function f of type $!_pA \multimap A$,

the function in (1) will have type $!_{(p^n)}A \multimap A$, and our compilation method is such that the space needed to evaluate a function of this type is typically $\log(p^n)$, i.e. linear in n, or above.

A second way of implementing the n-fold iteration of f on g is as in the following pseudo-code.

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

The main difference to the previous implementation is that the intermediate value v is stored in memory. Therefore, we cannot use this implementation for step functions that return large values. On the other hand, if the values of type A are small enough to be stored wholly in memory, then functions of type $!_pA \multimap A$ can be iterated. This will be shown in Section 3.6.

While these two ways of implementing interaction clearly complement each other, the universal quantifier in SBAL only accounts for the first implementation of iteration. Since being able to iterate functions of type $!_pA \multimap A$ is important for applications, such as for defining subtraction on $\mathbf{N}^p_x$, we now extend SBAL with a rule for *skewed iteration*. Skewed iteration represents the second way of implementing iteration.

For the definition of skewed iteration, we need a notion of small data types, that is types whose elements can all be stored in memory wholly without leading beyond logarithmic space.

Definition 3. Let the data types of finite sets with k elements and of the natural numbers be given by:

$$\mathbf{F}^p_k \stackrel{\text{def}}{=} \forall\alpha \leq p.\, \underbrace{\alpha \multimap \ldots \multimap \alpha}_{k \text{ times}} \multimap \alpha, \qquad \text{where } k \in \mathbb{N}$$

$$\mathbf{N}^p_x \stackrel{\text{def}}{=} \forall\alpha \leq p.\, !_{y<x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x)$$

The set of *small data types* is defined inductively by:

- if $p \geq k$, then $\mathbf{F}^p_k$ is a small data type;
- if $q \geq \lambda x.x$, then $\mathbf{N}^q_p$ is a small data type; and
- if both A and B are small data types, then so is $A \otimes B$.

With this definition, we can formulate a rule for skewed iteration, which appears in Figure 3. We write SBAL$^+$ for SBAL extended with the rule for skewed iteration.

2.3 Complexity

In this section we state our main results, whose proofs will occupy the rest of this paper.

First, we note 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 $\mathbf{N}_x$ are used to

A, B, C and D small data types,
p and q are the polynomials from Lemma 43,
$\mathcal{I}(A) = E[y/x] \qquad \mathcal{I}(B) = E[y+1/x] \qquad \mathcal{I}(C) = E[0/x] \qquad \mathcal{I}(D) = E$

$$\Sigma \mid \Gamma \vdash \mathbf{N}_x^p \multimap !_q !_{y<x}(A \multimap B) \multimap !_q C \multimap D$$

In this rule, $\mathcal{I}$ denotes the evident translation from SBAL to BLL.

Figure 3: Skewed iteration

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 3. *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.*

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

$$\mathbf{W}_x^{p,q,r} \stackrel{\text{def}}{=} (!_q \mathbf{N}_x^p \multimap \mathbf{F}_3^q) \otimes \mathbf{N}_x^r.$$

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 $(\mathbb{N} \to \{0,1,*\}) \times \mathbb{N}$ *represents the binary word* w, if $n > |w|$ holds and f satisfies the equation

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

We say that a function

$$g \colon (\mathbb{N} \to \{0,1,*\}) \times \mathbb{N} \longrightarrow (\mathbb{N} \to \{0,1,*\}) \times \mathbb{N}$$

represents a function $h \colon 2^* \to 2^*$ if, for all $w \in 2^*$, g maps any pair that represents w to a pair that represents $h(w)$.

Theorem 4 (Soundness)**.** *If $\vdash (!_{y<t} \mathbf{W}_x^{q,r,s}) \multimap \mathbf{W}_{p(x)}^{u,v,w}$ 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.*

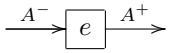
Theorem 5 (Completeness)**.** *For each* LOGSPACE *function $f \colon \{0,1\}^* \to \{0,1\}^*$, there exists in* SBAL+(SKEW) *a derivation of a sequent $\vdash (!_{y<t} \mathbf{W}_x^{q,r,s}) \multimap \mathbf{W}_{p(x)}^{u,v,w}$ whose underlying function represents f.*

Proofs appear in Section 3.7.

3 An Interaction Model for SBAL

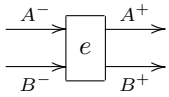
To prove Theorems 3–5, we show that second-order λ-terms with typing derivations in SBAL$^+$ can be compiled to space-efficient programs. In this section we define such a compilation method, which takes the form of an interpretation in a game semantic model. The import from game semantics is a description of computation with higher-order functions in terms of message-passing with simple messages. In the literature one can find a great variety of very general game semantic models. For our purposes, we do not need the full generality of these models. Here we use the framework known as Geometry of Interaction (GoI) Situation. Its relation to game semantics has been studied in [Abramsky and Jagadeesan, 1992].

In this paper we use one particular Geometry of Interaction situation, which captures a way of modelling computation by question/answer dialogues. This means that a type A is represented by two sets A^- and A^+, where A^- is a set of questions and A^+ is a set of answers. Elements of type A are represented by partial functions $e\colon A^- \to A^+$. Thus, elements can be viewed as boxes that can receive questions and that output answers to these questions.



Basic data types like **Bool** and **Nat** can be modelled in this way by letting the set of questions be a singleton set and letting the set of answers be the original set, e.g. $\mathbf{Nat}^- = \{*\}$ and $\mathbf{Nat}^+ = \mathbf{Nat}$.

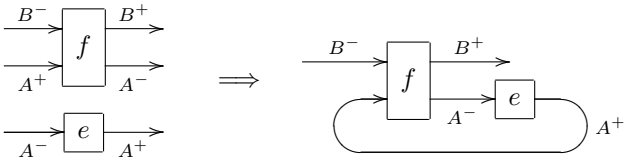
The GoI situation supports constructions that can be used to interpret multiplicative exponential intuitionistic linear logic[1]. The tensor product $A \otimes B$ is thought of as consisting of pairs of elements of A and B. Hence, a question to $A \otimes B$ is a question either to A or to B. As an answer to such a question we should expect a corresponding answer from A or B. Therefore, the set of questions $(A \otimes B)^-$ is $A^- + B^-$, the disjoint union of the question-sets for A and B. The set of answers $(A \otimes B)^+$ is the disjoint union $A^+ + B^+$. We depict the elements of $A \otimes B$ as follows.



More interesting is how function spaces are modelled in the GoI situation. The sets of questions and answers for $A \multimap B$ are defined so that an element of type $A \multimap B$, i.e. a function of type $(A \multimap B)^- \to (A \multimap B)^+$, explains how to answer questions in B, given that we already know how to answer questions in A. To this end, one can make the definitions $(A \multimap B)^- = A^+ + B^-$ and $(A \multimap B)^+ = A^- + B^+$. Then, an element $f\colon (A \multimap B)^- \to (A \multimap B)^+$ amounts to two functions $g\colon A^+ \to A^- + B^+$ and $h\colon B^- \to A^- + B^+$. If we know how to answer questions in A then the two functions g and h can be used in the following way to answer questions in B. Let q be a question in B^-. We pass this question q to h and as a result obtain either an answer in B^+ or a question

[1]It is not a full model, since it does not validate all the equations between proofs, see e.g. [Abramsky et al., 2002]. Nevertheless, each proof can be interpreted by a morphism in the category.

in A^-. In the first case we have our desired answer. In the second case we use the assumption that we can answer questions in A to obtain an answer in A^+ to the question returned by h. Now we pass this answer to g and we treat the result exactly like the result of h. In this way, g is iterated until it returns an answer in B^+. This way of computing an application has the following simple pictorial description.



The reader may have noticed that when a function asks for its argument, it has no way of storing any data that it could use again once the answer from the argument arrives. The ability to store some data can be added by modifying the sets of questions and answers for A such that each question and answer comes with some stored value. Define a new object $!A$ with $(!A)^- = \mathbb{N} \times A^-$ and $(!A)^+ = \mathbb{N} \times A^+$. A function that needs to store some data when passing a question to it argument can then be modelled as an element of type $!A \multimap B$ in the following way. Whenever the function wants to ask a question $q \in A^-$ of its argument, it packs up the question and the data s it wants to store in the pair $\langle s, q\rangle$ and passes this pair as the question to its argument. An answer from $!A$ will then typically have the form $\langle s, a\rangle$, so that the function gets back not only the answer to its question but also the value it wanted to store. We should point out that, at this point, there is no reason why the answer from $!A$ should be $\langle s, a\rangle$ and not $\langle t, a\rangle$ for some t other than s, but we will enforce this property later.

We use the Geometry of Interaction situation to interpret in it the proofs of SBAL in such a way that we can use the interpretation to compute the underlying functions of the proofs. Therefore, the interpretation amounts to a compilation of SBAL proofs to message passing networks of the kind depicted above. To establish space-bounds on the compiled programs, it then suffices to set up the model so that only messages of logarithmic size are being passed around and that the boxes that process the messages use no more than linear space in the size of messages.

3.1 Geometry of Interaction Situation

In this section we define the Geometry of Interaction situation used for the compilation of SBAL to space-efficient programs. In essence, this situation is that of sets and partial functions, which has been studied in detail in [Abramsky et al., 2002].

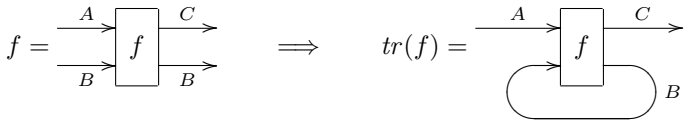
3.1.1 Sets and Partial Functions

The basis of the GoI situation is the category $\mathbb{B}$ of sets and partial functions. We assume that each object A of $\mathbb{B}$ is equipped with a total coding function $c\colon A \to 2^*$. For $a \in A$, we write $|a|$ for the length of the code $c(a)$. 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 element $a \in dom(f)$ to the code for $f(a) \in B$.

The category $\mathbb{B}$ has binary coproducts $+$ that are given by disjoint union. We define the coding function on $A+B$ by $c_{A+B}(inl(a)) = c_A(a)0$ and $c_{A+B}(inr(b)) = c_B(b)1$. The empty set $\emptyset$ is an initial object of $\mathbb{B}$. Furthermore, the category $\mathbb{B}$ 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 [Joyal et al., 1996]. In $\mathbb{B}$, the function $tr(f)$ is explicitly given by $tr_0 \circ inl$, where $tr_0\colon A+B \to C$ is the least function satisfying

$$tr_0(x) = \begin{cases} c & \text{if } f(x) = inl(c), \\ tr_0(inr(b)) & \text{if } f(x) = inr(b). \end{cases}$$

The trace operation can be depicted as follows.



Moreover, the natural numbers with binary encoding form an object $\mathbb{N}$ of $\mathbb{B}$. For any two objects A and B in $\mathbb{B}$, the set $A \times B$ of pairs can be made into an object of $\mathbb{B}$, such that $|\langle a, b\rangle| \in O(|a| + |b|)$. Likewise, for each finite set X, the set $V(X)$ of environments can be made into an object of $\mathbb{B}$, such that the coding function $V(X) \to 2^*$ 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 functions $\eta \mapsto \lfloor p[\eta] \rfloor$ and $\eta \mapsto \lceil p[\eta] \rceil$ of type $V(X) \to \mathbb{N}$ are computable in linear space.

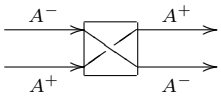
3.1.2 GoI Construction

The GoI construction on $\mathbb{B}$ defines a category $\mathbb{G}$ as follows.

Objects An object A of $\mathbb{G}$ is a pair (A^-, A^+) of two objects A^- and A^+ of $\mathbb{B}$.

Morphisms A morphism from A to B in $\mathbb{G}$ is a morphism of type $A^+ + B^- \to A^- + B^+$ in $\mathbb{B}$.

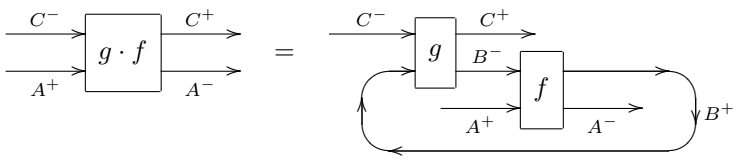
The identity morphism $id\colon A \to A$ is $[inr, inl]\colon A^+ + A^- \to A^- + A^+$.



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 the morphism

$$A^+ + B^- + C^- \xrightarrow{f+C^-} A^- + B^+ + C^- \xrightarrow{A^-+g} A^- + B^- + C^+$$

with respect to B^+. It may be depicted as follows.



As we have done in this definition, 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 $f\colon A^+ + B^- \to A^- + B^+$.

Before we define some of the structure of $\mathbb{G}$, we give a lemma about the space usage of the composition of two morphisms.

Lemma 6. *Let k be a natural number and let B be an object of $\mathbb{G}$ satisfying $|b| \leq k$ for all b in $B^- \cup B^+$. Let $f\colon A \to B$ and $g\colon B \to C$ be morphisms of $\mathbb{G}$. If f and g can be implemented using space $b_f(n)$ and $b_g(n)$ respectively, then $g \cdot f$ can be implemented using space $O(k + b_f(n+k) + b_g(n+k))$.*

Proof. The obvious implementation of tr_0 in the definition of trace above has the desired space-usage behaviour. □

The category $\mathbb{G}$ has been studied thoroughly, see e.g. [Haghverdi, 2000]. We define some of its structure in the rest of this section.

A symmetric monoidal structure $\otimes$ is defined on objects by

$$I = (\emptyset, \emptyset)$$
$$A \otimes B = (A^- + B^-, A^+ + B^+)$$

On morphisms, $f \otimes g$ is defined as $f + g$ on the underlying $\mathbb{B}$-morphisms.

There exists a functor $(-)^*\colon \mathbb{G}^{\mathrm{op}} \to \mathbb{G}$, which on objects acts by exchanging the sets of questions and answers.

$$A^* = (A^+, A^-)$$

For a morphism $f\colon A \to B$, the map $f^*\colon B^* \to A^*$ is given by

$$B^- + A^+ \xrightarrow{\cong} A^+ + B^- \xrightarrow{f} A^- + B^+ \xrightarrow{\cong} B^+ + A^-.$$

Lemma 7. *The category $\mathbb{G}$ is monoidal closed with respect to $\otimes$. A monoidal exponent $(-) \multimap (-)\colon \mathbb{G}^{\mathrm{op}} \times \mathbb{G} \to \mathbb{G}$ is defined by $(-) \multimap (-) = (-)^* \otimes (-)$. The application morphism $ev\colon (A \multimap B) \otimes A \to B$ is the canonical $\mathbb{B}$-function of its type.*

$$ev\colon A^- + B^+ + A^+ + B^- \longrightarrow A^+ + B^- + A^- + B^+$$

Lemma 8. *A functor $!\colon \mathbb{G} \to \mathbb{G}$ is defined on objects by $!A = (\mathbb{N} \times A^-, \mathbb{N} \times A^+)$. For a morphism $f\colon A \to B$, the map $!f$ is defined by*

$$\mathbb{N} \times A^+ + \mathbb{N} \times B^- \xrightarrow{\cong} \mathbb{N} \times (A^+ + B^-) \xrightarrow{\mathbb{N} \times f} \mathbb{N} \times (A^- + B^+) \xrightarrow{\cong} \mathbb{N} \times A^- + \mathbb{N} \times B^+.$$

Moreover, for each object A there are morphisms $der\colon !A \longrightarrow A$, $dig\colon !A \longrightarrow !!A$

and $contr\colon !A \longrightarrow !A \otimes !A$ *defined below.*

$$der_A\colon \mathbb{N} \times A^+ + A^- \longrightarrow \mathbb{N} \times A^- + A^+$$

$$der_A(x) = \begin{cases} inr(a) & \text{if } x = inl(n,a) \\ inl(0,q) & \text{if } x = inr(q) \end{cases}$$

$$dig_A\colon \mathbb{N} \times \mathbb{N} \times A^+ + \mathbb{N} \times A^- \longrightarrow \mathbb{N} \times \mathbb{N} \times A^- + \mathbb{N} \times A^+$$

$$dig_A(x) = \begin{cases} inr(pair(m,n),a) & \text{if } x = inl(m,n,a) \\ inl(fst(p),snd(p),q) & \text{if } x = inr(p,q) \end{cases}$$

$$contr_A\colon \mathbb{N} \times A^+ + \mathbb{N} \times A^- + \mathbb{N} \times A^- \longrightarrow \mathbb{N} \times A^- + \mathbb{N} \times A^+ + \mathbb{N} \times A^+$$

$$contr_A(x) = \begin{cases} inj_2(n/2,a) & \text{if } x = inj_1(n,a) \text{ and } n \text{ even} \\ inj_3((n-1)/2,a) & \text{if } x = inj_1(n,a) \text{ and } n \text{ odd} \\ inj_1(2n,q) & \text{if } x = inj_2(n,q) \\ inj_1(2n,q) & \text{if } x = inj_3(n,q). \end{cases}$$

Informally, the definitions in this lemma express that whenever we pass a question in A^- to $!A$, we may at the same time use one memory cell to store some data in. The map *der* expresses that we do not have to use this memory cell, *dig* expresses that two memory cells can be packed into one, and *contr* expresses that we can pack up a memory cell together with an additional bit that stores whether a question to $!A \otimes !A$ was a question to the left or the right component.

3.2 Realisability

Having defined the category $\mathbb{G}$, we now come to explaining how the terms in the second-order λ-calculus that are typeable in SBAL can be implemented in the GoI situation. We do this by introducing a realisability model, which formalises what it means for a function to be computed (realised) by a morphism in $\mathbb{G}$.

3.2.1 Second-Order Polymorphic λ-calculus

The logic SBAL can naturally be seen as a type system for the second-order λ-calculus (System F). Therefore, as the programs to be realised by the GoI, we take the functions from the second-order λ-calculus, for which we now fix the notation.

We use the second-order λ-calculus with explicit type abstraction and application, as defined in [Girard et al., 1989]. We write the typing judgements in this calculus as $\alpha_1\colon \mathsf{Type}, \ldots, \alpha_n\colon \mathsf{Type} \mid x_1\colon A_1, \ldots x_m\colon A_m \vdash_{\mathbb{F}} M\colon B$ and assume the standard equational theory with β- and η-equalities.

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 Σ. 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 $\mathbb{N}$. We write $\mathbb{F}$ for $\mathbb{F}(\emptyset)$.

We regard $\mathbb{F}$ as a 'set theory' and use set-theoretic notation for it. In particular, we write $a \in A$ for the global elements of A and we call the objects and morphisms also sets and functions.

3.2.2 Typed Realisers

In this section we define the realisability category $T(X)$. It has enough structure to model the multiplicative exponential fragment of SBAL and will be the basis of the construction of a model for full SBAL with universal quantification.

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

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

We require the objects to be such that there exist $c, d \in \mathbb{N}$ satisfying

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

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 $\eta \in V(X)$, all $a \in |A|$ and all $e\colon I \to \|A\|_\eta$, the following implication holds.

$$\eta, e \Vdash_A a \implies \eta, r_\eta \cdot e \Vdash_B f(a)$$

2. The function $\langle \eta, q \rangle \mapsto r_\eta(q)$ is LINSPACE-computable.

Lemma 9. *With the evident definitions of identity and composition on underlying sets, $T(X)$ becomes a category.*

Proof. The identity function on A is realised by $r_\eta = id_{\|A\|_\eta}$. For the composition of $f\colon A \to B$ and $g\colon B \to C$, let r be a realiser for f and s be a realiser for g. A realiser for $g \circ f$ is then given by $t_\eta = s_\eta \cdot r_\eta$. We need to verify that t satisfies the two conditions in the definition of morphisms. Property 1 follows immediately by unwinding the definitions. It remains to check property 2, that the mapping $\langle \eta, q \rangle \mapsto r_\eta(q)$ is computable in linear space. By definitions of the objects A, B and C, there exist natural numbers c and d, such that all sets $\|A\|_\eta^-$, $\|A\|_\eta^+$, $\|B\|_\eta^-$, $\|B\|_\eta^+$, $\|C\|_\eta^-$ and $\|C\|_\eta^+$ have only elements x satisfying $|x| \leq c \cdot |\eta| + d$. Write n_η for $c \cdot |\eta| + d$. By assumption, both $\langle \eta, x \rangle \mapsto r_\eta(x)$ and $\langle \eta, x \rangle \mapsto s_\eta(x)$ are computable in linear space. Hence, there exist c_r, d_r, c_s and d_s, such that both $\mathit{DSpace}(r_\eta(x)) \leq c_r(|\eta| + n_\eta) + d_r$ and $\mathit{DSpace}(s_\eta(x)) \leq c_s(|\eta| + n_\eta) + d_s$ holds. By Lemma 6 it follows that t can be implemented using space $O(n_\eta + c_r(|\eta| + n_\eta) + d_r + c_s(|\eta| + n_\eta) + d_s)$. This bound is evidently linear, thus completing the proof. □

Lemma 10. *Let $\sigma\colon X \to Y$ be a substitution and let A be an object of $T(Y)$. Then the equations below define an object $A[\sigma]$ of $T(X)$.*

$$\begin{aligned} |A[\sigma]| &= |A| \\ \|A[\sigma]\|_\eta &= \|A\|_{\sigma \circ \eta} \\ \eta, e \Vdash_{A[\sigma]} x &\iff \sigma \circ \eta, e \Vdash_A a \end{aligned}$$

We omit the routine proof.

Next we consider the structure of $T(X)$. We show that it has an affine symmetric monoidal structure $\otimes$ as well as a linear exponent $\multimap$.

Lemma 11. *The category $T(X)$ has a terminal object 1 defined by $|1| = 1$, $\|1\|_\eta = I$ and $(\eta, e \Vdash *)$ always.*

Proof. We have to show that, for each object A, there exists a unique morphism from A to 1. Define $t\colon A \to 1$ by $t(a) = *$. A realiser r for t is given by taking for r_η the empty function. The verification that r realises t is straightforward. That t is the only morphism of its type follows because $|1|$ is by definition a terminal object in $\mathbb{F}$. □

Lemma 12. *The category $T(X)$ has a symmetric monoidal structure $\otimes$. On objects it is defined by*

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

On morphisms, the underlying function of $f \otimes g$ is given by $(f \otimes g)(\langle a, b\rangle) = \langle f(a), g(b)\rangle$. Finally, the terminal object is a unit object for $\otimes$.

Proof. First we show that the morphism action of $\otimes$ is well-defined. Let r and s be a realisers of f and g respectively. It suffices to show that $\lambda\eta.\, r_\eta \otimes s_\eta$ is a realiser for $f \otimes g$. To this end we have:

$$\begin{aligned} \eta, e_a \otimes e_b \Vdash_{A\otimes B} \langle a, b\rangle &\iff (\eta, e_a \Vdash_A a) \ \wedge \ (\eta, e_b \Vdash_B b) \\ &\implies (\eta, r_\eta \cdot e_a \Vdash_C f(a)) \ \wedge \ (\eta, s_\eta \cdot e_b \Vdash_D g(b)) \\ &\implies (\eta, (r_\eta \cdot e_a) \otimes (s_\eta \cdot e_b) \Vdash_{C\otimes D} \langle f(a), g(b)\rangle) \\ &\iff (\eta, (r_\eta \otimes s_\eta) \cdot (e_a \otimes e_b) \Vdash_{C\otimes D} \langle f(a), g(b)\rangle) \end{aligned}$$

Since it is straightforward to see that the map $\langle \eta, x\rangle \mapsto (r_\eta \otimes s_\eta)(x)$ is computable in linear space, we have thus shown that $f \otimes g$ is indeed a morphism in $T(X)$.

The definition of $\otimes$ is functorial, since the identity of morphisms is determined only by the identity of the underlying functions, and on the underlying functions $\otimes$ agrees with the functor $\times$ in $\mathbb{F}$.

To show that $\otimes$ is a symmetric monoidal structure, it remains to define the coherent isomorphisms. We next give their definition.

$$\begin{aligned} a &\colon (A \otimes B) \otimes C \to A \otimes (B \otimes C) & a(\langle x, y\rangle, z) &= \langle x, \langle y, z\rangle\rangle \\ l &\colon 1 \otimes A \to A & l(x, *) &= x \\ l' &\colon A \to 1 \otimes A & l'(x) &= \langle *, x\rangle \\ s &\colon A \otimes B \to B \otimes A & s(x, y) &= \langle y, x\rangle \end{aligned}$$

The function a is realised by the coherent isomorphism

$$r^a_\eta \colon \|A\|_\eta \otimes (\|B\|_\eta \otimes \|C\|_\eta) \longrightarrow (\|A\|_\eta \otimes \|B\|_\eta) \otimes \|C\|_\eta$$

in $\mathbb{G}$. Likewise, the other morphisms are realised by the corresponding coherent isomorphism in $\mathbb{G}$. The verifications are routine.

Commutativity of the coherence diagrams follows in the same way as functoriality. □

Lemma 13. *The following definition of $\multimap$ makes $T(X)$ a monoidal closed category with respect to $\otimes$.*

$$\begin{aligned} |A \multimap B| &= |A| \Rightarrow |B| \\ \|A \multimap B\|_\eta &= \|A\|_\eta \multimap \|B\|_\eta \\ \eta, r \Vdash_{A\multimap B} f &\iff \textit{if } (\eta, e \Vdash_A a) \textit{ then } (\eta, ev \cdot (r \otimes e) \Vdash_B f(a)) \end{aligned}$$

Proof. Define the counit of the adjunction $\varepsilon_B \colon (A \multimap B) \otimes A \to B$ by $\varepsilon_B(f, a) = f(a)$. First we show that ε_B is a morphism in $T(X)$. We define a realiser $r \colon \Pi\eta \in V(X).\, \|A \multimap B\|_\eta \otimes \|A\|_\eta \longrightarrow \|B\|_\eta$ for ε_B by letting r_η be the application morphism in $\mathbb{G}$. Explicitly, $r_\eta = ev_{\|B\|_\eta}$ is the canonical function of type

$$\|A\|_\eta^- + \|B\|_\eta^+ + \|A\|_\eta^+ + \|B\|_\eta^- \longrightarrow \|A\|_\eta^+ + \|B\|_\eta^- + \|A\|_\eta^- + \|B\|_\eta^+.$$

This function is certainly computable in linear space, as required by the definition of morphisms, and the universal property of ε_B is easily verified.

□

The symmetric monoidal closed structure can be used to interpret multiplicative linear logic in $T(X)$. Next we show that $T(X)$ can also interpret the modality $!_{x<p}$ from Bounded Linear Logic.

Lemma 14. *For each resource polynomial $p \in P(X)$ and each $x \notin X$, there is a functor $!_{x<p} \colon T(X \cup \{x\}) \to T(X)$ that satisfies*

$$\begin{aligned} |!_{x<p}A| &= |A|, \\ \|!_{x<p}A\|_\eta &= \Big(\sum_{n<p[\eta]} \|A\|^-_{\eta[n/x]}, \sum_{n<p[\eta]} \|A\|^+_{\eta[n/x]} \Big), \\ \eta, e \Vdash_{!_{x<p}A} a \iff & \quad \forall n < p[\eta].\ \exists e' \colon I \to \|A\|_{\eta[n/x]}. \\ & \quad \Big(\forall q \in \|A\|^-_{\eta[n/x]}.\, e(n,q) = \langle n, e'(q) \rangle \Big) \\ & \quad \wedge \left(\eta[n/x], e' \Vdash_A a\right), \end{aligned}$$

and that acts as the identity on morphisms. Moreover, for all objects A and B of $T(X \cup \{x\})$, there exist the following morphisms.

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

Proof. We first show that the morphism-action of $!_{x<p}$ is well-defined. Let $f \colon A \to B$ be a morphism in $T(X \cup \{x\})$ that is realised by r. We have to find a realiser of $!_{x<p}f$. We show that s with s_η defined by

$$\begin{array}{ccc} \|!_{x<p}A\|^+_\eta + \|!_{x<p}B\|^-_\eta & \xrightarrow{\quad s_\eta \quad} & \|!_{x<p}A\|^-_\eta + \|!_{x<p}B\|^+_\eta \\ \cong \downarrow & & \uparrow \cong \\ \sum_{n<p[\eta]} (\|A\|^+_{\eta[n/x]} + \|B\|^-_{\eta[n/x]}) & \xrightarrow[\sum_{n<p[\eta]} r_{\eta[n/x]}]{} & \sum_{n<p[\eta]} (\|A\|^-_{\eta[n/x]} + \|B\|^+_{\eta[n/x]}) \end{array}$$

is such a realiser. We have to show the implication from $(\eta, e \Vdash_{!_{x<p}A} a)$ to $(\eta, s_\eta \cdot e \Vdash_{!_{x<p}B} f(a))$. Let $n < \eta(x)$. From $(\eta, e \Vdash_{!_{x<p}A} a)$ we obtain e' with $e(n,q) = \langle n, e'(q)\rangle$ and $(\eta[n/x], e' \Vdash_A a)$. By definition of s_η, we have $(s_\eta \cdot e)(n,q) = \langle n, r_{\eta[n/x]} \cdot e'(q)\rangle$. Furthermore, we have $(\eta[n/x], r_{\eta[n/x]} \cdot e' \Vdash_B f(a))$, since r is a realiser for f. Hence, we can show the required $(\eta, s_\eta \cdot e \Vdash_{!_{x<p}B} f(a))$ by using $r_{\eta[n/x]} \cdot e'$ to instantiate the existential quantifier. It only remains to show that the mapping $\langle \eta, q\rangle \mapsto s_\eta(q)$ remains in linear space. But this follows immediately because the isomorphisms in the definition of s and the mapping $\langle \eta, n\rangle \mapsto \eta[n/x]$ can be evaluated in linear space.

That $!_{x<p}$ satisfies the functoriality equations is trivial, as it acts as the identity on underlying functions.

It remains to show that the morphisms *der*, *distr*, *contr* and *dig* from the lemma all have realisers.

- $der\colon !_{x<1}A \longrightarrow A[0/x]$. A realiser is defined by:

$$r_\eta\colon \sum_{n<1} \|A\|^+_{\eta[n/x]} + \|A\|^-_{\eta[0/x]} \longrightarrow \sum_{n<1} \|A\|^-_{\eta[n/x]} + \|A\|^+_{\eta[0/x]}$$

$$\begin{aligned} r_\eta(inl(\langle i, a\rangle)) &= inl(a) \\ r_\eta(inr(q)) &= inr(0,q) \end{aligned}$$

  The mapping $\langle \eta, r\rangle \mapsto r_\eta$ is obviously in LINSPACE, since r_η does not depend on η.

- $distr\colon !_{x<p}A \otimes !_{x<p}B \longrightarrow !_{x<p}(A \otimes B)$. A realiser r_η has domain

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

  and range

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

  For r_η we choose the canonical function of its type.

- $contr\colon !_{x<p+q}A \longrightarrow !_{x<p}A \otimes !_{y<q}A[p+y/x]$. A realiser r_η has domain

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

  and its range is

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

We define r_η as follows:

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

The polynomial p is fixed. Further, the mapping $\eta \mapsto p[\eta]$ is computable in linear space. It follows from this fact that the mapping $\langle \eta, x \rangle \mapsto r_\eta(x)$ can be computed in linear space.

- $dig \colon !_{y<\sum_{x<p} q(x)} A \longrightarrow !_{x<p} !_{z<q(x)} A[z + \sum_{u<x} q(u)/y]$. A realiser r_η has domain

$$\sum_{n<\sum_{m<p[\eta]} q(m)[\eta]} \|A\|^+_{\eta[n/x]} + \sum_{n<p[\eta]} \sum_{m<q(n)[\eta]} \|A\|^-_{\eta[m+\sum_{u<n} q(u)[\eta]/y]}$$

and range

$$\sum_{n<\sum_{m<p[\eta]} q(m)[\eta]} \|A\|^-_{\eta[n/x]} + \sum_{n<p[\eta]} \sum_{m<q(n)[\eta]} \|A\|^+_{\eta[m+\sum_{u<n} q(u)[\eta]/y]}$$

We define the function r_η by:

$$r_\eta inl(n,a) = inr(x,y,a) \text{ where } n = \sum_{m<x-1} \lfloor q[m/x][\eta] \rfloor + y$$
$$r_\eta inr(x,y,z) = inl\left(\sum_{m<x-1} \lfloor q[m/x][\eta] \rfloor + y, z \right)$$

Notice that, in the first equation, x and y are because of their types uniquely determined by the equation.

To see $\langle \eta, x \rangle \mapsto r_\eta(x)$ can be computed in linear space, note first that the polynomials p and q are fixed. A linear space algorithm for r_η can then be given using the fact that fixed polynomials can be evaluated and the result can be stored in linear space.

□

Lemma 15. *Let A be an object in $T(X \cup \{x\})$, where $x \notin X$, and let p and q be resource polynomials in $P(X)$. If $p \leq q$, then the identity function is a morphism of type $!_{x<q} A \to !_{x<p} A$.*

Proof. A realiser $r \colon \Pi\eta \in V(X). \|!_{x<q} A\|_\eta \to \|!_{x<p} A\|_\eta$ is defined by:

$$r_\eta \colon \sum_{n<p[\eta]} \|A\|^-_{\eta[n/x]} + \sum_{n<q[\eta]} \|A\|^+_{\eta[n/x]} \longrightarrow \sum_{n<p[\eta]} \|A\|^+_{\eta[n/x]} + \sum_{n<q[\eta]} \|A\|^-_{\eta[n/x]}$$

$$r_\eta(inl(n,x)) = inr(n,x)$$
$$r_\eta(inr(n,x)) = \begin{cases} inr(n,x) & \text{if } n < p[\eta] \\ \bot & \text{otherwise} \end{cases}$$

Let $(\eta, e \Vdash_{!_{x<q} A} x)$. Then $r_\eta \cdot e$ behaves on pairs $\langle n, x \rangle$ that satisfy $n < p[\eta]$ exactly as does e. This implies the required $(\eta, r_\eta \cdot e \Vdash_{!_{x<p} A} x)$. □

3.2.3 Untyped Realisers

We have defined the category $T(X)$ and have found enough structure in it to interpret the quantifier-free fragment of SBAL. In the rest of this section we address the problem of modelling the universal quantifier.

A common way of modelling the universal quantifier in realisability models, e.g. [Jacobs, 1999, Dal Lago and Hofmann, 2005], is to require the elements of $\forall\alpha.\, A$ to be type-indexed families that are realised by a single realiser. This means that the underlying set of $\forall\alpha.\, A$ consists of families $(f(B) \in A[B/\alpha])_{B \text{ type}}$, i.e. functions that map each type B to an element of $A[B/\alpha]$. A realiser for such a function f is a single code that, for *all* types B, realises the value $f(B) \in A[B/\alpha]$. In essence, this is the definition we make in this paper. For the underlying set of $\forall\alpha \leq p.\, A$ we take the type $\forall\alpha.\, |A|$, whose elements are type-indexed families as above. For the realisation relation on $\forall\alpha \leq p.\, A$, we would like to define $(\eta, e \Vdash_{\forall\alpha\leq p.\, A} f)$ to hold if and only if $(\eta, e \Vdash_{A[B/\alpha]} f(|B|))$ holds for all objects B. We cannot quite make this definition in $T(X)$ however, for the technical reason that for two types B and B' the objects $\|A[B/\alpha]\|_\eta$ and $\|A[B'/\alpha]\|_\eta$ are different in general. The difference can be most easily seen in the case $A = \alpha$. Because of this difference, it does not make sense to define $(\eta, e \Vdash_{\forall\alpha\leq p.\, A} f)$ to hold if $(\eta, e \Vdash_{A[B/\alpha]} f(|B|))$ holds for all objects B, as e cannot be be a morphism of type $I \to \|A[B/\alpha]\|_\eta$ and $I \to \|A[B'/\alpha]\|_\eta$ at the same time.

In the interpretation of $\forall\alpha \leq p.\, A$, we avoid having to deal with different realising objects by using a generic object in which we can encode all types that the universal quantifier can be instantiated with. The idea is to encode the questions and answers of $\|B\|_\eta$ by natural numbers and to take as the generic object an object that has natural numbers as questions and answers. Concretely, we use the object U_η in $\mathbb{G}$, defined by $U_\eta^- = U_\eta^+ = \{n \in \mathbb{N} \mid n < p[\eta]\}$. The typing rules of SBAL are set up so that the universal quantifier $\forall\alpha \leq p.\, A$ can be instantiated only with types B whose realising objects can be encoded in U. Precisely, this means that there are coding and decoding morphisms $c_\eta \colon \|B\|_\eta \to U_\eta$ and $d_\eta \colon U_\eta \to \|B\|_\eta$ in $\mathbb{G}$ satisfying $d_\eta \cdot c_\eta = id$. For these encoding morphisms to exist, it is essential that the universal quantifier is bounded by the polynomial p. Using the object U_η, we can find a single object in which all the objects $\|A[B/\alpha]\|_\eta$ can be encoded, where B ranges over the types that can be used to instantiate the universal quantifier. Then we can model $\forall\alpha \leq p.\, A$ as type-indexed families that are realised by a single realiser.

In the following section we develop the technical tools for defining and using the encodings of questions and answers to natural numbers, as sketched above. For this purpose, we find it convenient to introduce a category $U(X)$, which differs from $T(X)$ only in that the realising objects $\|A\|_\eta$ are replaced by objects of the form U_η. Thus, $U(X)$ is a version of $T(X)$ with untyped realisers.

The definition of $U(X)$ is given below. In this definition we use the notation $\mathbb{N}_m$ for the set $\{n \in \mathbb{N} \mid n < m\}$. When clear from the context, we will write $\mathbb{N}_m$ also for the object $(\mathbb{N}_m, \mathbb{N}_m)$ of $\mathbb{G}$.

Objects An object A of $U(X)$ is a triple $(|A|, p, \Vdash)$. It consists of an object $|A|$ in $\mathbb{F}$, a resource polynomial $p \in V(X)$, and a relation

$$\Vdash \subseteq (\Sigma_{\eta\in V(X)} \mathbb{G}(I, \|A\|_\eta)) \times \mathbb{F}(1, |A|),$$

where we write $\|A\|_\eta$ for the object $(\mathbb{N}_{p[\eta]}, \mathbb{N}_{p[\eta]})$ in $\mathbb{G}$.

Morphisms A morphism $f\colon A \to B$ in $U(X)$ 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 $\eta \in V(X)$, all $a \in |A|$ and all $e\colon I \to \|A\|_\eta$, the following implication holds.

$$\eta, e \Vdash a \implies \eta, r_\eta \cdot e \Vdash f(a)$$

2. The function $\langle \eta, q \rangle \mapsto r_\eta(q)$ is LINSPACE-computable.

Corresponding to the syntactic concept of positive occurrences of variables, we introduce a semantic notion of positivity.

Definition 5. A resource variable $x \in X$ is *positive* in A if the implication

$$(\eta, e \Vdash_A a) \wedge (n \geq \eta(x)) \implies (\eta[n/x], e \Vdash_A a)$$

holds for all $a \in |A|$, all $\eta \in V(X)$, all $n \in \mathbb{N}$ and all $e\colon I \to \|A\|_\eta$. The variable x is *negative* in A if the implication below holds instead.

$$(\eta, e \Vdash_A a) \wedge (n \leq \eta(x)) \implies (\eta[n/x], e \Vdash_A a).$$

Next we show that the category $U(X)$ is equivalent to a full sub-category of $T(X)$ consisting of objects with realising objects that can be encoded effectively.

Definition 6. Let A be an object in $T(X)$ and let $p \in P(X)$. The object A is *p-encodable* if there exist functions

$$c\colon \Pi_{\eta \in V(X)} \|A\|_\eta \to \mathbb{N}_{p[\eta]}, \qquad d\colon \Pi_{\eta \in V(X)} \mathbb{N}_{p[\eta]} \to \|A\|_\eta,$$

such that $d_\eta \cdot c_\eta = id$ holds and that the maps $\langle \eta, x \rangle \mapsto c_\eta(x)$ and $\langle \eta, x \rangle \mapsto d_\eta(x)$ are both computable in linear space.

Lemma 16. *An object A is p-encodable if and only if there exist maps in $\mathbb{B}$*

$$\begin{aligned} c^-\colon \Pi_{\eta \in V(X)} \|A\|^-_\eta \to \mathbb{N}_{p[\eta]}, \qquad & d^-\colon \Pi_{\eta \in V(X)} \mathbb{N}_{p[\eta]} \to \|A\|^-_\eta, \\ c^+\colon \Pi_{\eta \in V(X)} \|A\|^+_\eta \to \mathbb{N}_{p[\eta]}, \qquad & d^+\colon \Pi_{\eta \in V(X)} \mathbb{N}_{p[\eta]} \to \|A\|^+_\eta, \end{aligned}$$

such that $d^- \circ c^- = id$ and $d^+ \circ c^+ = id$ hold and, for all $f \in \{c^-, d^-, c^+, d^+\}$, the map $\langle \eta, x \rangle \mapsto f_\eta(x)$ is computable in linear space.

We omit the routine proof.

Definition 7. Let $T_e(X)$ be the full sub-category of $T(X)$ in which each object is p-encodable for some $p \in P(X)$.

Lemma 17. *The categories $U(X)$ and $T_e(X)$ are equivalent.*

Proof. For each object A in $T_e(X)$ there exists a polynomial p_A such that A is p_A-encodable. We assume, for each object A, a choice of such a polynomial p_A. Define a functor $F\colon T_e(X) \to U(X)$ on objects by $|F(A)| = |A|$, $p_{F(A)} = p_A$ and $(\eta, e \Vdash_{F(A)} a) \iff (\eta, d_\eta \cdot e \Vdash_A a)$, and on morphisms as the identity. We show that F is a functor that is part of an equivalence of categories.

First we show that the morphism-action of F is well-defined. Let $f\colon A \to B$ be a morphism in $T_e(X)$ with realiser r. We have to show that $F(f) = f\colon FA \to FB$ also has a realiser. The following series of implications shows that $\lambda\eta.\, c^B_\eta \cdot r_\eta \cdot d^A_\eta$ is such a realiser.

$$\begin{aligned}
\eta, e \Vdash_{F(A)} a &\implies \eta, d^A_\eta \cdot e \Vdash_A a && \text{by definition}\\
&\implies \eta, r_\eta \cdot d^A_\eta \cdot e \Vdash_B f(a) && \text{since } r \text{ realises } f\\
&\implies \eta, d^B_\eta \cdot c^B_\eta \cdot r_\eta \cdot d^A_\eta \cdot e \Vdash_B f(a) && d^B_\eta \cdot c^B_\eta = id\\
&\implies \eta, (c^B_\eta \cdot r_\eta \cdot d^A_\eta) \cdot e \Vdash_{FB} f(a) && \text{by definition}
\end{aligned}$$

That the mapping $\langle \eta, x\rangle \mapsto (c^B_\eta \cdot r_\eta \cdot d^A_\eta)(x)$ is computable in linear space, follows immediately since linear space functions are closed under composition.

It remains to show that F is an equivalence. For this it suffices to show that F is full, faithful and essentially surjective on objects. That F is faithful is immediate from the definition. To show that it is full, let $f\colon F(A) \to F(B)$ be a morphism in $U(X)$ that is realised by r. We have to show that f is also a morphism of type $A \to B$ in $T_e(X)$. We have:

$$\begin{aligned}
\eta, e \Vdash_A a &\implies \eta, d_\eta \cdot c_\eta \cdot e \Vdash_A a && d_\eta \cdot c_\eta = id\\
&\implies \eta, c_\eta \cdot e \Vdash_{FA} a && \text{by definition}\\
&\implies \eta, r_\eta \cdot c_\eta \cdot e \Vdash_{FB} f(a) && \text{since } r \text{ realises } f\\
&\implies \eta, d_\eta \cdot r_\eta \cdot c_\eta \cdot e \Vdash_B f(a) && \text{by definition}
\end{aligned}$$

This chain of implications makes $\lambda\eta.\, d_\eta \cdot r_\eta \cdot c_\eta$ a realiser for $f\colon A \to B$, thus completing the proof that F is full.

It only remains to show that each object of $U(X)$ is isomorphic to an object in the image of F. But this follows immediately, since each object A of $U(X)$ can be made into a p_A-encodable object by taking $\|A\|_\eta = \mathbb{N}_{p_A[\eta]}$. □

Lemma 18. *The category $U(X)$ has a terminal object 1 defined by $|1| = \{*\}$, $p_1 = 0$ and $(\eta, e \Vdash *)$ always.*

The proof goes like that for Lemma 11.

Lemma 19. *The category $U(X)$ has an affine symmetric monoidal closed structure $(\otimes, \multimap)$, where the underlying sets and the realisability relations are defined as in $T(X)$ and the size polynomials are $p_{A\otimes B} = p_{A\multimap B} = 2 \cdot (p_A + p_B)$.*

Proof. It suffices to show that in $T(X)$ the objects $A \otimes B$ and $A \multimap B$ are both $2 \cdot (p_A + p_B)$-encodable, given that A is p_A-encodable and B is p_B-encodable. This is sufficient because then $T_e(X)$ is monoidal closed, which using the equivalence of $T_e(X)$ and $U(X)$ implies the statement of the lemma. Concretely, let $F\colon T_e(X) \to U(X)$ and $G\colon U(X) \to T_e(X)$ be functors that witness the equivalence from Lemma 17. The monoidal closed structure in $U(X)$ can then be defined by

$$\begin{aligned}
(-) \otimes_{U(X)} (-) &= F(G(-) \otimes_{T(X)} G(-)),\\
(-) \multimap_{U(X)} (-) &= F(G(-) \multimap_{T(X)} G(-)).
\end{aligned}$$

Hence, let A and B be objects of $T(X)$ such that A is p_A-encodable and B is p_B-encodable. To show that $A \otimes B$ and $A \multimap B$ are both $2 \cdot (p_A + p_B)$-encodable, we use the following coding functions $k_{a,b} \colon \mathbb{N}_{2(a+b)} \to \mathbb{N}_a + \mathbb{N}_b$ and $l_{a,b} \colon \mathbb{N}_a + \mathbb{N}_b \to \mathbb{N}_{2(a+b)}$ in $\mathbb{B}$.

$$k_{a,b}(m) = \begin{cases} inl(m/2) & \text{if } n \text{ is even and } m/2 < a, \\ inr((m-1)/2) & \text{if } n \text{ is odd and } (m-1)/2 < b, \\ \bot & \text{otherwise,} \end{cases}$$

$$l_{a,b}(m) = \begin{cases} 2n & \text{if } m = inl(n), \\ 2n+1 & \text{if } m = inr(n), \end{cases}$$

The mappings $\langle a, b, m\rangle \mapsto k_{a,b}(m)$ and $\langle a, b, m\rangle \mapsto l_{a,b}(m)$ are evidently computable in linear space. Using $k_{a,b}$ and $l_{a,b}$, it is easy to construct maps $k'_{a,b} \colon \mathbb{N}_{2(a+b)} \to \mathbb{N}_a \otimes \mathbb{N}_b$ and $l'_{a,b} \colon \mathbb{N}_a \otimes \mathbb{N}_b \to \mathbb{N}_{2(a+b)}$ in $\mathbb{G}$ that satisfy $k'_{a,b} \cdot l'_{a,b} = id$.

We spell out explicitly how the maps $k'_{a,b}$ and $l'_{a,b}$ are used in the definition of the coding and decoding maps for $A \multimap B$. The case for $A \otimes B$ is similar. We define explicitly:

$$c_\eta^{A \multimap B} \stackrel{\text{def}}{=} \|A\|_\eta^* \otimes \|B\|_\eta \xrightarrow{d^* \otimes c} \mathbb{N}_{p_A[\eta]} \otimes \mathbb{N}_{p_B[\eta]} \xrightarrow{l'} \mathbb{N}_{2(p_A[\eta]+p_B[\eta])}$$

$$d_\eta^{A \multimap B} \stackrel{\text{def}}{=} \mathbb{N}_{2(p_A[\eta]+p_B[\eta])} \xrightarrow{k'} \mathbb{N}_{p_A[\eta]} \otimes \mathbb{N}_{p_B[\eta]} \xrightarrow{c^* \otimes d} \|A\|_\eta^* \otimes \|B\|_\eta$$

The required equation $d^{A \multimap B} \cdot c^{A \multimap B} = id$ can be seen as follows.

$$\begin{aligned} d^{A \multimap B} \cdot c^{A \multimap B} &= (c^* \otimes d) \cdot l' \cdot k' \cdot (d^* \otimes c) \\ &= (c^* \otimes d) \cdot (d^* \otimes c) \\ &= (c^* \cdot d^*) \otimes (d \cdot c) \\ &= (d \cdot c)^* \otimes (d \cdot c) \\ &= id \end{aligned}$$

The condition for linear space computability is easy to see from the definition. □

Lemma 20. *Let X be a set of resource variables and $x \notin X$. For each resource polynomial $p \in P(X)$, there exists a functor $!_{x<p} \colon U(X \cup \{x\}) \to U(X)$ with underlying sets and realisation relation defined as in $T_e(X)$ and with size polynomial $p_{!_{x<p}A} = (p + p_A[p/x] + 1)^2$. Furthermore, there exist in $U(X)$ morphisms der, contr, dig and distr, as in Lemma 14.*

Proof. By the equivalence from Lemma 17, it suffices to show that for an object A of $T(X + \{x\})$ that is q-encodable, the object $!_{x<p}A$ is $(p + q[p/x] + 1)^2$-encodable. Then, if functors $F_X \colon T_e(X) \to U(X)$ and $G_X \colon U(X) \to T_e(X)$ witness the equivalence from Lemma 17, we can define $!_{x<p}A$ on $U(X)$ by $F_X(!_{x<p}G_{X\cup\{x\}}A)$.

Let A be a q-encodable object of $T(X)$. To show that $!_{x<p}A$ is $(p + q[p/x] + 1)^2$-encodable, it suffices, by Lemma 16, to define functions c^-, d^-, c^+ and d^+ that

encode and decode questions and answers. We give the details for c^- and d^- here. The other cases are similar. We use the following coding functions in $\mathbb{B}$.

$$k_\eta \colon \mathbb{N}_{((p+q[p/x]+1)^2)[\eta]} \to \mathbb{N}_{p[\eta]} \times \mathbb{N}_{q[p/x][\eta]}$$
$$k_\eta(m) = \begin{cases} \langle \mathit{fst}(m), \mathit{snd}(m) \rangle & \text{if } \mathit{fst}(m) < p[\eta] \text{ and } \mathit{snd}(m) < q[p/x][\eta] \\ \bot & \text{otherwise} \end{cases}$$

$$l_\eta \colon \mathbb{N}_{p[\eta]} \times \mathbb{N}_{q[p/x][\eta]} \to \mathbb{N}_{((p+q[p/x]+1)^2)[\eta]}$$
$$l_\eta(m,n) = \mathit{pair}(m,n)$$

We have $k_\eta \circ l_\eta = id$. Furthermore, we have functions

$$c'^-_\eta \colon \sum_{m<\eta(p)} \|A\|^-_{\eta[m/x]} \to \mathbb{N}_{p[\eta]} \times \mathbb{N}_{q[p/x][\eta]}$$
$$c'^-_\eta(m,n) = \langle m, c^{A^-}(n) \rangle$$
$$d'^-_\eta \colon \mathbb{N}_{p[\eta]} \times \mathbb{N}_{q[p/x][\eta]} \to \sum_{m<\eta(p)} \|A\|^-_{\eta[m/x]}$$
$$d'^-_\eta(m,n) = \begin{cases} \langle m, d^{A^-}_{\eta[m/x]}(n) \rangle & \text{if } n \le q[\eta[m/x]], \\ \bot & \text{otherwise.} \end{cases}$$

that satisfy $d'^-_\eta \circ c'^-_\eta = id$

We define $c^- \stackrel{\text{def}}{=} c'_\eta \circ k^-_\eta$ and $d^- \stackrel{\text{def}}{=} l^-_\eta \circ d'_\eta$. With this choice we have $d^- \circ c^- = id$. By Lemma 16, this completes the proof of $(p + q[p/x] + 1)^2$-encodability of $!_{x<p}A$.

It remains to define the morphisms *der*, *dig*, *contr* and *distr*. We give only the construction for *der*. The other cases are similar. First note that for each polynomial $p \in P(X)$ and each object A in $U(X \cup \{x\})$, we have $(F_{X\cup\{x\}}A)[p/x] = F_X(A[p/x])$. With this property, *der* in $U(X)$ can be defined from *dig* in $T_e(X)$ as the morphism in the conclusion of the following derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{der} \colon !_{x<1}(F_{X+\{x\}}A) \to (F_{X+\{x\}}A)[0/x] \text{ in } T_e(X)}{\mathit{der} \colon !_{x<1}(F_{X+\{x\}}A) \to F_X(A[0/x]) \text{ in } T_e(X)}}{G_X\,\mathit{der} \colon G_X !_{x<1}(F_{X+\{x\}}A) \to G_X F_X(A[0/x]) \text{ in } U(X)}}{G_X\,\mathit{der} \colon G_X !_{x<1}(F_{X+\{x\}}A) \to G_X F_X(A[0/x]) \text{ in } U(X)}}{i \circ G_X\,\mathit{der} \colon !_{x<1}A \to A[0/x] \text{ in } U(X)}\; i \colon G_X F_X \cong id$$

□

These lemmas show that $U(X)$ is as good a model as $T_e(X)$ for the multiplicative exponential fragment of SBAL.

Before we dive into modelling the full logic with universal quantification, we spell out concretely a few objects in $U(X)$ and give complexity results for $U(X)$. In particular, we define an object S_x of small values and give examples for its use.

Definition 8. Define an object S_x in $U(\{x\})$ by

$$\begin{aligned} |S_x| &= \mathbb{N}, \\ p_{S_x} &= x+1, \\ (\eta, e \Vdash_{S_x} n) &\iff (n \leq \eta(x)) \wedge (e(0) = n). \end{aligned}$$

Lemma 21. *There are morphisms* $zero\colon 1 \to S_x$ *and* $succ\colon S_x \to S_{x+1}$ *in* $U(\{x\})$*, whose underlying functions are the constant zero and the successor function respectively.*

We omit the straightforward proof.

Lemma 22. *Let* A *be a* p*-encodable object in* $U(X)$*. Then there exists a morphism*

$$case\colon !_p S_x \otimes A \otimes A \to A,$$

in $U(X \cup \{x\})$ *satisfying* $case(0, a, b) = a$ *and* $case(n, a, b) = b$ *for all* $n > 0$*.*

Proof. Let c^-, d^-, c^+ and d^+ be as obtained by Lemma 16 from the p-encodability of A. With these functions, a realiser r for *case* can be defined as follows. The domain of r_η is $\mathbb{N}_{p[\eta]} \times \mathbb{N}_{\eta(x)} + \|A\|^+_\eta + \|A\|^+_\eta + \|A\|^-_\eta$ and its range is $\mathbb{N}_{p[\eta]} \times \mathbb{N}_{\eta(x)} + \|A\|^-_\eta + \|A\|^-_\eta + \|A\|^+_\eta$. It is defined by

$$\begin{aligned} r_\eta(inj_1(q, n)) &= \begin{cases} inj_2(d^-(q)) & \text{if } n = 0, \\ inj_3(d^-(q)) & \text{if } n > 0, \end{cases} \\ r_\eta(inj_2(a)) &= inj_4(a), \\ r_\eta(inj_3(a)) &= inj_4(a), \\ r_\eta(inj_4(q)) &= inj_1(c^-(q), *). \end{aligned}$$

□

Lemma 23. *Let* A *in a* p*-encodable object in* $U(X \cup \{x\})$*. Then the following object has a global element, whose underlying function is the standard iteration combinator on natural numbers.*

$$!_3!_p!_x S_x \multimap !_{y<x}(A[y/x] \multimap A[y+1/x]) \multimap A[0/x] \multimap A$$

Proof. A realiser r_η has domain

$$\sum_{n<3} \sum_{c<p(x)[\eta]} \sum_{m<\eta(x)} \|S_x\|^+_\eta + \sum_{m<\eta(x)} (\|A\|^-_{\eta[m/x]} + \|A\|^+_{\eta[m+1/x]}) + \|A\|^+_{\eta[0/x]} + \|A\|^-_\eta$$

and range

$$\sum_{n<3} \sum_{c<p(x)[\eta]} \sum_{m<\eta(x)} \|S_x\|^-_\eta + \sum_{m<\eta(x)} (\|A\|^+_{\eta[m/x]} + \|A\|^-_{\eta[m+1/x]}) + \|A\|^-_{\eta[0/x]} + \|A\|^+_\eta.$$

Informally, the behaviour of r_η can be described as follows. When r_η receives a question in A it remembers this question and asks its S_x-argument for its value. If the answer from S_x is zero then r_η passes the original question to $A[0/x]$. If S_x gives an answer $n > 0$, then r_η passes the original question to the result type of the argument $A[n/x] \multimap A[n+1/x]$. This may result in a question to $A[n/x]$,

which r_η passes on to the result type of the argument $A[n-1/x] \multimap A[n/x]$. This may go on until a question is passed to the argument $A[0/x]$. When $A[0/x]$ sends an answer, r_η needs to know the value n of the argument S_x to decide whether the answer is the final answer (when $n = 0$) or whether the answer should be passed to $A[0/x] \multimap A[1/x]$ (when $n > 0$). Similarly, depending on the value n, an answer from $A[m/x]$ could either be the final answer or an answer for $A[m/x] \multimap A[m+1/x]$. Hence, when r_η receives an answer from $A[0/x]$ of $A[m/x]$, it remembers this answer and asks for the value of S_x. When it receives this value, it passes on the stored answer to the appropriate recipient.

This informal description is implemented by the definition below, in which c^-, d^-, c^+ and d^+ are obtained from the p-encodability of A by Lemma 16.

$$r_\eta(inj_1(0,c,m,n)) = \begin{cases} inj_3(d^-(c)) & \text{if } n = 0 \\ inj_2(n-1, inr(d^-(c))) & \text{if } n > 0 \end{cases}$$

$$r_\eta(inj_1(1,c,m,n)) = \begin{cases} inj_4(d^+(c)) & \text{if } n = 0 \\ inj_2(0, inl(d^+(c))) & \text{if } n > 0 \end{cases}$$

$$r_\eta(inj_1(2,c,m,n)) = \begin{cases} inj_4(d^+(c)) & \text{if } n = m+1 \\ inj_2(m+1, inl(d^+(c))) & \text{if } n > m+1 \end{cases}$$

$$r_\eta(inj_2(m, inl(q))) = \begin{cases} inj_3(q) & \text{if } m = 0 \\ inj_2(m-1, inr(q)) & \text{if } m > 0 \end{cases}$$

$$r_\eta(inj_2(m, inr(a)) = inj_1(2, e^+(a), m, *)$$

$$r_\eta(inj_3(a)) = inj_1(1, e^+(a), 0, *)$$

$$r_\eta(inj_4(q)) = inj_1(0, e^-(q), 0, *)$$

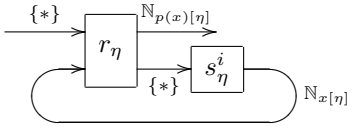
The mapping $\eta \mapsto r_\eta$ is constant and r_η is linear in $|\eta|$, since both m and n are smaller than $\eta(x)$. □

Proposition 24. *The underlying function of each morphism $!_q S_x \to S_{p(x)}$ in $U(\{x\})$ is linear space computable.*

Proof. Let $f\colon !_q S_x \to S_{p(x)}$ be a morphism in $U(\{x\})$ and let r be a realiser for it. For each $i \in \mathbb{N}$ and each $\eta \in V(\{x\})$ with $\eta(x) > i$, let $s^i_\eta\colon I \to \|!_q S_x\|_\eta$ be defined by

$$s^i_\eta\colon \mathbb{N}_{q[\eta]} \to \mathbb{N}_{q[\eta]} \times \mathbb{N}_{x[\eta]},$$
$$s^i_\eta(n) = \langle n, i\rangle$$

By definition we have $\eta, s^i_\eta \Vdash_{!_q S_x} i$ whenever $\eta(x) > i$. Since r realises f, this implies $[i+1/x], r_\eta \cdot s^i_\eta \Vdash_{S_{p(x)}} f(i)$. By the definition of S_x, this implies $(r_{[i+1/x]} \cdot s^i_{[i+1/x]})(*) = f(i)$. Hence, to show the assertion of the lemma, it suffices to show that the mapping $i \mapsto (r_{[i+1/x]} \cdot s^i_{[i+1/x]})(*)$ is computable in linear space. Clearly the mapping $i \mapsto [i+1/x]$ can be computed in linear space. A linear space algorithm for $\langle \eta, i\rangle \mapsto (r_\eta \cdot s^i_\eta)(*)$ is obtained as in the definition of composition in $T(X)$.

$(r \cdot s^i)_\eta =$ 

□

Proposition 25. *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.*

Proof. Let $f\colon !_q(S_x \multimap S_3) \otimes !_q S_x \to (!_q S_{p(x)} \multimap S_3) \otimes S_{p(x)}$ be a morphism in $U(\{x\})$ and let r be a realiser for it. For each $w \in \{0,1\}^*$, let $s^w\colon \Pi_{\eta \in V(X)} I \to \|!_q(S_x \multimap S_3)\|_\eta$ be defined by

$$\begin{aligned} s^w_\eta &\colon \mathbb{N}_{q[\eta]} \times \mathbb{N}_{x[\eta]} + \mathbb{N}_{q[\eta]} \to \mathbb{N}_{q[\eta]} + \mathbb{N}_{q[\eta]} \times \{0,1,*\}, \\ s^w_\eta(inr(s)) &= inl(s) \\ s^w_\eta(inl(s,i)) &= \begin{cases} \langle s, w_i \rangle & \text{if } i < |w|, \\ \langle s, * \rangle & \text{otherwise.} \end{cases} \end{aligned}$$

Define $t^w\colon \Pi_{\eta \in V(X)} I \to \|!_q S_x\|_\eta$ by

$$\begin{aligned} t^w_\eta &\colon \mathbb{N}_{q[\eta]} \to \mathbb{N}_{q[\eta]} \times \mathbb{N}_{x[\eta]}, \\ t^w_\eta(n) &= \begin{cases} \langle n, |w| \rangle & \text{if } |w| \in \mathbb{N}_{x[\eta]}, \text{ i.e. if } |w| < x[\eta], \\ \langle n, 0 \rangle & \text{otherwise.} \end{cases} \end{aligned}$$

Whenever $\eta(x) > |w|$ holds, we have $\eta, s^w_\eta \otimes t^w_\eta \Vdash \langle f_w, |w| \rangle$. Since r realises f, this implies $\eta, r_\eta \cdot (s^w_\eta \otimes t^w_\eta) \Vdash f(f_w, |w|)$.

By definition, both mappings $\langle w, \eta, q \rangle \mapsto s^w_\eta(q)$ and $\langle w, \eta, q \rangle \mapsto t^w_\eta(q)$ use space logarithmic in $|w|$ and linear in $|\eta|$ and $|q|$. The construction of composition in $T(x)$ is such that the same holds for the mapping $\langle w, \eta, q \rangle \mapsto (r \cdot (s^w \otimes t^w))(q)$.

Since f represents a LOGSPACE-function in the sense of Definition 4, we know that $f(f_w, |w|)$ is a pair $\langle g, n \rangle$, where g computes the individual bits of the output and n is an upper bound on the length of the output. We can therefore use

$$(r_\eta \cdot (s^w_\eta \otimes t^w_\eta))\colon \|!_q S_{p(x)} \multimap S_3\|^-_\eta + \|S_{p(x)}\|^-_\eta \longrightarrow \|!_q S_{p(x)} \multimap S_3\|^+_\eta + \|S_{p(x)}\|^+_\eta$$

first to compute an upper bound of the length of the output and then to compute the bits of the output iteratively. For instance, to compute the second bit of the output, we first send a question in $\|S_3\|^-$ to $(r_\eta \cdot (s^w_\eta \otimes t^w_\eta))$. If it responds with an answer, then this is the required bit. If it responds with a question $\langle s, * \rangle \in \|!_q S_{p(x)}\|^-_\eta$, then we respond with $\langle s, 1 \rangle$, since we are interested in the second bit of the output. By the realisation property, an answer in $\|S_3\|^+$ will be given after a finite number of iterations.

By construction, this algorithm uses space linear in η and logarithmic in w. Since for the above to work we can take $\eta = [|w| + 1/x]$, the whole procedure works in logarithmic space in w. □

3.3 Free Variables and Quantification

With the definition of the category $U(X)$ we have the basic model for SBAL in place. To define a model for full SBAL, it remains to account for formulae with free second-order variables and for universal quantification.

In this section we show 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_ρ indexed by environments ρ. 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_ρ share a single realiser. This uniformity condition is what makes the interpretation of the universal quantifier work.

In the rest of this section we define uniform families and exhibit enough structure for the interpretation of SBAL.

Definition 9. Let X be a set of resource variables, let $y_1, \ldots, y_n$ be resource variables not occurring in X and let $p \in P_n(X)$. Define $\mathbf{Obj}^p_{y_1,\ldots,y_n}(X)$ to be the set of all objects A in $U(X \cup \{y_1, \ldots, y_n\})$ that are positive in $y_1, \ldots y_n$ and whose size polynomial is $p(y_1, \ldots, y_n)$.

Definition 10. Let Σ be a second-order context with free resource variables in X. A Σ-*environment on* X is a function that assigns to each variable α, for which some declaration $\alpha \leq p \colon \mathsf{Fm}_n$ appears in Σ, a pair $\langle \langle y_1, \ldots, y_n \rangle, A \rangle$ such that the variables in $\{y_1, \ldots, y_n\}$ do not appear in X and A is an object in $\mathbf{Obj}^p_{y_1,\ldots,y_n}(X)$.

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

For a context Σ, 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 Σ-environment ρ 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 Σ 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 $A_\rho \stackrel{\text{def}}{=} (|A|[\rho_{\mathbb{F}}], p, \Vdash_\rho)$ yields an object in $U(X)$.

Morphisms A morphism $f \colon A \to B$ in $\mathrm{UFam}(\Sigma, X)$ 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 for all $\rho \in E(\Sigma, X)$ the following two conditions are satisfied.

1. $(\eta, e \Vdash_{A_\rho} x) \implies (\eta, r_\eta \cdot e \Vdash_{B_\rho} f[\rho](x))$
2. The mapping $\langle \eta, x \rangle \mapsto r_\eta(x)$ is computable in linear space.

In other words, the objects are families $(A_\rho)_{\rho \in E(\Sigma,X)}$ of objects in $U(X)$, and these families are subject to the uniformity condition that they arise from $|A|$ and p by $|A_\rho| = |A|[\rho_{\mathbb{F}}]$ and $p_{A_\rho} = p_A$. The morphisms are families $(f_\rho \colon A_\rho \to B_\rho)_{\rho \in E(\Sigma,X)}$ of morphisms in $U(X)$. These families must be uniform in the sense that they arise from a single map f in $\mathbb{F}$ by $f_\rho = f[\rho_{\mathbb{F}}]$ and that they have a single common realiser. When defining objects and morphisms in $\mathrm{UFam}(\Sigma, X)$, we often give only definitions for the families of objects and morphisms, omitting explicit definitions of $|A|$, p_A and f.

Lemma 26. *Let A be an object of* $\mathrm{UFam}((\Sigma, \alpha \leq p\colon \mathsf{Fm}_n), X)$. *Define the object* $\forall\alpha \leq p.\, A$ *in* $\mathrm{UFam}(\Sigma, X)$ *by:*

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

where $\vec{y} = y_1, \ldots, y_n$ is a vector of variables that do not occur in X.

Then this definition does not depend on the choice of variables $\vec{y}$ and can be extended to a functor $\mathrm{UFam}((\Sigma, \alpha \leq p\colon \mathsf{Fm}_n), X) \to \mathrm{UFam}(\Sigma, X)$.

Proof. We show only functoriality – independence from the choice of variables $\vec{y}$ is easy to see. A morphism $f\colon A \to B$ in $\mathrm{UFam}((\Sigma, \alpha \leq p\colon \mathsf{Fm}_n), X)$ is being mapped to the morphism $f' = \lambda x\colon (\forall\alpha.|A|).\, \Lambda\alpha.\, f(x(\alpha))$. For any realiser r of f, the chain of implications below shows that r is also a realiser for f'. This shows that f' is a morphism from $(\forall\alpha \leq p.\, A)$ to $(\forall\alpha \leq p.\, B)$ in $\mathrm{UFam}(\Sigma, X)$.

$$\begin{aligned} (\eta, e \Vdash_{(\forall\alpha\leq p.\, A)_\rho} x) &\implies \forall C \in \mathbf{Obj}^p_{\vec{y}}(X).\ \eta, e \Vdash_{A_{\rho[\alpha\mapsto\langle\vec{y},C\rangle]}} x(|C|) \\ &\implies \forall C \in \mathbf{Obj}^p_{\vec{y}}(X).\ \eta, r_\eta \cdot e \Vdash_{B_{\rho[\alpha\mapsto\langle\vec{y},C\rangle]}} f(x(|C|)) \\ &\implies \eta, r_\eta \cdot e \Vdash_{(\forall\alpha\leq p.\, B)_\rho} \Lambda\alpha.\, f(x(\alpha)) \end{aligned}$$

The functoriality equations then follow easily from the $\beta\eta$-equalities in $\mathbb{F}$. □

The next two lemmas state that the structure for $\otimes$, $\multimap$ and $!_{x<p}$ can be lifted from $U(X)$ to $\mathrm{UFam}(\Sigma, X)$ by a point-wise construction. We omit the proofs, as it is straightforward to check the uniformity of the realisers in the constructions for $U(X)$.

Lemma 27. *The category* $\mathrm{UFam}(\Sigma, X)$ *has a symmetric monoidal closed structure $(\otimes, \multimap)$. This closed structure is defined point-wise from the corresponding structure in $U(X)$, that is*

$$\begin{aligned} (A \otimes B)_\rho &= A_\rho \otimes B_\rho, \\ (A \multimap B)_\rho &= A_\rho \multimap B_\rho, \end{aligned}$$

and likewise for the morphisms.

Lemma 28. *For each second-order context Σ with free resource variables in X, each $x \notin X$ and each resource polynomial $p \in P(X)$, there exists a functor* $!_{x<p}\colon \mathrm{UFam}(\Sigma, X \cup \{x\}) \to \mathrm{UFam}(\Sigma, X)$ *defined point-wise from the corresponding functor for $U(X)$, i.e. $(!_{x<p}A)_\rho = !_{x<p}A_\rho$. Furthermore, there exist in* $\mathrm{UFam}(X)$ *morphisms der, contr, dig and distr, as in Lemma 14.*

The previous three lemmas exhibit the structure of $\mathrm{UFam}(\Sigma, X)$ that is being used for the interpretation of SBAL. We define the interpretation in the next section. For the interpretation of the rule (∀-L) we need the following lemmas.

Lemma 29. *Let m and n be natural numbers with $m \leq n$. Define morphisms* $i_{m,n}\colon \mathbb{N}_m \to \mathbb{N}_n$ *and* $j_{m,n}\colon \mathbb{N}_n \to \mathbb{N}_m$ *in* $\mathbb{G}$ *by:*

$$\begin{aligned} i_{m,n}\colon \mathbb{N}_n + \mathbb{N}_m &\longrightarrow \mathbb{N}_n + \mathbb{N}_m \\ i_{m,n}(\mathit{inl}(k)) &= \begin{cases} \mathit{inr}(k) & \text{if } k < m, \\ \bot & \text{otherwise,} \end{cases} \\ i_{m,n}(\mathit{inr}(k)) &= \mathit{inl}(k), \end{aligned}$$

$$
\begin{aligned}
j_{m,n} &\colon \mathbb{N}_m + \mathbb{N}_n \longrightarrow \mathbb{N}_m + \mathbb{N}_n \\
j_{m,n}(inl(k)) &= inr(k), \\
j_{m,n}(inr(k)) &= \begin{cases} inl(k) & \text{if } k < m, \\ \bot & \text{otherwise.} \end{cases}
\end{aligned}
$$

Then $j_{m,n} \cdot i_{m,n} = id$ holds.

Definition 11. Let A be an object of $\mathrm{UFam}(\Sigma, X)$ with size polynomial $p_A \in P(X)$. Let $p \in P(X)$ be a polynomial $p_A \leq p$. Then we write $A[p]$ for the following object of $\mathrm{UFam}(\Sigma, X)$.

$$
\begin{aligned}
|A[p]| &= |A| \\
p_{A[p]} &= p \\
(\eta, e \Vdash_{A[p]_\rho} x) &\iff (\eta, j_{p_A[\eta], p[\eta]} \cdot e \Vdash_{A_\rho} x)
\end{aligned}
$$

Note that, because of the monotonicity of the polynomials (Lemma 2), we do indeed have $p_A[\eta] \leq p[\eta]$, as required in the definition of $j_{p_A[\eta],p[\eta]}$. Furthermore, note that $(A[p])_\rho = A_\rho[p]$ holds by definition.

Lemma 30. *Let A be an object in $\mathrm{UFam}(\Sigma, X)$ with size polynomial $p_A \in P(X)$. Let $p \in P(X)$ be a polynomial with $p_A \leq p$. Then the identity function is an isomorphism between A and $A[p]$.*

Proof. A routine verification using Lemma 29 shows that $r_\eta = i_{p_A[\eta],p[\eta]}$ realises $id \colon A \to A[p]$ and that $s_\eta = j_{p_A[\eta],p[\eta]}$ realises $id \colon A[p] \to A$. □

3.4 Interpretation

3.4.1 Interpretation of Formulae

A formula A in context Σ and with free resource variables in X is interpreted as an object in $\mathrm{UFam}(\Sigma, X)$. Let Σ be a second-order context, let $\alpha \leq p \colon \mathsf{Fm}_n$ be a declaration in Σ and let $\vec{p}$ be a n-tuple of resource polynomials on X. Define an object $\alpha(\vec{p})$ in $\mathrm{UFam}(\Sigma, X)$ by $|\alpha(\vec{p})| = \alpha$, $p_{\alpha(\vec{p})} = p(\vec{p})$ and

$$
\eta, e \Vdash_{\alpha(\vec{p})_\rho} x \iff \text{if } (\rho(\alpha) = \langle \vec{y}, C \rangle) \text{ then } (\eta, e \Vdash_{C[\vec{p}/\vec{y}]} x).
$$

With this definition, the interpretation of formulae is given directly by the structure of UFam constructed in the last section.

$$
\begin{aligned}
[\![\alpha(\vec{p})]\!] &= \alpha(\vec{p}) \\
[\![A \otimes B]\!] &= [\![A]\!] \otimes [\![B]\!] \\
[\![A \multimap B]\!] &= [\![A]\!] \multimap [\![B]\!] \\
[\![!_{x<p} A]\!] &= !_{x<p} [\![A]\!] \\
[\![\forall \alpha \leq p.\, A]\!] &= \forall \alpha \leq p.\, [\![A]\!]
\end{aligned}
$$

For each formula A in context Σ with resource variables in X, we therefore have an object $[\![A]\!]$ of $\mathrm{UFam}(\Sigma, X)$. Formally, the definition of $[\![-]\!]$ is parametrised by Σ and X, so that it would be more precise to write $[\![A]\!]_{\Sigma,X}$ instead of $[\![A]\!]$, but since Σ and X will always be clear from the context, we omit these explicit annotations.

A context $\Gamma = A_1, \ldots, A_n$ is being interpreted by $[\![\Gamma]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!]$.

Lemma 31. *Let A be a formula in context Σ and with free resource variables in $X + \{\vec{y}\}$, such that the variables $\vec{y}$ are positive for A. Then the variables in $\{\vec{y}\}$ are also positive for the object $[\![A]\!]$ in* $\mathrm{UFam}(\Sigma, X + \{\vec{y}\})$.

Proof. We show by structural induction on the formula A that if x is positive respectively negative for A, then it is so also for the interpretation $[\![A]\!]$.

As a representative case, we consider the case for $A \multimap B$, where x is positive in A and negative in B. We have to show that x is negative in $A \multimap B$. We have to show the implication $(\eta, r \Vdash_{A\multimap B} f) \implies (\eta[n/x], r \Vdash_{A\multimap B} f)$ for all environments η and all numbers $n \leq \eta(x)$. But this follows directly thus:

$$\begin{aligned}
(\eta[n/x], e \Vdash_A a) &\implies (\eta, e \Vdash_A a) && x \text{ positive in } A \\
&\implies (\eta, r_\eta \cdot e \Vdash_B f(a)) && (\eta, r \Vdash_{A\multimap B} f) \\
&\implies (\eta[n/x], r_\eta \cdot e \Vdash_B f(a)) && x \text{ negative in } B
\end{aligned}$$

□

Lemma 32. *Let A be a formulae in context Σ and with free resource variables in X. If $\Sigma \vdash A \leq p$ is derivable, then the object $[\![A]\!]$ of* $\mathrm{UFam}(\Sigma, X)$ *has size polynomial $\leq p$.*

Proof. The proof goes by induction on the derivation of $\Sigma \vdash A \leq p$. The base case for second-order variables holds by definition of the Σ-environments. In all other cases, the polynomial in the derivation of $\Sigma \vdash A \leq p$ is just the size polynomial in the construction of the semantic structure. □

Lemma 33. *Let A be a formula with type variables in $\Sigma, \alpha \leq p : \mathsf{Fm}_n$ and resource variables in X. Let $i : B \cong C$ be an isomorphism in $U(X + \{y_1, \ldots, y_n\})$. Then there is an isomorphism*

$$[\![A]\!](i) : ([\![A]\!]_{\rho[\alpha \mapsto \langle \vec{y}, B\rangle]})_{\rho \in E(\Sigma, X)} \cong ([\![A]\!]_{\rho[\alpha \mapsto \langle \vec{y}, C\rangle]})_{\rho \in E(\Sigma, X)}$$

in $\mathrm{UFam}(\Sigma, X)$.

Proof Sketch. Note first that A can be considered as a formula in context $\Sigma, \alpha \leq (\lambda \vec{y}.\, p_B) : \mathsf{Fm}_n$, but also in context $\Sigma, \alpha \leq (\lambda \vec{y}.\, p_C) : \mathsf{Fm}_n$. This shows that domain and codomain of $[\![A]\!](i)$ are indeed well-defined.

The assertion of the lemma then follows by structural induction on the formula A. The base case is given by the isomorphism i and all other cases follow by functoriality of the respective semantic construction. □

Lemma 34 (Substitution Lemma)**.** *Let A be a formula with free type variables in Σ, $\alpha \leq p : \mathsf{Fm}_n$ and resource variables in X. Let $\Sigma \vdash B \leq p(x_1, \ldots, x_n)$ be derivable, where the variables in $\vec{x}$ are fresh for p, and let the variables in $\vec{x}$ be positive for B. The following equation holds in* $\mathrm{UFam}(\Sigma, X)$.

$$([\![A]\!]_{\rho[\alpha \mapsto \langle \vec{x}, [\![B]\!]_\rho\rangle]})_{\rho \in E(\Sigma, X)} = [\![A[\lambda \vec{x}.\, B/\alpha]]\!]$$

Proof. The proof goes by structural induction on the formula A. As representative cases, we spell out the cases for variables and linear functions.

- Case: A is a second-order variable $\beta(\vec{p})$. We have to show

$$(\llbracket \beta(\vec{p}) \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]})_{\rho \in E(V,X)} = \llbracket \beta(\vec{p})[\lambda \vec{x}.\, B/\alpha] \rrbracket.$$

  We continue by case distinction on whether $\alpha = \beta$ holds.

  Case $\beta = \alpha$. We have to show $(\llbracket \alpha(\vec{p}) \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]})_{\rho \in E(V,X)} = \llbracket B[\vec{p}/\vec{x}] \rrbracket$. This equation can be shown as follows.

$$\begin{aligned} \llbracket \alpha(\vec{p}) \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]} &= \alpha(\vec{p})_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]} && \text{by definition of the interpretation} \\ &= \llbracket B \rrbracket_\rho[\vec{p}/\vec{x}] && \text{by definition of } \alpha(\vec{p}) \end{aligned}$$

  Case $\beta \neq \alpha$. We have to show $(\llbracket \beta(\vec{p}) \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]})_{\rho \in E(V,X)} = \llbracket \beta(\vec{p}) \rrbracket$. If we let $\rho(\beta) = \langle \vec{y}, D \rangle$, then this equation can be show as follows.

$$\begin{aligned} \llbracket \beta(\vec{p}) \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]} &= \beta(\vec{p})_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]} && \text{Definition} \\ &= D[\vec{p}/\vec{y}] && \text{Definition des Objektes } \beta(\vec{p}) \\ &= \beta(\vec{p})_\rho && \text{Definition} \\ &= \llbracket \beta(\vec{p}) \rrbracket_\rho \end{aligned}$$

- Case: A is a linear function $C \multimap D$.

$$\begin{aligned} &\llbracket C \multimap D \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]} \\ &= \llbracket C \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]} \multimap \llbracket D \rrbracket_{\rho[\alpha \mapsto \langle \vec{x}, \llbracket B \rrbracket_\rho \rangle]} && \text{by definition} \\ &= \llbracket C[\lambda \vec{x}.\, B/\alpha] \rrbracket_\rho \multimap \llbracket D[\lambda \vec{x}.\, B/\alpha] \rrbracket_\rho && \text{by induction hypothesis} \\ &= \llbracket C[\lambda \vec{x}.\, B/\alpha] \multimap D[\lambda \vec{x}.\, B/\alpha] \rrbracket_\rho && \text{by definition} \\ &= \llbracket (C \multimap D)[\lambda \vec{x}.\, B/\alpha] \rrbracket_\rho && \text{property of substitution} \end{aligned}$$

□

3.4.2 Interpretation of Proofs

Lemma 35. *Let the sequent $\Sigma \vdash A \leq A'$ with resource variables in X be derivable. Then there is a morphism $\llbracket A \rrbracket \to \llbracket A' \rrbracket$ in $\mathrm{UFam}(\Sigma, X)$, whose underlying function is the identity.*

Proof. The proof goes by induction on the derivation of $\Sigma \vdash A \leq A'$. We show here the cases for variables, for the modality and for universal quantification.

- Case $\alpha(p_1, \ldots, p_n) \leq \alpha(q_1, \ldots, q_n)$ with $p_i \leq q_i$ for all$i \in \{1 \ldots n\}$. Let $\rho \in E(\Sigma, X)$. By definition of $E(\Sigma, X)$, we know that $\rho(\alpha)$ is a pair $\langle \vec{y}, C \rangle$, where $\vec{y}$ is positive for C. By the semantic definition of positivity, this implies that the identity is a morphism of type $\alpha(\vec{p})_\rho = C_\rho[\vec{p}/\vec{y}] \to C_\rho[\vec{q}/\vec{y}] = \alpha(\vec{q})$. The required result follows from this, since we have both $\llbracket A \rrbracket_\rho = \alpha(\vec{p})_\rho$ and $\llbracket A' \rrbracket_\rho = \alpha(\vec{q})_\rho$.
- Case $!_{x \leq p} A \leq !_{x \leq p} A'$ with $p \leq q$ and $\Sigma \vdash A \leq A'$. By induction hypothesis, the identity is a morphism of type $intA \to \llbracket A' \rrbracket$. By functoriality of $!_{x \leq q}$, this implies that the identity is a morphism of type $\llbracket !_{x \leq q} A \rrbracket \to \llbracket !_{x \leq q} A' \rrbracket$. By Lemma 15, we have a map $\llbracket !_{x \leq q} A' \rrbracket \to \llbracket !_{x \leq p} A' \rrbracket$, whose underlying function is the identity. Composition yields the required result.

- Case $(\forall\alpha \leq q.\, A) \leq (\forall\alpha \leq p.\, A')$ with $A \leq A'$. The induction hypothesis, we have a map $[\![A]\!] \to [\![A']\!]$ in $\mathrm{UFam}((\Sigma,\, \alpha \leq p), X)$, from which the required morphism $[\![\forall\alpha \leq p.\, A]\!] \to [\![\forall\alpha \leq p.\, A']\!]$ follows by functoriality.

□

We interpret a sequent $\Sigma \mid \Gamma \vdash A$ with resource variables in X as a morphism in $[\![\Gamma]\!] \longrightarrow [\![A]\!]$ in $\mathrm{UFam}(\Sigma, X)$. The interpretation of sequents is given by induction on the derivation. We make a case distinction on the last rule in a derivation.

- The case for (Axiom) follows by Lemma 35.
- The case for (Cut) follows by composition.
- Rules ($\otimes$-L), ($\otimes$-R), (Weakening), ($\multimap$-L) and ($\multimap$-R) are interpreted by the affine symmetric monoidal closed structure of $\mathrm{UFam}(\Sigma, X)$.
- ($\forall$-R)

$$(\forall\text{-R})\ \frac{\Sigma,\, \alpha \leq p\colon \mathsf{Fm}_n \mid \Gamma \vdash A}{\Sigma \mid \Gamma \vdash \forall\alpha \leq p\colon \mathsf{Fm}_n.\, A}$$

  The induction hypothesis gives us a morphism $f\colon [\![\Gamma]\!] \longrightarrow [\![A]\!]$ in the category $\mathrm{UFam}((\Sigma, \alpha \leq p\colon \mathsf{Fm}_n), X)$. We have to define a morphism $g\colon [\![\Gamma]\!] \longrightarrow (\forall\alpha \leq p\colon \mathsf{Fm}_n.\, [\![A]\!])$ in $\mathrm{UFam}(\Sigma, X)$. Let $g = \lambda\gamma\colon |\Gamma|.\, \Lambda\alpha.\, f(\gamma)$. To show that this indeed defines a morphism, we show that each realiser for f is also a realiser for g. To do this, it suffices to show, for all $\rho \in E(\Sigma, X)$, the implication

$$\eta, e \Vdash_{\Gamma_\rho} \gamma \implies \eta, r_\eta \cdot e \Vdash_{(\forall\alpha \leq p.\, A)_\rho} g[\rho_{\mathbb{F}}](\gamma).$$

  holds. Let $(\eta, e \Vdash_{\Gamma_\rho} \gamma)$. To show $(\eta, e \Vdash_{(\forall\alpha \leq p.\, A)_\rho} g[\rho_{\mathbb{F}}](\gamma))$, we have to show that

$$(\eta, r_\eta \cdot e \Vdash_{A_{\rho[\alpha \mapsto \langle \vec{y}, B\rangle]}} g[\rho_{\mathbb{F}}](\gamma)(|B|))$$

  holds for all $B \in \mathbf{Obj}^p_{\vec{y}}(X)$. let B be such an object. The assumption $(\eta, e \Vdash_{\Gamma_\rho} \gamma)$ implies $(\eta, e \Vdash_{\Gamma_{\rho[\alpha \mapsto \langle \vec{y}, B\rangle]}} \gamma)$, since α is not free in Γ. Since r is a realiser for f, this implies

$$(\eta, r_\eta \cdot e \Vdash_{\Gamma_{\rho[\alpha \mapsto \langle \vec{y}, B\rangle]}} f[\rho_{\mathbb{F}}][|B|/\alpha](\gamma)).$$

  But we have

$$\begin{aligned} g[\rho_{\mathbb{F}}](\gamma)(|B|) &= (\lambda\gamma.\, \Lambda\alpha.\, f(\gamma))[\rho_{\mathbb{F}}](\gamma)(|B|) \\ &= (\lambda\gamma.\, \Lambda\alpha.\, f[\rho_{\mathbb{F}}](\gamma))(\gamma)(|B|) \\ &= f[\rho_{\mathbb{F}}][|B|/\alpha](\gamma), \end{aligned}$$

  so that we have shown the assertion.

- ($\forall$-L)

$$(\forall\text{-L})\ \frac{\Sigma \vdash B \leq p(\vec{y}) \qquad \Sigma \mid \Gamma, A[\lambda\vec{y}.\, B/\alpha] \vdash C}{\Sigma \mid \Gamma, \forall\alpha \leq p.\, A \vdash C}$$

Let A be an object in $\mathrm{UFam}((\Sigma,\ \alpha \leq p\colon \mathsf{Fm}_n), X)$. By Lemma 32, the interpretation $[\![B]\!]$ is an object of $\mathrm{UFam}(\Sigma, X + \{\vec{y}\})$ with size bound $\leq p(\vec{y})$. By Lemma 31, the variables $\vec{y}$ are positive for $[\![B]\!]$. Hence, an the assignment

$$C_\rho \stackrel{\text{def}}{=} [\![A]\!]_{\rho[\alpha \mapsto \langle \vec{y}, [\![B]\!][p(\vec{y})]_\rho]}$$

defines an object of $\mathrm{UFam}(\Sigma, X)$. Recall the notation $A[p]$ from Definition 11.

For the interpretation of the rule, we first define $f\colon [\![\forall \alpha \leq p.\, A]\!] \to C$ in $\mathrm{UFam}(\Sigma, X)$ by $f_\rho(g) = g(|B|)$. It is realised by the identity, since whenever we have $(\eta, e \Vdash_{[\![\forall \alpha \leq p.\ A]\!]_\rho} g)$, then we also have $\eta, e \Vdash_{C_\rho} g(|B|)$, by definition of the realisation relation on $\forall \alpha \leq p.\, A$.

By Lemma 30, we have an isomorphism $[\![B]\!] \cong [\![B]\!][p(\vec{y})]$. With Lemma 33, this can be lifted to an isomorphism between C and $([\![A]\!]_{\rho[\alpha \mapsto \langle \vec{y}, [\![B]\!]_\rho \rangle]})_{\rho \in E(\Sigma, X)}$. Furthermore, by the substitution lemma we know that this object is isomorphic to $[\![A[\lambda \vec{y}.B/\alpha]]\!]$. By composition of f with these isomorphisms, we obtain the required map of type $[\![\forall \alpha \leq p.\, A]\!] \to [\![A[\lambda \vec{y}.B/\alpha]]\!]$.

- The rules (Dereliction), (Contraction), (Functoriality) and the Storage-Axiom follow by Lemma 28.

The interpretation is defined such that each derivation is interpreted by a morphism, whose underlying function is the function corresponding to this derivation under the Curry-Howard isomorphism.

Proposition 36. *Let π be a derivation of $\Sigma \mid \Gamma \vdash A$ with free resource variables in X. Let $\Sigma_{\mathbb{F}} \mid \Gamma_{\mathbb{F}} \vdash M\colon A_{\mathbb{F}}$ be the derivation obtained by translating π to System F. Then M is the underlying function of the interpretation of the morphism in $\mathrm{UFam}(\Sigma, X)$ that interprets π.*

3.5 Encoding Small Data Types

In Definition 3, small data types are defined using impredicative-style encodings. Therefore, the underlying sets of $\mathbf{F}$ and $\mathbf{N}$ therefore are sets of functions, which makes their interpretation somewhat awkward to work with, for instance to prove complexity results. Instead of working directly with the interpretations of the impredicatively coded small data types, we find it convenient to work with the type S_x introduced in Section 3.2.3 and to encode all other small data types in it. In this section we show how small data types can be encoded as values of S_x.

Lemma 37. *Let $k \in \mathbb{N}$ and let $p \in P(X)$ be a resource polynomial with $p \geq k$. Then there are morphisms $c\colon \mathbf{F}_k^p \to S_k$ and $d\colon !_p S_k \to \mathbf{F}_l^p$ in $U(X \cup \{x\})$, whose underlying functions are an isomorphism.*

We omit the proof of this lemma, since it is similar to the proof of the following one.

Lemma 38. *For each resource polynomial $p \in P_1(X)$, there are in $U(X \cup \{x\})$ morphisms $c\colon \mathbf{N}_x^{\lambda x.x} \to S_x$ and $d\colon !_{3x \cdot p(x)} S_x \to \mathbf{N}_x^p$, whose underlying functions are an isomorphism.*

Proof. For the proof we use an extension of SBAL with the concrete morphisms for small numbers from Lemmas 21–23.

The morphism c arises from the derivation

$$\dfrac{\dfrac{}{\vdash S_z \leq z} \qquad \dfrac{\text{zero and successor (Lemma 21)}}{!_{y\leq x}(S_y \multimap S_{y+1}) \multimap S_0 \multimap S_x \vdash S_x}}{\dfrac{\forall \alpha \leq (\lambda x.x).\, !_{y\leq x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x) \vdash S_x}{\mathbf{N}_x^{\lambda x.x} \vdash S_x,}}$$

and d arises from the derivation

$$\dfrac{\dfrac{\text{Iteration for } S_x \text{ (Lemma 23)}}{\alpha \leq p\colon \mathsf{Fm}_1 \mid !_3 !_{p(x)} !_x S_x \vdash !_{y\leq x}(\alpha(y) \multimap \alpha(y+1)) \multimap \alpha(0) \multimap \alpha(x)}}{!_3 !_{p(x)} !_x S_x \vdash \mathbf{N}_x^p.}$$

These proofs are such that the underlying functions of c and d satisfy the equations

$$c(\lambda C.\, \lambda s.\, \lambda z.\, s^n(z)) = n, \qquad\qquad d(n) = \lambda C.\, \lambda s.\, \lambda z.\, s^n(z).$$

Hence, they constitute an isomorphism of the underlying sets. □

Finally, for the encoding of tuples, we use an extension of S_x to tuples.

Definition 12. For each n, define an object $S_{x_1,\ldots,x_n}$ in $U(\{x_1,\ldots,x_n\})$ by

$$\begin{aligned} |S_{\vec{x}}| &= \mathbb{N}^n, \\ p_{S_x} &= \langle \vec{x} \rangle, \\ (\eta, e \Vdash_{S_{\vec{x}}} \vec{k}) &\iff (\forall i \leq n.\, k_i \leq \eta(x_i)) \wedge (e(0) = \mathit{tuple}(\vec{k})), \end{aligned}$$

where $\langle \vec{x} \rangle$ is defined inductively by $\langle\rangle = 0$ and $\langle x, \vec{y} \rangle = ((x+1) + \langle \vec{y} \rangle + 1)^2$, and where $\mathit{tuple}(\vec{k})$ is defined to be $\mathit{pair}(k_1, \mathit{pair}(k_2, \ldots, \mathit{pair}(k_{n-1}, k_n)\ldots))$.

It is evident that there are pairing and unpairing morphisms $S_{\vec{p}} \otimes S_{\vec{q}} \to S_{\vec{p},\vec{q}}$ and $S_{\vec{p},\vec{q}} \to S_{\vec{p}} \otimes S_{\vec{q}}$.

Definition 13. The *size of a small data type* is a tuple of polynomials, defined inductively by the following clauses.

- The size of $\mathbf{F}_k^p$ is k;
- The size of $\mathbf{N}_p^q$ is p; and
- if $\vec{x}$ is the size of A and $\vec{y}$ is the size of B, then $\vec{x}, \vec{y}$ is the size of $A \otimes B$.

We write $\mathit{size}(A)$ for the size of a small data type A.

Lemma 39. *Let A be a small data type. Then there exist resource polynomials p and q and morphisms $c_A\colon !_p A \to S_{\mathit{size}(A)}$ and $d_A\colon !_q S_{\mathit{size}(A)} \to A$ in $U(X)$, whose underlying functions are an isomorphism.*

Proof. By induction on the definition of the small data type A, using Lemmas 37 and 38. □

We will use this lemma in the following two sections for the interpretation of skewed iteration and for the proofs of the complexity results.

3.6 Interpreting Skewed Iteration

Lemma 40 (Skewed Iteration on S_x)**.** *For each polynomial $p \in P_1(X)$ and each polynomial $i \in P(X)$, there exists a polynomial $q \in P(X)$, such that the following object has a global element, whose underlying function is the standard iteration combinator on natural numbers.*

$$S_x \multimap !_q!_{y<x}(!_i S_{p(y)} \multimap S_{p(y+1)}) \multimap !_q S_{p(0)} \multimap S_{p(x)}$$

Proof. The idea of how this iteration combinator is implemented is explained informally in Section 2.2. To make it precise, it suffices to construct a morphism of type

$$S_x \multimap !_{p(x)}!_x!_x!_{y<x}(!_i S_{p(y)} \multimap S_{p(y+1)}) \multimap !_x S_{p(0)} \multimap S_{p(x)}.$$

We use the string of modalities $!_{p(x)}!_x!_x$ to store tuples $\langle sa, sd, d\rangle$ that encode some stored answer (sa), the depth of the stored answer (sd) and the total depth of the recursion (d).

A realiser r_η has the following domain, in which we write M for the set $\mathbb{N}_{p(x)[\eta]} \times \mathbb{N}_{x[\eta]} \times \mathbb{N}_{x[\eta]}$. We write $m.sa$ for $\pi_1(m)$, $m.sd$ for $\pi_2(m)$, and $m.d$ for $\pi_3(m)$. Furthermore, we write A for $S_{p(x)}$.

$$\|S_x\|^+_\eta + M \times \sum_{n<\eta(x)} \left(\mathbb{N}_{i[\eta]} \times \|A\|^-_{\eta[n/x]} + \|A\|^+_{\eta[n+1/x]}\right) + \mathbb{N}_{x[\eta]} \times \|A\|^+_{\eta[0/x]} + \|A\|^-_\eta$$

The range of r_η is

$$\|S_x\|^-_\eta + M \times \sum_{n<\eta(x)} \left(\mathbb{N}_{i[\eta]} \times \|A\|^+_{\eta[n/x]} + \|A\|^-_{\eta[n+1/x]}\right) + \mathbb{N}_{x[\eta]} \times \|A\|^-_{\eta[0/x]} + \|A\|^+_\eta.$$

The realiser r_η is defined as follows.

$$\begin{aligned}
r_\eta(inj_1(n)) &= inj_3(n, *) \\
r_\eta(inj_2(\langle sa, sd, d\rangle, n, inl(s, *))) &= \begin{cases} inj_3(d, inr(*)) & \text{if } n = 0 \\ inj_2(\langle sa, sd, d\rangle, n-1, inl(s, sa)) & \text{if } n > 0 \end{cases} \\
r_\eta(inj_2(\langle sa, sd, d\rangle, n, inr(a))) &= \begin{cases} inj_4(a) & \text{if } d = n+1 \\ inj_2(\langle a, n, d\rangle, n+1, inr(*)) & \text{otherwise} \end{cases} \\
r_\eta(inj_3(n, a)) &= \begin{cases} inj_4(a) & \text{if } n = 0 \\ inj_2(\langle a, 0, n\rangle, 0, inr(*)) & \text{if } n > 0 \end{cases} \\
r_\eta(inj_4(q)) &= inj_1(*)
\end{aligned}$$

The realiser works as follows. Upon an initial question, r_η asks for its first argument S_x. The answer from this argument is the iteration depth d. This value is stored throughout the rest of the computation. After receiving this answer, r_η computes the base case by asking $A[0/x]$ for its value. When receiving the answer a, the realiser knows that the value after zero iteration steps is a. Therefore it stores $\langle a, 0, d\rangle$. Now r_η asks the step function $!_i A[0/x] \multimap A[1/x]$ for the value of $A[1/x]$. If the step function queries its argument, then r_η answers this question from memory. Note that because the set $\|S_x\|^-_\eta$ contains exactly

one question, the step function can only ask for the value that has been stored. When the answer a from $A[1/x]$ arrives, r_η updates its store to $\langle a, 1, d\rangle$. This triple represents the information that the answer after one iteration is a and that the total number of iterations is d. This process is continued until we reach the state where $\langle a, d, d\rangle$ would be stored. Then a is the final answer.

Note that for the correctness of this implementation it is essential that the realising object of S_x has exactly one question. □

This proof can be generalised to the type $S_{\vec{p}}$ of vectors of small types, giving us:

Lemma 41. *For each n-tuple $\vec{p}$ of polynomials in $P_1(X)$ and each polynomial $i \in P(X)$, there exists a polynomial $q \in P(X)$, such that the following object has a global element whose underlying function is the standard iteration combinator on natural numbers.*

$$S_x \multimap !_q!_{y<x}(!_i S_{\vec{p}(y)} \multimap S_{\vec{p}(y+1)}) \multimap !_q S_{\vec{p}(0)} \multimap S_{\vec{p}(x)}$$

Lemma 42. *Let A and B be small data types with resource variables in X. Then there exists a morphism*

$$t\colon !_p(A \multimap B) \to (!_p S_{size(A)} \multimap S_{size(B)})$$

in $U(X)$. Moreover, the underlying function of t satisfies the equation $t(f) = c_B \circ f \circ d_A$, where c_B and d_A are coding and decoding functions from Lemma 39.

Proof. We define the morphism using the following derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!_q B \longrightarrow S_{size(B)}}\text{ Lemma 39}}{!_q((A \multimap B) \otimes A) \longrightarrow S_{size(B)}}\text{ pre-composition with application}}{!_q((A \multimap B) \otimes !_p S_{size(A)}) \longrightarrow S_{size(B)}}\text{ Lemma 39}}{!_q(A \multimap B) \otimes !_q!_p S_{size(A)} \longrightarrow S_{size(B)}}\text{ canonical map } !_qX \otimes !_qY \to !_q(X \otimes Y)}{!_{p\cdot q}(A \multimap B) \otimes !_{p\cdot q} S_{size(A)} \longrightarrow S_{size(B)}}\text{ dereliction, simplification}}{!_{p\cdot q}(A \multimap B) \longrightarrow (!_{p\cdot q} S_{size(A)} \multimap S_{size(B)})}\text{ transposition}$$

The assertion about the underlying function follows directly by construction. □

Lemma 43 (Correctness of Skewed Iteration)**.** *Let A, B, C and D be small data types, such that there exists a* BLL*-formula E satisfying $\mathcal{I}(A) = E[y/x]$ and $\mathcal{I}(B) = E[y+1/x]$ and $\mathcal{I}(C) = E[0/x]$ and $\mathcal{I}(D) = E$. Then there are polynomials p and q, for which the object*

$$\mathbf{N}^p_x \multimap !_q!_{y<x}(A \multimap B) \multimap !_q C \multimap D$$

has a global element, whose underlying function is the standard iteration combinator.

Proof. We use the skewed iteration principle from Lemma 41. By this lemma, we can assume the iteration principle

$$!_p!_{y<x}S_x \multimap !_q!_{y<x}(!_i S_{\vec{p}(y)} \multimap S_{\vec{p}(y+1)}) \multimap !_q S_{\vec{p}(0)} \multimap S_{\vec{p}(x)}.$$

Due to the assumptions on the types A, B, C and D, their sizes are such that we can assume the following iteration principle.

$$!_p!_{y<x}S_x \multimap !_q!_{y<x}(!_i S_{size(A)} \multimap S_{size(B)}) \multimap !_q S_{size(C)} \multimap S_{size(D)}.$$

Using Lemmas 42 and 39, we get from this

$$!_p!_{y<x}S_x \multimap !_q!_{y<x}!_p(A \multimap B) \multimap !_q!_r C \multimap S_{size(D)}$$

for suitable polynomials r and s. This, in turn, is equivalent to

$$!_p!_{y<x}S_x \otimes !_q!_{y<x}!_s(A \multimap B) \otimes !_q!_r C \multimap S_{size(D)},$$

and using Lemma 39 we get from this the iteration principle below.

$$!_t(!_p!_{y<x}S_x \otimes !_q!_{y<x}!_s(A \multimap B) \otimes !_q!_r C) \multimap D$$

With the canonical morphism of the form $!_t X \otimes !_t Y \to !_t(X \otimes Y)$, we get from this

$$!_t!_p!_{y<x}S_x \multimap !_t!_q!_{y<x}!_s(A \multimap B) \multimap !_s!_q!_r C \multimap D.$$

Using the decoding morphism from Lemma 38, this becomes

$$!_t!_p!_{y<x}N^u_x \multimap !_t!_q!_{y<x}!_s(A \multimap B) \multimap !_s!_q!_r C \multimap D.$$

for a suitable polynomial u. Using the coercion on $\mathbf{N}$ and by combining the modalities, we obtain the required iteration principle. □

3.7 Complexity

Lemma 44. *Let $p \in P_1(X)$, $q \in P(X)$ and $r \in P_1(X)$ be resource polynomials satisfying $p, r \geq \lambda x.x$ and $q \geq 3$. Then there exist in $U(X \cup \{x\})$ morphisms $\mathbf{W}^{p,q,r}_x \to (!_{3x\cdot p(x)}S_x \multimap S_3) \otimes S_x$ and $\mathbf{W}^{p,q,r}_x \to !_q(S_x \multimap S_3) \otimes !_{3x\cdot r(x)}S_x$, whose underlying functions are an isomorphism.*

Proof. Using Lemmas 37 and 38 and the fact that for $u \leq v$ one can easily give a proof $\mathbf{N}^u_x \multimap \mathbf{N}^v_x$ that has the identity as its underlying function. □

Proof of Theorem 3. By Proposition 24 and Lemma 44. □

Proof of Theorem 4. By Proposition 25 and Lemma 44. □

4 Completeness

Having shown LOGSPACE-soundness for SBAL, it remains to show LOGSPACE-completeness, that any LOGSPACE-computable function can be represented in SBAL (Theorem 5). We do this in this section by showing that each LOGSPACE Turing Machine can be encoded in SBAL.

We fix notation for off-line Turing Machines. Without loss of generality, we restrict our attention to machines with input and output alphabet $3 = \{*, 0, 1\}$, where $*$ is a blank symbol, and with a single work tape with alphabet $2 = \{0, 1\}$. A machine M is a triple (S, s_i, δ) where S is a finite set of states, $s_i \in S$ is the initial state and $\delta\colon S{\times}3{\times}2 \to S{\times}2{\times}\{\mathit{halt}, \mathit{left}_I, \mathit{right}_I, \mathit{left}_W, \mathit{right}_W, \mathit{output}\}$ is a transition function. The input to δ consists of the current state, the character read by the input head and the character read by the work head. Its output consists of a new state, a character that is written by the work head and a move instruction. The instructions left_I and right_I represent moves of the input head, left_W and right_W represent moves of the work head and *output* stands for the output of the current character on the work tape to the output tape.

For the rest of this section, we fix a LOGSPACE Turing Machine M. Let $p \in P(\{x\})$ be a polynomial with natural coefficients, such that on input $x \in \{0, 1\}^*$ the machine M halts in no more than $p(|x|)$ steps and writes no more than $\log(p(|x|))$ characters on its work tape.

A state of the machine M in a run with input w is a tuple $\langle s, t, i, o, \sigma\rangle \in S{\times}(\mathbb{N}_{p(x)}{\times}\mathbb{N}_{p(x)}){\times}\mathbb{N}_x{\times}\mathbb{N}_{p(x)}{\times}3$, where $x = |w|{+}1$ and where s is the state of M, t encodes the work tape and the position of the work head, i encodes the position of the input head, o is the position of the output head and σ is the character scanned by o. The work tape is represented by a pair $t = \langle t_l, t_r\rangle$ as follows. Suppose the content of the work tape is $w_0 \dots w_n$, where $w_i \in 2$ for all $i \leq n$, and the position of the work head is $k \leq n$. If we e then $t_l = \sum_{i=0}^{k} 2^i w_{k-i}$ and $t_r = \sum_{i=k+1}^{n} 2^{i-k-1} w_i$. For the value i that represents the position of the input head, we can make the assumption that it never exceeds x, i.e. the input head never moves beyond the first blank symbol on the input tape. The transition function mapping states to states is defined from δ in the obvious way.

To encode the Turing Machine M in SBAL, we represent its state by the following formulae

$$\begin{aligned}
\mathbf{Head}_x^q &\stackrel{\text{def}}{=} \mathbf{N}_x^q \otimes \mathbf{N}_x^q,\\
\mathbf{Tape}_x^q &\stackrel{\text{def}}{=} \mathbf{N}_x^q \otimes \mathbf{N}_x^q \otimes \mathbf{N}_x^q,\\
\mathbf{State}_p^q &\stackrel{\text{def}}{=} \mathbf{F}_l^{q(0)} \otimes \mathbf{Tape}_{p(x)}^q \otimes \mathbf{Head}_x^q \otimes \mathbf{Head}_{p(x)}^q \otimes \mathbf{F}_k^{q(0)},
\end{aligned}$$

where q is a resource polynomials in $P_1(\{x\})$. The formula **Head** is used to represent the positions of the input and output head. The intended meaning of a pair $\langle m, k\rangle$ in **Head** is that k is the position of the head on the tape and m is an upper bound on the length of the tape. We need to include a bound on the length of the tape in order to be able to give the movement-operations of heads the type $\mathbf{Head}_x \multimap \mathbf{Head}_x$. If one represents the head position by the number $k \in \mathbf{N}_x^q$ only, then the right-shift-operation can only be given type $\mathbf{N}_x \multimap \mathbf{N}_{x+1}$ and not type $\mathbf{N}_x \multimap \mathbf{N}_x$. The formula **Tape** for the representation of the work-tape also has an additional component, which we use to represent a bound on the length of the tape. Thus, the content of a work tape is represented by a triple $\langle p(m), t_l, t_r\rangle$, where t_l and t_r are as described above and m is an upper bound on the length of the input of M.

Lemma 45. *For each polynomial $p \in P(X)$, there exist polynomials $q, r \in P(X)$, such that the addition, subtraction and multiplication functions are de-*

finable with types

$$\vdash \mathit{plus} \colon \mathbf{N}^p_x \multimap \mathbf{N}^q_y \multimap \mathbf{N}^p_{x+y},$$
$$\vdash \mathit{minus} \colon \mathbf{N}^r_x \multimap \mathbf{N}^q_y \multimap \mathbf{N}^p_x,$$
$$\vdash \mathit{times} \colon \mathbf{N}^r_x \multimap \mathbf{N}^q_y \multimap \mathbf{N}^p_{x \cdot y}.$$

Proof. The definition of addition appears in Section 2.1.1. Multiplication is easily defined by iterating addition. For subtraction, one can first define the predecessor function by the following iteration.

$$\begin{aligned}
&\mathit{base} = \langle \mathit{zero}, \mathit{true} \rangle \\
&\mathit{step}(\langle n, \mathit{true} \rangle) = \langle \mathit{zero}, \mathit{false} \rangle \\
&\mathit{step}(\langle n, \mathit{false} \rangle) = \langle \mathit{succ}(n), \mathit{false} \rangle \\
&\mathit{pred}'(\mathit{zero}) = \mathit{base} \\
&\mathit{pred}'(\mathit{succ}(n)) = \mathit{step}(\mathit{pred}'\ n) \\
&\mathit{pred}(n) = \pi_1(\mathit{pred}'(n))
\end{aligned}$$

It is clear that this works using skewed iteration, but it is also possible to define this iteration just by normal iteration. However, to define subtraction by iteration of the predecessor function, skewed iteration appears to be essential. □

Lemma 46. *For each polynomial $p \in P(X)$, there exist polynomials $q, r \in P(X)$ and derivable sequents*

$$\vdash \mathit{isnull} \colon \mathbf{N}^q_x \multimap \mathbf{B}^p, \qquad \vdash \mathit{half} \colon \mathbf{N}^q_x \multimap \mathbf{N}^r_y \multimap \mathbf{N}^p_x,$$
$$\vdash \mathit{eq} \colon \mathbf{N}^q_x \multimap \mathbf{N}^r_y \multimap \mathbf{B}^p, \qquad \vdash \mathit{min} \colon \mathbf{N}^q_x \multimap \mathbf{N}^r_y \multimap \mathbf{N}^p_y,$$
$$\vdash \mathit{lt} \colon \mathbf{N}^q_x \multimap \mathbf{N}^r_y \multimap \mathbf{B}^p,$$

whose underlying functions satisfy the equations below.

$$\begin{aligned}
\mathit{isnull}(\mathit{zero}) &= \top & \mathit{lt}(x, \mathit{zero}) &= \bot \\
\mathit{isnull}(\mathit{succ}(n)) &= \bot & \mathit{lt}(x, \mathit{succ}(y)) &= \mathit{or}(\mathit{lt}(x, y), \mathit{eq}(x, y))
\end{aligned}$$

$$\mathit{eq}(x, y) = \mathit{and}(\mathit{isnull}(\mathit{minus}(x, y)), \mathit{isnull}(\mathit{minus}(y, x)))$$

$$\mathit{half}(\mathit{zero}) = \mathit{zero}$$

$$\mathit{half}(\mathit{succ}(y)) = \begin{cases} \mathit{succ}(\mathit{half}(y)) & \text{if } \mathit{lt}(\mathit{double}(\mathit{half}(y)), y) = \top \\ \mathit{half}(y) & \text{otherwise} \end{cases}$$

$$\mathit{min}(\mathit{zero}) = \mathit{zero}$$

$$\mathit{min}(\mathit{succ}(y)) = \begin{cases} \mathit{succ}(\mathit{ceil}(y)) & \text{if } \mathit{lt}(\mathit{min}(y), y) = \top \\ \mathit{min}(y) & \text{otherwise} \end{cases}$$

Proof. Using skewed iteration, the functions can evidently be defined by the iterative definitions indicated by the underlying functions. □

Operations on heads and tapes are given by the next two lemmas. which are proved by straightforward combination of the functions from Lemmas 45 and 46.

Lemma 47. *For all $p \in P_1(\{x\})$, there exists a polynomial $q \in P_1(\{x\})$ with $q \geq p$, such that there are derivable sequents*

$$\vdash \mathit{left} \colon \mathbf{Head}_x^q \multimap \mathbf{Head}_x^p,$$
$$\vdash \mathit{right} \colon \mathbf{Head}_x^q \multimap \mathbf{Head}_x^p,$$

whose underlying functions satisfy $\mathit{left}(m,i) = \langle m, \mathit{pred}(i)\rangle$ and $\mathit{right}(m,j) = \langle m, \min(m, j+1)\rangle$.

Lemma 48. *For all $p \in P_1(\{x\})$, there exists a polynomial $q \in P_1(\{x\})$ with $q \geq p$, such that there are derivable sequents*

$$\vdash \mathit{left} \colon \mathbf{Tape}_x^q \multimap \mathbf{Tape}_x^p,$$
$$\vdash \mathit{right} \colon \mathbf{Tape}_x^q \multimap \mathbf{Tape}_x^p,$$
$$\vdash \mathit{read} \colon \mathbf{Tape}_x^q \multimap \mathbf{B}^{p(0)},$$
$$\vdash \mathit{write} \colon \mathbf{B}^{q(0)} \multimap \mathbf{Tape}_x^q \multimap \mathbf{Tape}_x^p,$$

whose underlying functions satisfy

$$\begin{aligned}
\mathit{left}(m,l,r) &= \langle m, l/2, \min(m, 2r + (l \bmod 2))\rangle,\\
\mathit{right}(m,l,r) &= \langle m, \min(m, 2l + (r \bmod 2)), r/2\rangle,\\
\mathit{read}(m,l,r) &= (l \bmod 2),\\
\mathit{write}(\langle m,l,r\rangle, b) &= \langle m, \min(m, l/2 + b), r\rangle.
\end{aligned}$$

Lemma 49. *For all $s \in S$, $x \in 3$, $w \in 2$ and all $p, q \in P_1(\{x\})$, there exists a polynomial $r \in P_1(\{x\})$ and a derivable sequent*

$$\mathbf{Tape}_{p(x)}^r, \mathbf{Head}_x^r, \mathbf{Head}_{p(x)}^r, \mathbf{F}_3^{r(0)} \vdash \mathbf{State}_p^q,$$

whose underlying function f has the following property. If the Turing Machine M is in state $\langle s,t,i,o,\sigma\rangle$, if its input head scans the character x and its work-head scans the character w, then f applied to t, i and o, returns the state that M assumes next after $\langle s,t,i,o,\sigma\rangle$.

Proof. The state M assumes after $\langle s,t,i,o,\sigma\rangle$, can be arrived at by moving its input, output and work heads and updating the symbol σ. This can be done using the operations on heads and tapes from Lemmas 47 and 48. □

Lemma 50. *For all $p, q \in P_1(\{x\})$, there exist polynomials u, v, w and r such that the transition function of M can be derived as a sequent of the following type.*

$$!_{12k}\mathbf{W}_x^{u,v,w} \vdash \mathbf{State}_p^r \multimap \mathbf{State}_p^q$$

Proof. We must find resource polynomials u, v, w and r, using which the transition function can be implemented as asserted. We will derive these polynomials in the course of the construction of the derivation.

Besides the polynomials u, v, w and r, we will use two polynomials $r_1, r_2 \in P_1(\{x\})$ that satisfy $r \geq r_1 \geq r_2 \geq u$. In the following construction, the polynomials will be determined step by step. The dependencies between the polynomials are: $u = u(p,q)$, $v = v(p,q)$, $r_2 = r_2(u,v)$, $r_1 = r_1(r_2)$ and $r = r(r_1)$, i.e. u depends on p and q and so on.

To derive the transition function, we can assume elements of $!_{12k}\mathbf{W}_x^{u,v,w}$ and $\mathbf{State}_p^r$ and must define an appropriate element of $\mathbf{State}_p^q$. First, using contraction and projection, the element in $!_{12k}\mathbf{W}_x^{u,v,w}$ gives us two elements

$$f\colon !_{6k}(!_v\mathbf{N}_x^u \multimap \mathbf{F}_3^v), \quad b\colon !_{6k}\mathbf{N}_x^w,$$

where f represents the bits of the input and b is an upper bound on the length of the input.

Next, since for **State** we can define a coercion like that for **N** from Section 2.1.1, we derive from $\mathbf{State}_p^r$ an element $!_5!_{12k}\mathbf{State}_p^{r_1}$. Note that r is determined from r_1 by the coercion. By contraction and dereliction, we obtain elements

$$s\colon \mathbf{F}_k^{r_1(0)},\ t\colon !_{12k}\mathbf{Tape}_{p(x)}^{r_1},\ h_i\colon !_{12k}\mathbf{Head}_x^{r_1},\ h_o\colon !_{6k}\mathbf{Head}_{p(x)}^{r_1},\ o\colon !_{6k}\mathbf{F}_3^{r_1(0)}.$$

Of these elements, s represents the state of the Turing Machine, t represents its work tape and work head, h_i and h_o represent the input and output head, and o represents the symbol that was output last. Using contraction and the read function from Lemma 46, we get from t two elements

$$w\colon !_{6k}\mathbf{B}^{r_2(0)},\ t'\colon !_{6k}\mathbf{Tape}_{p(x)}^{r_1},$$

where r_1 with $r_1 \geq r_2$ is determined from r_2. From h_i, we obtain by contraction, projection and coercion the position i of the input head and a copy of h_i for further use:

$$i\colon !_{6k}\mathbf{N}_x^{r_1},\ h_i'\colon !_{6k}\mathbf{Head}_x^{r_1}.$$

By the coercion on **N** from Section 2.1.1, we obtain from i an element $i'\colon !_{6k}!_v\mathbf{N}_x^u$. By this construction, r_2 is determined from u and v and we can assume that $r_2 \geq u$ holds. Application of f to i' gives us an element $x\colon !_{6k}\mathbf{F}_3^v$. Since $r_1 \geq r_2 \geq u$ and because we can replace larger polynomials by smaller ones, we can replace the bound r_1 by u in all elements. With further dereliction, the construction up to now therefore gives us elements

$$s\colon \mathbf{F}_k^{u(0)},\ w\colon !_k\mathbf{B}^{u(0)},\ x\colon !_{2k}\mathbf{F}_3^v,$$

$$t'\colon !_{6k}\mathbf{Tape}_{p(x)}^u,\ h_i'\colon !_{6k}\mathbf{Head}_x^u,\ h_o\colon !_{6k}\mathbf{Head}_{p(x)}^u,\ o\colon !_{6k}\mathbf{F}_3^{u(0)}.$$

Now we proceed by case distinction, first on s, then on w and finally on x. This gives rise to $6k$ cases, where in each case we must define an element of $\mathbf{State}_p^q$ from

$$t''\colon \mathbf{Tape}_{p(x)}^u,\ h_i''\colon \mathbf{Head}_x^u,\ h_o'\colon \mathbf{Head}_{p(x)}^u,\ o'\colon \mathbf{F}_3^{u(0)}.$$

For this we use Lemma 49, thus completing the construction. □

Lemma 51. *For each polynomial $p \in P_1(\{x\})$ with natural coefficients and each $q \in P_1(\{x\})$ there exists a $r \in P_1(\{x\})$ and a derivable sequent $\vdash \mathbf{N}_x^r \multimap \mathbf{N}_{p(x)}^q$, whose underlying function is the mapping $x \mapsto p(x)$.*

Proof. Constants, addition and multiplication are definable. □

Lemma 52. *For all $p, q \in P_1(\{x\})$, there exists a $r \in P_1(\{x\})$ and a derivable sequent*

$$\mathbf{N}_x^r \vdash \mathbf{State}_p^q$$

whose underlying function maps any upper bound on the length of the input to M to a code for the initial state for M.

Proof of Theorem 5. We use Lemma 50. Using the transition function defined in that lemma, it is straightforward to show that there exist u, v, w, r and s, such that the following function tr' is represented by a derivation.

$$tr' \colon \mathbf{W}_x^{u,v,w} \multimap \mathbf{N}_{p(x)}^s \multimap \mathbf{State}_p^r \multimap \mathbf{State}_p^q$$

$$tr'(w, n, \langle q, t, i, o, \sigma\rangle) = \begin{cases} tr(w, \langle q, t, i, o, \sigma\rangle) & \text{if } o < n, \\ \langle q, t, i, o, \sigma\rangle & \text{otherwise.} \end{cases}$$

Since $\mathbf{State}_p^q$ is a small data type, we can use the skewed iteration principle to iterate it. By Lemma 51, we can compute an upper bound on the computation length on M. We use this number to iterate the step function tr' as often. Moreover, by Lemma 52 the initial state of M can be encoded. Putting these facts together, we obtain that for each q there is some t and a derivable term

$$!_t\mathbf{W}_x^{u,v,w},\ !_t\mathbf{N}_{p(x)}^s \vdash c \colon \mathbf{State}_p^q$$

such that $c(\langle f_w, l\rangle, n)$ is the $p(l)$-fold iteration of the function $tr'(\langle f_w, l\rangle, n)$ on the initial state. From $c(w, n)$, the nth bit of the output of M can be extracted directly. If $c(w, n) = \langle \_, \_, \_, o, \sigma\rangle$ and $o = n$ then the nth output bit is σ, otherwise it must be the blank symbol. Hence, for each q, we can find a t and a derivable term

$$!_t\mathbf{W}_x^{u,v,w},\ !_t\mathbf{N}_{p(x)}^s \vdash c' \colon \mathbf{F}_3^q$$

such that $c'(w, n)$ is the nth bit of the output. Using dereliction and the fact that an upper bound for the length of the output of M can be computed easily, we can from c' easily construct a term

$$!_t\mathbf{W}_x^{u,v,w} \vdash \mathbf{W}_{p(x)}^{u',v',w'}$$

that represents the function computed by M. □

5 Soundness for Impredicatively Coded Words

A LOGSPACE-soundness result can also be formulated with $\mathbf{S}_x$ (impredicatively encoded words) instead of $\mathbf{W}_x$ (words represented by functions). To show this we need to strengthen the iteration principled embodied by $\mathbf{N}_x$ and $\mathbf{S}_x$ to the recursion principles

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

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

in which we write B for $(\mathbf{N}_{y+w}^v \multimap \alpha(y) \multimap \alpha(y+1))$, where q is a suitable polynomial depending on p, and where v and w are arbitrary polynomials. 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, they can be interpreted in the above semantics, as we will now show.

Lemma 53 (Recursion principle on S_x)**.** *Let A in a p-encodable object in $U(X \cup \{x\})$ and let $v, w \in P(X \cup \{y\})$. Then the object*

$$!_3!_p!_x S_x \multimap !_{y<x}(!_v S_{y+w} \multimap A[y/x] \multimap A[y+1/x]) \multimap A[0/x] \multimap A$$

has a global element whose underlying function rec satisfies

$$rec(zero, f, g) = g,$$
$$rec(succ(n), f, g) = f(n, rec(n, f, g)).$$

Proof. A realiser for *rec* is defined almost exactly as in the proof of Lemma 23. The domain of the realiser r_η from Lemma 23 gets a new summand

$$\sum_{m<\eta(x)} \mathbb{N}_{v[\eta]} \times \|S_{y+w}\|^-_{\eta[m/x]}.$$

Its range gets a new summand

$$\sum_{m<\eta(x)} \mathbb{N}_{v[\eta]} \times \|S_{y+w}\|^+_{\eta[m/x]}.$$

The realiser r_η from Lemma 23 is now extended such that whenever it receives a question $\langle m, k, * \rangle$ in the new summand in the domain, it outputs the answer $\langle m, k, m \rangle$ in the new summand in its range. Informally, whenever r_η is being asked for the number in $!_v S_{y+w}$, it answers with the value that is stored in the memory cell of the modality $!_{y<x}$. □

This lemma suffices for modelling (RecN), since for each p there exists a q and a morphism $\mathbf{N}^q_x \to !_p S_x$ having the identity as the underlying function. This morphism can be derived using Lemma 38 and the coercion from Section 2.1.

To justify the recursion principle on $\mathbf{S}_x$, we use now introduce an object T_x in $U(X)$, which relates to $\mathbf{S}_x$ in the same way S_x relates to $\mathbf{N}_x$.

$$|T_x| = \{0,1\}^*$$
$$\|T_x\|_\eta = (\mathbb{N}_{\eta(x)}, \{0,1,*\})$$
$$\eta, e \Vdash_{T_x} b_1 \dots b_n \iff n \leq \eta(x) \ \wedge \ e(i) = \begin{cases} b_{i+1} & \text{if } 0 \leq i < n \\ * & \text{otherwise} \end{cases}$$

The realisation relation on T_x is such that with a single question, we can obtain a single character of the word.

Lemma 54. *There are morphisms empty*$\colon I \to T_x$*, cons*$_0\colon T_x \to T_{x+1}$ *and cons*$_1\colon T_x \to T_{x+1}$ *in* $U(\{x\})$.

Proof. A realiser for *empty* is the constant function returning $*$. A realiser r_η for $cons_i$ has type $\{0,1,*\} + \mathbb{N}_{\eta(x+1)} \to \mathbb{N}_{\eta(x+1)} + \{0,1,*\}$. It can be defined by

$$r_\eta(inl(a)) = inr(a)$$
$$r_\eta(inr(q)) = \begin{cases} inl(q-1) & \text{if } q > 0 \\ inr(i) & \text{otherwise} \end{cases}$$

□

Lemma 55. *There exists a global element of* $\mathbf{S}^{\lambda x.x+3}_x \multimap T_x$.

Proof. A size polynomial for T_x is given by $x + 3$. Hence, the required element can be defined by iteration of the zero and successor functions from the previous lemma. □

Lemma 56 (Recursion principle on S_x)**.** *Let A in a p-encodable object in $U(X \cup \{x\})$ and let $v, w \in P(X \cup \{y\})$. Let B denote the object $!_{y<x}(!_v S_{y+w} \multimap A[y/x] \multimap A[y+1/x])$ Then there exists a polynomial $q \in P(X \cup \{x\})$ (depending on p), such the object*

$$!_3 !_p !_x T_x \multimap B \multimap B \multimap A[0/x] \multimap A$$

has a global element whose underlying function rec satisfies

$$\begin{aligned} rec(empty, f_0, f_1, g) &= g, \\ rec(cons_0(n), f_0, f_1, g) &= f_0(n, rec(n, f, g)), \\ rec(cons_1(n), f_0, f_1, g) &= f_1(n, rec(n, f, g)). \end{aligned}$$

The proof is similar to that of Lemma 53.

Using Lemmas 55 and 56, we can justify the recursion principle (RecS).

5.1 From Words to Functions and Back

We now show that with the recursion principles (RecN) and (RecS) from the previous section, we can go back and forth between the types **W** and **S**.

Lemma 57. *There exists polynomials u, v, w and p such that $\vdash \mathbf{S}_x^p \multimap \mathbf{W}_{x+1}^{u,v,w}$ has a proof in* SBAL+(RecS)*, whose underlying function maps any binary word b to a pair $\langle f, n\rangle$ representing that word in the sense of Definition 4.*

Proof. It suffices to derive both $\vdash \mathbf{S}_x^q \multimap \mathbf{N}_{x+1}^w$ and $\vdash \mathbf{S}_x^p \multimap (!_v \mathbf{N}_{x+1}^u \multimap \mathbf{F}_3^v)$ with the underlying function of the first derivation computing an upper bound on the length of the input word and the underlying function of the second derivation yielding a function that computes the bits of the input word. This is the case because $\mathbf{W}_{x+1}^{u,v,w}$ is defined as $(!_v \mathbf{N}_{x+1}^u \multimap \mathbf{F}_3^v) \otimes \mathbf{N}_{x+1}^w$, because we have rule (Contraction), and because we can derive coercions $\mathbf{S}_x^{p+q} \multimap \mathbf{S}_x^p$ and $\mathbf{S}_x^{p+q} \multimap \mathbf{S}_x^p$.

For the derivation of the first sequent, we note that there exists a polynomial q and a derivation of $\vdash \mathbf{S}_x^q \multimap \mathbf{N}_{x+1}^w$, whose underlying function maps a word b to $length(b) + 1$. To see this, assume $\mathbf{S}_x$ and instantiate the universal quantifier in it with $\alpha(x) = \mathbf{N}_{x+1}^w$. The resulting formula is

$$!_{y<x}(\mathbf{N}_{y+1}^w \multimap \mathbf{N}_{y+2}^w) \multimap !_{y<x}(\mathbf{N}_{y+1}^w \multimap \mathbf{N}_{y+2}^w) \multimap \mathbf{N}_1^w \multimap \mathbf{N}_{x+1}^w.$$

Applying this function to the successor function and the constant 'one', this gives the required result in $\mathbf{N}_{x+1}^w$.

Next we derive $\vdash \mathbf{S}_x^p \multimap (!_v \mathbf{N}_{x+1}^u \multimap \mathbf{F}_3^v)$ for some $v \geq 3$ and $u, w \geq \lambda x.\, x$. We take $v = 2x + 3$. The underlying function of the derivation will be

$$\begin{aligned} \lambda w.\, \lambda n.\, rec\ n \quad & (\lambda b, x.\, (\text{if } b \text{ then } 0 \text{ else } x)) \\ & (\lambda b, x.\, (\text{if } b \text{ then } 1 \text{ else } x)) \\ & *, \end{aligned}$$

where *rec* is the underlying function of the recursion principle for $\mathbf{S}_x$. Also, recall that we think of $\mathbf{F}_3^v$ as the three-element set $\{0, 1, *\}$). To derive $\vdash \mathbf{S}_x^p \multimap (!_v \mathbf{N}_{x+1}^u \multimap \mathbf{F}_3^v)$, we start by using ($\multimap$-R) and (RecS). This step determines p and reduces the goal to

$$!_{y<x}(\mathbf{N}_x^r \multimap \mathbf{F}_3^v \multimap \mathbf{F}_3^v) \multimap !_{y<x}(\mathbf{N}_x^r \multimap \mathbf{F}_3^v \multimap \mathbf{F}_3^v) \multimap \mathbf{F}_3^v \multimap \mathbf{F}_3^v,\ !_v \mathbf{N}_{x+1}^u \vdash \mathbf{F}_3^v.$$

Using contraction and dereliction, we can replace $!_v \mathbf{N}_{x+1}^u$ by $!_x \mathbf{N}_{x+1}^u, !_x \mathbf{N}_{x+1}^u$. By ($\forall$-L), the goal therefore reduces to three sub-goals:

$$!_x \mathbf{N}_{x+1}^u \vdash !_{y<x}(\mathbf{N}_x^r \multimap \mathbf{F}_3^v \multimap \mathbf{F}_3^v) \tag{2}$$

$$!_x \mathbf{N}_{x+1}^u \vdash !_{y<x}(\mathbf{N}_x^r \multimap \mathbf{F}_3^v \multimap \mathbf{F}_3^v) \tag{3}$$

$$\vdash \mathbf{F}_3^v \tag{4}$$

For the last sequent we take the proof that represents $*$. For the other two sequents, we continue the derivation as follows. Using rule (Functoriality), it suffices to derive

$$\mathbf{N}_{x+1}^u \vdash \mathbf{N}_x^r \multimap \mathbf{F}_3^v \multimap \mathbf{F}_3^v$$

Using ($\multimap$-L) twice, this reduces to

$$\mathbf{N}_{x+1}^u, \mathbf{N}_x^r, \mathbf{F}_3^v \vdash \mathbf{F}_3^v$$

Let s be the size polynomial of $\mathbf{F}_3^v$. By Lemma 46, there exists polynomials u' and r' such that the equality test has type $\mathbf{N}_{x+1}^{u'} \multimap \mathbf{N}_x^{r'} \multimap \mathbf{B}^s$. Since, as described in Section 2.1 we can increase the polynomials on the left, we may assume $u', r' \geq \lambda x.x$. Hence, we choose u and v to be u' and r' respectively. The goal of the derivation reduces to

$$\mathbf{B}^s, \mathbf{F}_3^v \vdash \mathbf{F}_3^v$$

By the choice of s, we can instantiate the universal quantifier in $\mathbf{B}^s$ with $\mathbf{F}_3^v$. Hence, using the evident applications, we can continue the proof such that it has the underlying function $\lambda b, x.$ (if b then 0 else x). This is what we choose to complete the derivation of sequent (2). For sequent (3), we instead take a proof with the underlying function $\lambda b, x.$ (if b then 1 else x). □

Lemma 58. *For all polynomials p, there exist polynomials u, v, w and z, such that $\vdash !_z \mathbf{W}_x^{u,v,w} \multimap \mathbf{S}_x^p$ has a proof in* SBAL+(RecN)*, whose underlying function maps a any pair $\langle f, n\rangle$ that represents a word in the sense of Definition 4 to the word it represents.*

Proof. We give a derivation of $\vdash !_z \mathbf{W}_x^{u,v,w} \multimap \mathbf{S}_x^p$ with the underlying proof given by

$$\lambda\langle f, n\rangle.\ rec_{\mathbf{N}}\ n\ (\lambda m\ w.\ case\ (f\ m)\ of\ 0 \mapsto succ_0(w) | 1 \mapsto succ_1(w) | * \mapsto w)\ empty,$$

where $rec_{\mathbf{N}}$ is the underlying function of the recursion principle and *empty* is the underlying term of the empty word in $\mathbf{S}_0^p$.

We start the derivation as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!_v\mathbf{N}^u_x \vdash !_v\mathbf{N}^u_x} \quad \dfrac{(1)}{\mathbf{F}^v_3 \vdash \mathbf{S}^p_y \multimap \mathbf{S}^p_{y+1}}}{!_v\mathbf{N}^u_x \multimap \mathbf{F}^v_3,\ !_v\mathbf{N}^u_x \vdash \mathbf{S}^p_y \multimap \mathbf{S}^p_{y+1}}}{!_v\mathbf{N}^u_x \multimap \mathbf{F}^v_3 \vdash !_v\mathbf{N}^u_x \multimap \mathbf{S}^p_y \multimap \mathbf{S}^p_{y+1}}}{!_{y<x}(!_v\mathbf{N}^u_x \multimap \mathbf{F}^v_3) \vdash !_{y<x}(!_v\mathbf{N}^u_x \multimap \mathbf{S}^p_y \multimap \mathbf{S}^p_{y+1})} \quad \dfrac{\dfrac{\text{empty}}{\vdash \mathbf{S}^p_0} \quad \dfrac{}{\mathbf{S}^p_x \vdash \mathbf{S}^p_x}}{\mathbf{S}^p_0 \multimap \mathbf{S}^p_x \vdash \mathbf{S}^p_x}}{\mathbf{S}^p_x \leq w \qquad !_{y<x}(!_v\mathbf{N}^u_x \multimap \mathbf{F}^v_3),\ !_{y<x}(!_v\mathbf{N}^u_x \multimap \mathbf{S}^p_y \multimap \mathbf{S}^p_{y+1}) \multimap \mathbf{S}^p_0 \multimap \mathbf{S}^p_x \vdash \mathbf{S}^p_x}$$
$$\dfrac{!_{y<x}(!_v\mathbf{N}^u_x \multimap \mathbf{F}^v_3),\ \mathbf{N}^w_x \vdash \mathbf{S}^p_x}{\vdash !_{y<x}(!_v\mathbf{N}^u_x \multimap \mathbf{F}^v_3) \otimes \mathbf{N}^w_x \multimap \mathbf{S}^p_x}$$

At leaf (1) we continue with a case distinction on the over $\mathbf{F}^v_3$. Thus, for v we can choose any polynomial satisfying $\mathbf{S}^p_y \multimap \mathbf{S}^p_{y+1} \leq v$. We use the notation $\{0, 1, *\}$ for the underlying set of $\mathbf{F}^v_3$. The case distinction in (1) such that if the element of $\mathbf{F}^v_3$ is 0 then we choose the first successor function for $\mathbf{S}^p_y \multimap \mathbf{S}^p_{y+1}$, if the element is 1 then we choose the second successor function, and if the element is $*$ then we choose the identity function.

Notice that we can choose the polynomial u, v and w so that they satisfy the bounds in the definition of $\mathbf{W}^{u,v,w}_x$. □

Using the previous two lemmas, it is now possible to derive from Theorems 4 a statement to effect that the underlying function $f\colon \{0,1\}^* \to \{0,1\}^*$ of each derivation $\vdash \mathbf{S}^r_x \multimap \mathbf{S}^q_{p(x)}$ for sufficiently large q is computable in logarithmic space.

We leave it for further work whether completeness also holds for sequents of the form $\vdash \mathbf{S}^r_x \multimap \mathbf{S}^q_{p(x)}$.

6 Conclusion and Further Work

We believe that Stratified Bounded Affine Logic improves on existing type systems for logarithmic space, such as [Møller-Neergaard, 2004, Møller-Neegaard, 2004, Kristiansen, 2005, Schöpp, 2006]. 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 [Schöpp, 2006], but improves on [Schöpp, 2006] in simplicity and generality.

For further work, it should be interesting to study the representation of structured data in SBAL. In this paper we have only considered strings in Σ^*, which we have encoded as pairs $(\mathbb{N} \to \Sigma) \times \mathbb{N}$. It should be possible to represent other data types in this style. Graphs, for example, could be represented by pairs $(\mathbb{N}\times\mathbb{N} \to 2)\times\mathbb{N}$, where $\langle f, n\rangle$ represents the graph (V, E) with $V = \{0, \ldots, n-1\}$ and $E(x, y) \iff f(x, y) = 1$. It should be investigated whether using this encoding LOGSPACE graph algorithms can be represented in SBAL in a natural way.

Another direction for further work is to look for simplifications of the logic. For example, one may aim for a logic that does not contain resource polynomials. Perhaps existing polynomial-time logics, such as Light Linear Logic [Girard, 1998], can serve as a starting point in this direction.

We also believe that it should be interesting to study SBAL-based type systems for functional programming. To be able to use a type system based on

SBAL in practice, questions such as type checking and type inference should be settled.

Finally, it would be interesting to ascertain whether the ideas in the present paper can be used to capture other complexity classes apart from LOGSPACE.

References

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

[Abramsky and Jagadeesan, 1992] Abramsky, S. and Jagadeesan, R. (1992). Games and full completeness for multiplicative linear logic (extended abstract). In *FSTTCS92*, pages 291–301.

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

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

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

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

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

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

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

[Jacobs, 1999] Jacobs, B. (1999). *Categorical Logic and Type Theory*. Elsevier Science.

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

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

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

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

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

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