# Typed intermediate languages for shape-analysis

G. Bellè and E. Moggi
{gbelle,moggi}@disi.unige.it
DISI, Univ. di Genova, v. Dodecaneso 35, 16146 Genova, Italy
phone: +39-10-3536629, fax: +39-10-3536699

**Abstract**

We introduce $S2$, a typed intermediate language for vectors, based on a 2-level type-theory, which distinguishes between compile-time and run-time. The paper shows how $S2$ can be used to extract useful information from programs written in the Nested Sequence Calculus $\mathcal{NSC}$, an idealized high-level parallel calculus for nested sequences. We study two translations from $\mathcal{NSC}$ to $S2$. The most interesting shows that shape analysis (in the sense of Jay) can be handled at compile-time.

## Introduction

Good intermediate languages are an important prerequisite for program analysis and optimization, the main purpose of such languages is to make as explicit as possible the information that is only implicit in source programs (see [18]). A common features of such intermediate languages is an *aggressive* use of types to incorporate additional information, e.g.: binding times (see [18]), boxed/unboxed values (see [20]), effects (see [23]). In particular, among the ML community the use of types in intermediate languages has been advocated for the TIL compiler (see [9]) and for region inference (see [24, 1]).

In areas like parallel programming, where efficiency is a paramount issue, good intermediate languages are even more critical to bridge the gap between high-level languages (e.g. NESL) and efficient implementations on a variety of architectures (see [2, 3, 22]). However, in this area of computing intermediate languages (e.g. VCODE) have not made significant use of types, yet.

This paper proposes a typed intermediate language $S2$ for vector languages, and shows how it may be used to extract useful information from programs written in the Nested Sequence Calculus $\mathcal{NSC}$ (see [22]), an idealized vector language very closed to NESL. For an efficient compilation of $\mathcal{NSC}$ (and similar languages) on parallel machines it is very important to know in advance the size of vectors (more generally the shape of data structures). We study two translations from the $\mathcal{NSC}$ to $S2$. The most interesting one separates what can be computed at compile-time from what must be computed at run-time, in particular array bound-checking can be done at compile-time (provided the while-loop of $\mathcal{NSC}$ is replaced by a for-loop).

Section 1 introduces the two-level calculus $S2$ and outlines its categorical semantics. Section 2 summarizes the high-level language $\mathcal{NSC}$, outlines two translations from $\mathcal{NSC}$ in $S2$ and the main results about them. Sections 3 and 4 give the syntactic details of the translations. Appendix A gives a formal description of $S2$ and defines auxiliary notation and notational conventions used in the paper.

**Related work.** The language $S2$ borrows the idea of built-in phase distinction from $HML$ (see [17]), and few inductive types at compile-time from type theory (e.g. see [19, 5]). There are also analogies with work on partial evaluation, in particular 2-level lambda calculi for binding time analysis (see [18, 6]). None of these calculi make use of dependent types. Also [16] deals with shape checking of array programs, but without going through an intermediate language.

The translation of $\mathcal{NSC}$ into $S2$ has several analogies with that considered in [8] to give a type-theoretic account of higher-order modules, and has been strongly influenced by ideas from shape theory and shape analysis (see [14, 15]). There are also analogies with techniques for constant propagation, but these technique tend to cope with languages (e.g. Pascal) where constant expressions are much simpler than compile-time expressions in $S2$.

This paper uses categorical semantics as a high-level language for describing what is happening at the syntactic level. A systematic link between type theories and categorical structures is given in [12].

# 1 The 2-level calculus $S2$ and its semantics

This section gives a compact description of $S2$ as a Type System a la Jacobs (see [12]), and summarizes its type-constructors. For a more detailed description of $S2$ we refer to appendix A.

- Sorts: $c$ and $r$, $c$ classifies compile-time types and $r$ run-time types.
- Setting: $c < c$ and $c < r$, i.e. $c$- and $r$-types may depend on $c$-values, but they may not depend on $r$-values.
- Closure properties of $c$-types: dependent products $\Pi x{:}\, A.B$, unit 1, sums $A{+}B$, dependent sums $\Sigma x{:}\, A.B$, NNO $N$, finite cardinals $n{:}\, N$.
- Closure properties of $r$-types: exponentials $A \to B$, universal types $\forall x{:}\, A.B$, unit 1, sums $A + B$, products $A \times B$, weak existential types $\exists x{:}\, A.B$.

The setting of $S2$ is very close to that of $HML$ (see [12, 17, 11]).

**Proposition 1.1** *$S2$ satisfies the following properties:*

- *Context separation:* $\Gamma \vdash J$ *implies* $\Gamma_c, \Gamma_r \vdash J$, *where* $\Gamma_\alpha$ *is the sequence of declarations* $x{:}\, A{:}\, \alpha$ *of sort* $\alpha$ *in* $\Gamma$
- *Phase distinction:* $\Gamma \vdash J_c$ *implies* $\Gamma_c \vdash J_c$, *where* $J_c$ *is an assertion of sort* $c$ *(i.e. nothing,* $A{:}\, c$ *or* $M{:}\, A{:}\, c$*)*
- *No run-time dependent types:* $\Gamma \vdash A{:}\, r$ *implies* $\Gamma_c \vdash A{:}\, r$.

**Proof** They follow immediately from the restrictions imposed by the setting. ∎

In this section we briefly outline a semantics of $S2$. This is important not only as a complement to the formal description, but also to suggest possible improvements to $S2$ and discuss semantic properties of translations (which rely on features of models not captured by $S2$ without *extensionality*). In general, a categorical model for $S2$ is given by a fibration $\pi{:}\, \mathcal{C} \to \mathcal{B}$ with the following additional properties:

- the base category $\mathcal{B}$ is locally cartesian closed and **extensive** (see [4]), i.e. has sums and the functors $+{:}\, \mathcal{B}/I \times \mathcal{B}/J \to \mathcal{B}/(I + J)$ are equivalences, and has a natural number object (NNO);

- the fibration $\pi\colon \mathcal{C} \to \mathcal{B}$ is bicartesian closed and has $\forall$- and $\exists$-**quantification** along maps in the base (see [12, 21]), i.e. for any $f\colon J \to I$ in $\mathcal{B}$ the substitution functor $f^*\colon \mathcal{C}_I \to \mathcal{C}_J$ has left and right adjoints $\exists_f \vdash f^* \vdash \forall_f$ and they commute with substitution;
- all functors $\langle in_0^*, in_1^* \rangle\colon \mathcal{C}_{I+J} \to \mathcal{C}_I \times \mathcal{C}_J$ (and $!\colon \mathcal{C}_0 \to 1$) are equivalences.

**Remark 1.2** Extensivity of sums and the last property are essential to validate the elimination rules for $+$ over sorts ($+$-$E$-$\alpha$). In a locally cartesian closed base category it is possible to interpret also identity types (of sort $c$), and validate the rules for extensional equality. Finite cardinals and vectors are definable from the natural numbers, using the other properties of the base category. One could have derived that the fibration $\pi$ has sums from the other properties. In fact, $A_0 + A_1 = \exists_{[id,id]\colon I+I \to I} A$, where $A \in \mathcal{C}_{I+I}$ is such that $in_i^*(A) = A_i$.

There is a simple way to construct a model $\pi\colon \mathbf{Fam}(\mathcal{C}) \to \mathbf{Set}$ of $S2$ starting from any cartesian closed category $\mathcal{C}$ with small products and small sums (e.g. **Set** and **Cpo**) using the **Fam**-construction, where

- $\mathbf{Fam}(\mathcal{C})$ is the category whose objects are pairs $\langle I \in \mathbf{Set}, a \in \mathcal{C}^I \rangle$ and morphisms from $\langle I, a \rangle$ to $\langle J, b \rangle$ are pairs $\langle f\colon I \to J, g \in \mathcal{C}^I(a, f^*b) \rangle$, where $f^*b$ is the $I$-indexed family of objects s.t. $(f^*b)_i = b_{fi}$ for any $i \in I$. Identity and composition are defined in the obvious way;
- $\pi\colon \mathbf{Fam}(\mathcal{C}) \to \mathbf{Set}$ is the functor $\langle I, a \rangle \mapsto I$ forgetting the second component (this is the standard way of turning a category $\mathcal{C}$ into a fibration over **Set**). In particular, the fiber $\mathbf{Fam}(\mathcal{C})_I$ over $I$ is (up to isomorphism) $\mathcal{C}^I$, i.e. the product of $I$ copies of $\mathcal{C}$.

**Remark 1.3** The first property for a categorical model is clearly satisfied, since the base category is **Set** . The second follows from the assumptions about $\mathcal{C}$, since in $\mathcal{C}^I$ products, sums and exponentials are computed pointwise. The third is also immediate since $\mathcal{C}^{I+J}$ and $\mathcal{C}^I \times \mathcal{C}^J$ are isomorphic (and therefore equivalent).

The interpretation of judgements in $\pi\colon \mathbf{Fam}(\mathcal{C}) \to \mathbf{Set}$ is fairly simple to describe:

- $\Gamma \vdash$ is interpreted by an object $\langle I, a \rangle$ in $\mathbf{Fam}(\mathcal{C})$, namely $I \in \mathbf{Set}$ corresponds to $\Gamma_c$ and $a = \langle a_i | i \in I \rangle \in \mathcal{C}^I$ corresponds to $\Gamma_r$;
- $\Gamma \vdash A\colon c$ is interpreted by a family $\langle X_i | i \in I \rangle$ of sets;
- $\Gamma \vdash M\colon A\colon c$ is interpreted by a family $\langle x_i \in X_i | i \in I \rangle$ of elements;
- $\Gamma \vdash B\colon r$ is interpreted by a family $b = \langle b_i | i \in I \rangle$ of objects of $\mathcal{C}$;
- $\Gamma \vdash N\colon B\colon r$ is interpreted by a family $\langle f_i\colon a_i \to b_i | i \in I \rangle$ of morphisms in $\mathcal{C}$.

We summarize some properties valid in these models, which are particularly relevant in relation to the translations defined subsequently.

**Proposition 1.4 ($\forall\exists$-exchange)**
*Given* $n\colon N\colon c \quad i\colon n\colon c \vdash A(i)\colon c \quad i\colon n\colon c, x\colon A(i)\colon c \vdash B(i,x)\colon r$

$$n\colon N\colon c \vdash \exists f\colon (\Pi i\colon n.A(i)).\forall i\colon n.B(i, fi) \cong \forall i\colon n.\exists x\colon A(i).B(i,x)$$

*namely the* canonical map $\langle f, g \rangle \mapsto \Lambda i\colon n.\langle fi, gi \rangle$ *is an isomorphism.*

**Remark 1.5** This property can be proved formally in *extensional* $S2$ by induction on the NNO $N$. The key lemma is $\Pi i{:}\, sn.A(i) \cong A(0) \times (\Pi i{:}\, n.A(si))$ and similarly for $\forall i{:}\, sn{:}\, A(i)$. The property is the *internal* version of the following property (which can be proved in system $F$ with surjective pairing):

$$\exists \langle \overline{x} \rangle{:}\, (A_1 \times \ldots \times A_n).(B_1 \times \ldots \times B_n) \cong (\exists x_1{:}\, A_1.B_1) \times \ldots \times (\exists x_n{:}\, A_n.B_n)$$

where $\Gamma \vdash A_i{:}\, c$ and $\Gamma, x_i{:}\, A_i{:}\, c \vdash B_i(x_i){:}\, r$ for $i = 1, \ldots, n$.

**Proposition 1.6 (Extensivity)** *Given* $f{:}\, A \to 2{:}\, c$, *then* $A \cong A_0 + A_1$, *where* $2 = 1 + 1 \quad A_i = \Sigma a{:}\, A.eq_2(i, fa) \quad x, y{:}\, 2{:}\, c \vdash eq_2(x, y){:}\, c$ *is equality on* 2, *i.e.*

$$eq_2(0,0) = 1 \mid eq_2(0,1) = 0 \mid eq_2(1,0) = 0 \mid eq_2(1,1) = 1$$

**Remark 1.7** Also this property can be proved formally in *extensional* $S2$. The key lemma is $x{:}\, 2{:}\, c \vdash eq_2(0, i) + eq_2(1, i) \cong 1$.

# 2 Translations of $\mathcal{NSC}$ into $S2$

The Nested Sequence Calculus $\mathcal{NSC}$ (see [22]) is an idealized vector language. Unlike NESL, it has a small set of primitive operations and no polymorphism, therefore is simpler to analyze. For the purposes of this paper we introduce the abstract syntax of $\mathcal{NSC}$ and refer the interested reader to [22] for the operational semantics. The syntax of $\mathcal{NSC}$ is parameterized w.r.t. a signature $\Sigma$ of atomic types $D$ and operations $op{:}\, \overline{\tau} \to \tau$

- Types $\tau ::= 1 \mid N \mid D \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid [\tau]$. Arities for operations are of the form $\sigma ::= \overline{\tau} \to \tau$, and those for term-constructors are $\overline{\sigma}, \overline{\tau} \to \tau$.
- Raw terms $e ::= x \mid op(\overline{e}) \mid c(\overline{f}, \overline{e})$, where $c$ ranges over term-constructors (see Figure 1) and $f$ over abstractions $f ::= \lambda \overline{x}{:}\, \overline{\tau}.e$.

**Remark 2.1** $\mathcal{NSC}$ has a term-constructor $while{:}\, (\tau \to \tau), (\tau \to 1 + 1), \tau \to \tau$ instead of $for$. We have decided to ignore the issue of non-termination, to avoid additional complications in $S2$ and translations. One must be rather careful when translating a source language which exhibits non-termination or other computational effects. In fact, in $S2$ such effects should be confined to the run-time part, since we want to keep type-checking and shape-analysis decidable.

The following sections describe in details two translations of $\mathcal{NSC}$ in $S2$. In this section we only outline the translations and give a concise account of them and their properties in terms of categorical models.

## 2.1 The simple translation

The simple translation $\_^*{:}\, \mathcal{NSC} \to S2$ has the following pattern

- types $\vdash_{\mathcal{NSC}} \tau$ are translated to $r$-types $\vdash_{S2} \tau^*{:}\, r$
- terms $\overline{x}{:}\, \overline{\tau} \vdash_{\mathcal{NSC}} e{:}\, \tau$ are translated to terms $\overline{x}{:}\, \overline{\tau}^*{:}\, r \vdash_{S2} e^*{:}\, T\tau^*{:}\, r$
  where $T$ is the error monad on $r$-types.

Term-constructors marked with $*$ can raise an error

| $c$ | arity | informal meaning |
|---|---|---|
| $err*$ | $\tau$ | error |
| $0$ | $N$ | zero |
| $s$ | $N \to N$ | successor |
| $eq$ | $N, N \to 1+1$ | equality of natural numbers |
| $for*$ | $(N, \tau \to \tau), \tau, N \to \tau$ | iteration |
| $*$ | $1$ | empty tuple |
| $pair$ | $\tau_1, \tau_2 \to \tau_1 \times \tau_2$ | pairing |
| $\pi_i$ | $\tau_1 \times \tau_2 \to \tau_i$ | projection |
| $in_i$ | $\tau_i \to \tau_1 + \tau_2$ | injection |
| $case*$ | $(\tau_1 \to \tau), (\tau_2 \to \tau), \tau_1 + \tau_2 \to \tau$ | case analysis |
| nil | $[\tau]$ | empty sequence |
| $sgl$ | $\tau \to [\tau]$ | singleton sequence |
| $at$ | $[\tau], [\tau] \to [\tau]$ | concatenation |
| $flat$ | $[[\tau]] \to [\tau]$ | flattening |
| $map*$ | $(\tau_1 \to \tau_2), [\tau_1] \to [\tau_2]$ | mapping |
| $length$ | $[\tau] \to N$ | length of sequence |
| $get*$ | $[\tau] \to \tau$ | get unique element of sequence |
| $zip*$ | $[\tau_1], [\tau_2] \to [\tau_1 \times \tau_2]$ | zipping |
| $enum$ | $[\tau] \to [N]$ | enumerate elements of sequence |
| $split*$ | $[\tau], [N] \to [[\tau]]$ | splitting of sequence |

Figure 1: Term-constructors of $\mathcal{NSC}$

**Definition 2.2 (Error monad)** *Given $A\colon r$ the type $TA\colon r$ is given by $TA = A+1$. The corresponding monad structure is defined by*

| | |
|---|---|
| $val$ | $A \to TA$ |
| | $val(x) = in_0(x)$ |
| $let$ | $(A \to TB), TA \to TB$ |
| | $let(f, in_0(x)) = f(x)$ |
| | $let(f, in_1(*)) = in_1(*)$ |

*We write $[M]$ for $val(M)$ and* $(\text{let } x \Leftarrow M \text{ in } N)$ *for $let([x\colon A]N, M)$.*

## 2.2 The mixed translation

The mixed translation consists of a pair of translations $(\_^c, \_^r)\colon \mathcal{NSC} \to S2$ s.t.

- types $\vdash_{\mathcal{NSC}} \tau$ are translated to families of $r$-types $x\colon \tau^c\colon c \vdash_{S2} \tau^r(x)\colon r$
- terms $\overline{x}\colon \overline{\tau} \vdash_{\mathcal{NSC}} e\colon \tau$ are translated to pairs of *compatible* terms
  $\overline{x}\colon \overline{\tau}^c\colon c \vdash_{S2} e^c\colon T\tau^c\colon c$ and
  $\overline{x}\colon \overline{\tau}^c\colon c, \overline{x}'\colon \overline{\tau}'\colon r \vdash_{S2} e^r\colon T'([x\colon \tau^c]\tau^r, e^c)\colon r$

  where $(T, T')$ is the error monad on families of $r$-types.

**Definition 2.3 (Error monad on families of types)** *Given $x\colon A\colon c \vdash A'\colon r$ the family $x\colon TA\colon c \vdash T'([x\colon A]A', x)\colon r$ is given by $TA = A + 1$ and*
$T'([x\colon A]A', in_0(x)) = A'(x)$
$T'([x\colon A]A', in_0(*)) = 1$
*We write $T'(A, A', M)$ for $T'([x\colon A]A', M)$.*

*The corresponding monad structure is defined by pair of compatible terms*

| | |
|---|---|
| $val$<br>$val'$ | $A \to TA$<br>$\forall x{:}\, A.A' \to T'(A, A', val(x))$ |
| | $val(x) = in_0(x)$<br>$val'(x, x') = x'$ |
| $let$<br>$let'$ | $(A \to TB), TA \to TB$<br>$\forall f, x{:}\, (A \to TB), TA,$<br>$(\forall x{:}\, A.A' \to T'(B, B', fx)), T'(A, A', x) \to T'(B, B', let(f, x))$ |
| | $let(f, in_0(x)) = f(x)$<br>$let'(f, in_0(x), f', x') = f'(x, x')$<br>$let(f, in_1(*)) = in_1(*)$<br>$let'(f, in_1(*), f', *) = *$ |

*We write* $[M]$ *for* $val(M)$, $(\text{let } x \Leftarrow M \text{ in } N)$ *for* $let([x{:}\, A]N, M)$, $M'$ *for* $val'(M, M')$ *and* $(\text{let}'\, x, x' \Leftarrow M, M' \text{ in } N')$ *for* $let'([x{:}\, A]N, M, [x{:}\, A, x'{:}\, A']N', M')$.

In defining the mixed translation of *collection* types $[\tau]$ we use the list type-constructor $(L, L')$ acting on families of $r$-types.

**Definition 2.4 (List objects for families of types)** *Given* $x{:}\, A{:}\, c \vdash A'{:}\, r$ *the family* $x{:}\, LA{:}\, c \vdash L'([x{:}\, A]A', x){:}\, r$ *is given by* $LA = \Sigma n{:}\, N.V(n, A)$ *and* $L'([x{:}\, A]A', \langle n, v\rangle) = \forall i{:}\, n.A'(vi)$. *We write* $L'(A, A', M)$ *for* $L'([x{:}\, A]A', M)$.

## 2.3 Semantic view and main results

Given a model $\pi{:}\, \mathbf{Fam}(\mathcal{C}) \to \mathbf{Set}$ of $S2$ (see Section 1), one may compose the two translations of $\mathcal{NSC}$ in $S2$ with the interpretation of $S2$ in the model, and thus investigate the properties of the resulting interpretations. In fact, it is often easier to start from a direct interpretation of $\mathcal{NSC}$, and then work out the corresponding translation (with its low level details). In what follows we assume that $\mathcal{C}$ is a cartesian closed category with small products and small sums (as done in Section 1 to ensure that $\pi{:}\, \mathbf{Fam}(\mathcal{C}) \to \mathbf{Set}$ is a model of $S2$).

- The simple translation corresponds to an interpretation of $\mathcal{NSC}$ in the Kleisli category $\mathcal{C}_T$ for the monad $T(\_) = \_ + 1$.
- The mixed translation corresponds to an interpretation of $\mathcal{NSC}$ in the Kleisli category $\mathbf{Fam}(\mathcal{C})_T$ for the monad $T(\_) = \_ + 1$, namely $T(\langle I, c\rangle) = \langle I+1, [c, 1]\rangle$, where $[c, 1] \in \mathcal{C}^{I+1}$ is the family s.t. $[c, 1]_{in_0(i)} = c_i$ and $[c, 1]_{in_1(*)} = 1$.

**Remark 2.5** These interpretations could be parameterized w.r.t. a (strong) monad $S$ on $\mathcal{C}$, i.e.: $TA = S(A + 1)$ in $\mathcal{C}$; $T(\langle I, c\rangle) = \langle I + 1, [c', 1]\rangle$ with $c'_i = S(c_i)$ in $\mathbf{Fam}(\mathcal{C})$. This generalization is interesting because it suggests a way for dealing with non-termination and other computational effects. Unfortunately, in intensional $S2$ it is not possible to mimic the definition of $T$ in $\mathbf{Fam}(\mathcal{C})$ from $S$ (the difficulty is in the definition $let$). The reason is lack of extensivity (see Proposition 1.6).

In order to interpret $\mathcal{NSC}$ in the Kleisli category $\mathcal{A}_T$ for a strong monad $T$ over $\mathcal{A}$, the category $\mathcal{A}$ must have finite products, binary sums and list objects ( satisfying certain additional properties). Moreover, one can always take $T(\_) = \_ + 1$. When $\mathcal{A}$ is cartesian closed and has countable sums, the necessary structure and properties are automatically available.

**Lemma 2.6** *The categories* $\mathbf{Set}$, $\mathcal{C}$ *and* $\mathbf{Fam}(\mathcal{C})$ *are cartesian closed and have small sums. The following are full reflections*

$$\mathbf{Set} \underset{\Delta}{\overset{\pi}{\underset{\longrightarrow}{\longleftarrow}}} \mathbf{Fam}(\mathcal{C}) \underset{\longleftarrow}{\overset{\exists}{\longrightarrow}} \mathcal{C}$$

*Moreover, the functors* $\Delta\colon \mathbf{Set} \to \mathbf{Fam}(\mathcal{C})$, $\pi\colon \mathbf{Fam}(\mathcal{C}) \to \mathbf{Set}$ *and* $\exists\colon \mathbf{Fam}(\mathcal{C}) \to \mathcal{C}$ *preserve finite products and small sums (*$\Delta$ *and* $\pi$ *preserve also exponentials).*

**Proof** The relevant categorical structure in $\mathbf{Fam}(\mathcal{C})$ is defined as follows: $1 = \langle 1, 1\rangle$, $\langle I, a\rangle \times \langle J, b\rangle = \langle I \times J, c\rangle$ with $c_{i,j} = a_i \times b_j$, $\langle J, b\rangle^{\langle I,a\rangle} = \langle J^I, c\rangle$ with $c_f = \prod_{i\in I} b_{fi}^{a_i}$, $\coprod_{i\in I}\langle J_i, b_i\rangle = \langle \coprod_{i\in I} J_i, c\rangle$ with $c_{i,j} = (b_i)_j$. The adjoint functors are given by

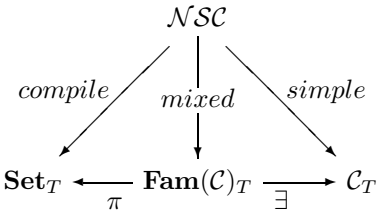
$$I \overset{\pi}{\longleftarrow} \langle I, a\rangle \qquad \langle I, a\rangle \overset{\exists}{\longmapsto} \coprod_{i\in I} a_i$$

$$J \underset{\Delta}{\longmapsto} \langle J, 1\rangle \qquad \langle 1, b\rangle \longleftarrow b$$

A simple check shows that all functors preserve finite products and exponentials, and all functors except $\mathcal{C} \hookrightarrow \mathbf{Fam}(\mathcal{C})$ preserve small sums. ∎

This lemma says that we can interpret $\mathcal{NSC}$ in any of the three categories by taking $T(\_) = \_ + 1$ (and fixing an interpretation for the signature $\Sigma$).

**Theorem 2.7** *The following diagrams commute (up to a natural isomorphism)*




**Remark 2.8** There is a proviso to the above theorem: the simple and mixed interpretation of $\mathcal{NSC}$ are related (as stated), if and only if the simple and mixed interpretation of $\Sigma$ are. The syntactic counterpart of this theorem says that the following assertions are provable (in extensional $S2$):

- $\tau^* \cong \exists x\colon \tau^c.\tau^r$ and $T\tau^* \cong \exists x\colon T\tau^c.T'(\tau^c, \tau^r, x)$;
- $\overline{x}\colon \overline{\tau}^c, \overline{x}'\colon \overline{\tau}^r \vdash [\overline{\langle x_i, x_i'\rangle}/\overline{x}_i]e^* = \langle e^c, e^r\rangle\colon T\tau^*$ (up to isomorphism).

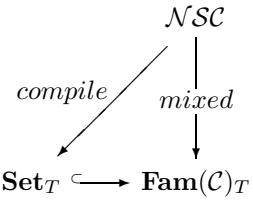
The delicate step in the proof of the syntactic result is the case $[\tau]$, where one should use Proposition 1.4. Informally speaking, the theorem says that the mixed translation *extracts* more information than the simple translation.

**Lemma 2.9** *If* $\mathcal{C}$ *is extensive and non-trivial (i.e.* $0 \ncong 1$*), then* $\exists\colon \mathbf{Fam}(\mathcal{C})(1, x) \to \mathcal{C}(1, \exists x)$ *is injective for any* $x$*.*

Thus one can conclude (when the hypothesis of the lemma are satisfied) that the interpretation of a closed expression of $\mathcal{NSC}$ is equal to of error in the simple semantics if and only if it is in the mixed semantics.

The mixed interpretation of $\mathcal{NSC}$ is rather boring when the interpretation of atomic types in $\Sigma$ are *trivial*, i.e. isomorphic to the terminal object.

**Theorem 2.10** *If the mixed interpretation of base types are trivial, then the following diagram commutes (up to a natural isomorphism)*




**Remark 2.11** The syntactic counterpart of this theorem says that $x{:}\tau^c \vdash \tau^r \cong 1$ is provable (in extensional $S2$). Therefore, the run-time part of $S2$ is not really used.

**Theorem 2.12** *The compile-time part compile*$:\mathcal{NSC} \to \mathbf{Set}_T$ *of the mixed interpretation factors through the full sub-category of countable sets.*

**Remark 2.13** The syntactic counterpart of this result is much stronger, namely (in extensional $S2$) $\tau^c$ is provable isomorphic either to a finite cardinal or to the NNO. This means that the compile-time part of the translation uses very simple types (though the provable isomorphisms may get rather complex).

# 3 The simple translation

The simple translation $\_^*$ corresponds to translate $\mathcal{NSC}$ in a simply typed lambda calculus with unit, sums, products, NNO and list objects (extended with the analogue $\Sigma^*$ of the $\mathcal{NSC}$-signature $\Sigma$).

- types $\vdash_{\mathcal{NSC}} \tau$ are translated to $r$-types $\vdash_{S2} \tau^*{:}\,r$

| $\tau$ of $\mathcal{NSC}$ | $\tau^*{:}\,r$ of $S2$ |
|---|---|
| $1$ | $1$ |
| $N$ | $\exists n{:}\,N.1$ |
| $D$ | $D$ |
| $\tau_1 \times \tau_2$ | $\tau_1^* \times \tau_2^*$ |
| $\tau_1 + \tau_2$ | $\tau_1^* + \tau_2^*$ |
| $[\tau]$ | $\exists n{:}\,N.n \Rightarrow \tau^*$ |
| arities of $\mathcal{NSC}$ | |
| $\overline{\tau} \to \tau$ | $\overline{\tau}^* \to T\tau^*$ |
| $\overline{\sigma}, \overline{\tau} \to \tau$ | $\overline{\sigma}^*, \overline{\tau}^* \to T\tau^*$ |

- terms $\overline{x}{:}\,\overline{\tau} \vdash_{\mathcal{NSC}} e{:}\,\tau$ are translated to terms $\overline{x}{:}\,\overline{\tau}^*{:}\,r \vdash_{S2} e^*{:}\,T\tau^*{:}\,r$

| $e{:}\,\tau$ of $\mathcal{NSC}$ | $e^*{:}\,T\tau^*{:}\,r$ of $S2$ |
|---|---|
| $x$ | $[x]$ |
| $op(\overline{e})$ | let $\overline{x} \Leftarrow \overline{e}^*$ in $op^*(\overline{x})$ |
| $c(\overline{f}, \overline{e})$ | let $\overline{x} \Leftarrow \overline{e}^*$ in $c^*(\overline{f}^*, \overline{x})$ |
| $\lambda\overline{x}{:}\,\overline{\tau}.e$ | $\lambda\overline{x}{:}\,\overline{\tau}^*.e^*$ |

  when a term-constructor $c$ cannot raise an error (i.e. is not marked by $*$ in Figure 1), we translate $c(\overline{f}, \overline{e})$ to let $\overline{x} \Leftarrow \overline{e}^*$ in $[c^*(\overline{f}^*, \overline{x})]$.

- term-constructors $c{:}\,\overline{\sigma}, \overline{\tau} \to \tau$ are translated to terms $\vdash_{S2} c^*{:}\,\overline{\sigma}^*, \overline{\tau}^* \to T\tau^*{:}\,r$, or to $\vdash_{S2} c^*{:}\,\overline{\sigma}^*, \overline{\tau}^* \to \tau^*{:}\,r$ when $c$ cannot raise an error.

Figure 2 gives $c^*$ for the term-constructors $c$ of $\mathcal{NSC}$ which can raise an error (and for *length* and *enum*), the reader could figure out for himself the definition of $c^*$ for the other term-constructors. In Figure 2 we use ML-style notation for function definitions and other auxiliary notation for $S2$, which is defined in Appendix A.

**Remark 3.1** Given a $\mathcal{NSC}$-signature $\Sigma$, its analogue $\Sigma^*$ in $S2$ is defined as follows:

- a constant type $D\colon r$ for each atomic type $D$ in $\Sigma$;
- a constant term $\overline{x}\colon\overline{\tau}^*\colon r \vdash op^*(\overline{x})\colon T\tau^*\colon r$ for each operation $op\colon\overline{\tau}\to\tau$ in $\Sigma$.

# 4 The mixed translation

The mixed translation highlights phase distinction between compile-time and run-time, and exploits fully the features of $S2$ (extended with the analogue of the $\mathcal{NSC}$-signature $\Sigma$). Theorem 2.7 (and Lemma 2.9) says that *shape errors* are detected at compile-time. Theorem 2.10 says that the run-time translation of types is *trivial*, when it is trivial for all base types. Theorem 2.12 says that the compile-time translation of types is very simple, i.e. (up to isomorphism) it is either a finite cardinal or the NNO.

- types $\vdash_{\mathcal{NSC}}\tau$ are translated to families of $r$-types $x\colon\tau^c\colon c\vdash_{S2}\tau^r(x)\colon r$

| $\tau$ of $\mathcal{NSC}$ | $x\colon\tau^c\colon c$ and | $\tau^r(x)\colon r$ of $S2$ |
|---|---|---|
| $1$ | $\_\colon 1$ | $1$ |
| $N$ | $\_\colon N$ | $1$ |
| $D$ | $\_\colon 1$ | $D$ |
| $\tau_1\times\tau_2$ | $\langle x_1,x_2\rangle\colon\tau_1^c\times\tau_2^c$ | $\tau_1^r(x_1)\times\tau_2^r(x_2)$ |
| $\tau_1+\tau_2$ | $in_i(x_i)\colon\tau_1^c+\tau_2^c$ | $\tau_i^r(x_i)$ |
| $[\tau]$ | $\langle n,v\rangle\colon\Sigma n\colon N.V(n,\tau^c)$ | $\forall i\colon n.\tau^r(vi)$ |
| arities of $\mathcal{NSC}$ | | |
| $\overline{\tau}\to\tau$ | $f\colon\overline{\tau}^c\to T\tau^c$ | $\forall\overline{x}\colon\overline{\tau}^c.\overline{\tau}^r\to T'(\tau^c,\tau^r,f(\overline{x}))$ |
| $\overline{\sigma},\overline{\tau}\to\tau$ | $F\colon\overline{\sigma}^c,\overline{\tau}^c\to T\tau^c$ | $\forall\overline{f},\overline{x}\colon\overline{\sigma}^c,\overline{\tau}^c.\overline{\sigma}^r,\overline{\tau}^r\to T'(\tau^c,\tau^r,F(\overline{f},\overline{x}))$ |

- terms $\overline{x}\colon\overline{\tau}\vdash_{\mathcal{NSC}}e\colon\tau$ are translated to pairs of *compatible* terms
  $\overline{x}\colon\overline{\tau}^c\colon c\vdash_{S2}e^c\colon T\tau^c\colon c$ and
  $\overline{x}\colon\overline{\tau}^c\colon c,\overline{x}'\colon\overline{\tau}^r\colon r\vdash_{S2}e^r\colon T'(\tau^c,\tau^r,e^c)\colon r$

| $e\colon\tau$ of $\mathcal{NSC}$ | $e^c\colon T\tau^c\colon c$ and | $e^r\colon T'(\tau^c,\tau^r,e^c)\colon r$ of $S2$ |
|---|---|---|
| $x$ | $[x]$ | $x'$ |
| $op(\overline{e})$ | $\mathrm{let}\,\overline{x}\Leftarrow\overline{e}^c\,\mathrm{in}\,op^c(\overline{x})$ | $\mathrm{let}'\,\overline{x},\overline{x}'\Leftarrow\overline{e}^c,\overline{e}^r\,\mathrm{in}\,op^r(\overline{x},\overline{x}')$ |
| $c(\overline{f},\overline{e})$ | $\mathrm{let}\,\overline{x}\Leftarrow\overline{e}^c\,\mathrm{in}\,c^c(\overline{f}^c,\overline{x})$ | $\mathrm{let}'\,\overline{x},\overline{x}'\Leftarrow\overline{e}^c,\overline{e}^r\,\mathrm{in}\,c^r(\overline{f}^c,\overline{f}^r,\overline{x},\overline{x}')$ |
| $\lambda\overline{x}\colon\overline{\tau}.e$ | $\lambda\overline{x}\colon\overline{\tau}^c.e^c$ | $\Lambda\overline{x}\colon\overline{\tau}^c.\lambda\overline{x}'\colon\overline{\tau}^r.e^r$ |

  when a term-constructor $c$ cannot raise an error, we translate $c(\overline{f},\overline{e})$ to $\mathrm{let}\,\overline{x}\Leftarrow\overline{e}^c\,\mathrm{in}\,[c^c(\overline{f}^c,\overline{x})]$ and $\mathrm{let}'\,\overline{x},\overline{x}'\Leftarrow\overline{e}^c,\overline{e}^r\,\mathrm{in}\,c^r(\overline{f}^c,\overline{f}^r,\overline{x},\overline{x}')$.

- term-constructors $c\colon\overline{\sigma},\overline{\tau}\to\tau$ are translated to pairs of *compatible* terms
  $\vdash_{S2}c^c\colon\overline{\sigma}^c,\overline{\tau}^c\to T\tau^c\colon c$ and
  $\vdash_{S2}c^r\colon\forall\overline{f},\overline{x}\colon\overline{\sigma}^c,\overline{\tau}^c.\overline{\sigma}^r,\overline{\tau}^r\to T'(\tau^c,\tau^r,c^c(\overline{f},\overline{x}))\colon r$, or to
  $\vdash_{S2}c^c\colon\overline{\sigma}^c,\overline{\tau}^c\to\tau^c\colon c$ and
  $\vdash_{S2}c^r\colon\forall\overline{f},\overline{x}\colon\overline{\sigma}^c,\overline{\tau}^c.\overline{\sigma}^r,\overline{\tau}^r\to\tau^r(c^c(\overline{f},\overline{x}))\colon r$ when $c$ cannot raise an error.

Figure 3 and 4 gives $c^c$ and $c^r$ for the term-constructors $c$ of $\mathcal{NSC}$ which can raise an error (and for *length* and *enum*). The tables are organized as follows:

- For each term-constructor $c$ of $\mathcal{NSC}$ we write

| | |
|---|---|
| $c^c$ | $\overline{f},\overline{x}\colon\overline{\sigma},\overline{\tau}\to T(\tau)$ |
| $c^r$ | $\overline{\sigma}',\overline{\tau}'\to T'(\tau,\tau',c^c(\overline{f},\overline{x}))$ |

  to mean that $c^c\colon\overline{\sigma},\overline{\tau}\to T(\tau)$ and $c^r\colon\forall\overline{f},\overline{x}\colon\overline{\sigma},\overline{\tau}.\overline{\sigma}',\overline{\tau}'\to T'(\tau,\tau',c^c(\overline{f},\overline{x}))$ .

| $c^*$ | arity and ML-like definition of $c^*(\overline{f},\overline{x})$ |
|---|---|
| $err^*$ | $T\tau$ |
| | $err^* = in_1(*)$ |
| $for^*$ | $(N, \tau \to T\tau), \tau, N \to T\tau$ |
| | $for^*(f,x,0) = [x]$ |
| | $for^*(f,x,sn) = \text{let } y \Leftarrow for^*(f,x,n) \text{ in } f(n,y)$ |
| $case^*$ | $(\tau_1 \to T\tau), (\tau_2 \to T\tau), \tau_1 + \tau_2 \to T\tau$ |
| | $case^*(f_0, f_1, in_i(x)) = f_i(x) \quad (i = 0,1)$ |
| $map^*$ | $(\tau_1 \to T\tau_2), L\tau_1 \to T(L\tau_2)$ |
| | $map^*(f, nil) = [nil]$ |
| | $map^*(f, h::t) = \text{let } x \Leftarrow f(h) \text{ in let } l \Leftarrow map^*(f,t) \text{ in } [x::l]$ |
| $length^*$ | $L\tau \to N$ |
| | $length^*(nil) = 0$ |
| | $length^*(h::t) = s(length^*(t))$ |
| $get^*$ | $L\tau \to T\tau$ |
| | $get^*(nil) = err^*$ |
| | $get^*(h::t) = case^*([h], err^*, eq^*(0, length^*(t)))$ |
| | where $eq^*: N, N \to 1+1$ is equality for the NNO |
| $zip^*$ | $L\tau_1, L\tau_2 \to T(L(\tau_1 \times \tau_2))$ |
| | $zip^*(nil, nil) = [nil]$ |
| | $zip^*(nil, h_2::t_2) = err^*$ |
| | $zip^*(h_1::t_1, nil) = err^*$ |
| | $zip^*(h_1::t_1, h_2::t_2) = \text{let } l \Leftarrow zip^*(t_1,t_2) \text{ in } [\langle h_1, h_2\rangle :: l]$ |
| $enum^*$ | $L\tau \to LN$ |
| | $enum^*(nil) = nil$ |
| | $enum^*(h::t) = snoc(enum^*(t), length^*(t))$ |
| | where $snoc(nil, x) = x::nil \mid snoc(h::t, x) = h::snoc(t,x)$ |
| $split^*$ | $L\tau, LN \to T(L(L\tau))$ |
| | $split^*(nil, nil) = [nil]$ |
| | $split^*(h::t, nil) = err^*$ |
| | $split^*(x, 0::p) = \text{let } l \Leftarrow split^*(x,p) \text{ in } [nil::l]$ |
| | $split^*(nil, sn::p) = err^*$ |
| | $split^*(h::t, sn::p) = \text{let } l \Leftarrow split^*(t, n::p) \text{ in } cons'(x,l)$ |
| | where $cons'(x, nil) = err^* \mid cons'(x, h::t) = [(x::h)::t]$ |

Figure 2: Simple translation of $\mathcal{NSC}$ term-constructors

- Each case of the ML-like definition of $c^c$ is immediately followed by the corresponding case for $c^r$.
- Arguments of trivial run-time type (i.e. isomorphic to 1) are omitted. This happens in the definition of $for$ and $split$.
- The definition of $c^r$ is omitted when the type of the result is trivial. This happens in the definition of $length$ and $enum$.

**Remark 4.1** Given a $\mathcal{NSC}$-signature $\Sigma$, its analogue $\Sigma^{c,r}$ in $S2$ is defined as follows:

- a constant type $D\colon r$ for each atomic type $D$ in $\Sigma$;
- a pair of compatible constant terms $\overline{x}\colon\overline{\tau}^c\colon c \vdash op^c(\overline{x})\colon T\tau^c\colon c$ and $\overline{x}\colon\overline{\tau}^c\colon c, \overline{x}'\colon\overline{\tau}^r\colon r \vdash op^r(\overline{x},\overline{x}')\colon T'(\tau,\tau',op^c(\overline{x}))\colon r$ for each $op\colon\overline{\tau}\to\tau$ in $\Sigma$.

At the semantic level this translation of $\Sigma$ imposes strong restrictions. For instance, consider the translation of $op\colon D, D \to 1+1$. This is given by $op^c\colon 1+1$ and $op^r\colon D, D \to 1$ (when ignoring arguments of trivial type). Therefore, we cannot interpret $op$ as equality on $D$, because both $op^c$ and $op^r$ are constant. The mixed translation can only cope with *shapely* operations (see [14]), where the shape of the result is determined uniquely by the shape of the arguments. However, in $S2$ one can give a type to non-shapely operations, for instance $op\colon D, D \to (\exists x\colon 1+1.1)$.

# 5 Conclusions and further research

[19] advocates the use of Martin-Löf Type Theory for program construction. This paper advocates the use of Martin-Löf Type Theory as part of an intermediate language $S2$. This avoids two major problems: a programmer does not have to deal with dependent type directly, decidability of type-checking in $S2$ does not rely on strong normalization of run-time expressions (since dependent types are confined to the compile-time part of $S2$). [16] introduces a simply typed language for vectors with an operator $\#\colon\tau\to\#\tau$ to extract shape information from terms. The type $\#\tau$ is like our $\tau^c$, while the term $\#e$ *performs* the translation $e^c$ lazily. Our approach gains in clarity and generality by separating the programming language from the intermediate language. There are many unresolved issues about $S2$ that should be addressed. The following is a partial list with some hints on how one may proceed.

$S2$ is based on intensional type theory (to ensure decidability of type-checking), however most of the semantic properties of translations rely on extensionality. It would be interesting to investigate whether some of the extensional properties considered (e.g. extensivity) could be added safely to intensional type theory.

We have not given an operational semantics for $S2$, probably this can be done relatively easy by borrowing ideas from [7, 6].

$\mathcal{NSC}$ is a simply typed language, while NESL has also ML-like polymorphism. This is likely to require a refinement of $S2$ by adding a sort of *shapes*. Shape theory should provide useful guidelines for such refinement. There are also obvious extensions to the run-time part of $S2$, e.g. recursive types.

One must fill the gap between $S2$ and parallel machines (or already implemented intermediate languages, like VCODE). Moreover, translations should be *efficient* in the sense of [22, 3].

We have used a modified version of $\mathcal{NSC}$ with for-loops rather than while-loops. It should be possible to incorporate run-time computational aspects in $S2$ using monads. In such extension it should be possible to translate more realistic languages.

| $c$ | arity and ML-like definition of $c^c(\overline{f},\overline{x})$ and $c^r(\overline{f},\overline{f}',\overline{x},\overline{x}')$ |
|---|---|
| $err^c$<br>$err^r$ | $T\tau$<br>$T'(\tau,\tau',err^c)$ |
| | $err^c = in_1(*)$<br>$err^r = *$ |
| $for^c$<br>$for^r$ | $f,x,n\colon (N,\tau\to T\tau),\tau,N\to T\tau$<br>$(\forall n,x\colon N,\tau.\tau'\to T'(\tau,\tau',f(n,x))),\tau'(x)\to T'(\tau,\tau',for^c(f,x,n))$ |
| | $for^c(f,x,0) = [x]$<br>$for^r(f,x,0,f',x') = x'$<br>$for^c(f,x,sn) = \text{let}\, y\Leftarrow for^c(f,x,n)\,\text{in}\, f(n,y)$<br>$for^r(f,x,sn,f',x') = \text{let}'\, y,y'\Leftarrow for^c(f,x,n), for^r(f,x,n,f',x')\,\text{in}\, f'(n,y,y')$ |
| $case^c$<br>$case^r$ | $f_1,f_2,in_i(x)\colon(\tau_1\to T\tau),(\tau_2\to T\tau),\tau_1+\tau_2\to T\tau$<br>$(\forall y\colon\tau_1.\tau_1'\to T'(\tau,\tau',f_1y)),(\forall y\colon\tau_2.\tau_2'\to T'(\tau,\tau',f_2y)),\tau_i'(x)\to T'(\tau,\tau',case^c(f_1,f_2,x))$ |
| | $case^c(f_0,f_1,in_i(x)) = f_i(x) \quad (i=0,1)$<br>$case^r(f_0,f_1,in_i(x),f_1',f_2',x') = f_i'(x,x') \quad (i=0,1)$ |
| $map^c$<br>$map^r$ | $f,l\colon(\tau_1\to T\tau_2),L\tau_1\to T(L\tau_2)$<br>$\forall x\colon\tau_1.\tau_1'\to T'(\tau_2,\tau_2',fx),L'(\tau_1,\tau_1',l)\to T'([x\colon L\tau_2]L'(\tau_2,\tau_2',x),map^c(f,l))$ |
| | $map^c(f,nil) = [nil]$<br>$map^r(f,nil,f',[]) = []$<br>$map^c(f,h::t) = \text{let}\, y\Leftarrow f(h)\,\text{in let}\, z\Leftarrow map^c(f,t)\,\text{in}\,[y::z]$<br>$map^r(f,h::t,f',[h',t']) = \text{let}'\, y,y'\Leftarrow f(h),f'(h,h')\,\text{in}$<br>$\text{let}'\, z,z'\Leftarrow map^c(f,t),map^r(f,t,f',t')\,\text{in}\,[y',z']$ |
| $length^c$<br>$length^r$ | $l\colon L\tau\to N$<br>$L'(\tau,\tau',l)\to 1$ |
| | $length^c(nil) = 0$<br>$length^c(h::t) = s(length^c(t))$ |
| $get^c$<br>$get^r$ | $l\colon L\tau\to T\tau$<br>$L'(\tau,\tau',l)\to T'(\tau,\tau',get^c(l))$ |
| | $get^c(nil) = err^c$<br>$get^r(nil,[]) = err^r$<br>$get^c(h::t) = case^c([h],err^c,eq^c(0,length^c(t)))$<br>$get^r(h::t,[h',t']) = case^r([h]),err^c,eq^c(0,length^c(t)),h',err^r,*)$<br>where $eq^c\colon N,N\to 1+1$ is equality for the NNO |
| $zip^c$<br>$zip^r$ | $l_1,l_2\colon L\tau_1,L\tau_2\to T(L(\tau_1\times\tau_2))$<br>$L'(\tau_1,\tau_1',l_1),L'(\tau_2,\tau_2',l_2)\to T'([x\colon L(\tau_1\times\tau_2)]L'(\tau_1\times\tau_2,\tau_1'\times\tau_2',x),zip^c(l_1,l_2))$ |
| | $zip^c(nil,nil) = [nil]$<br>$zip^r(nil,nil,[],[]) = []$<br>$zip^c(nil,h_2::t_2) = err^c$<br>$zip^r(nil,h_2::t_2,\_,\_) = err^r$<br>$zip^c(h_1::t_1,nil) = err^c$<br>$zip^r(h_1::t_1,nil,\_,\_) = err^r$<br>$zip^c(h_1::t_1,h_2::t_2) = \text{let}\, y\Leftarrow zip^c(t_1,t_2)\,\text{in}\,[\langle h_1,h_2\rangle::y]$<br>$zip^r(h_1::t_1,h_2::t_2,[h_1',t_1'],[h_2',t_2']) = \text{let}'\, y,y'\Leftarrow zip^c(t_1,t_2),zip^r(t_1,t_2,t_1',t_2')\,\text{in}$<br>$[\langle h_1',h_2'\rangle,y']$ |
| $enum^c$<br>$enum^r$ | $l\colon L\tau\to LN$<br>$L'(\tau,\tau',l)\to 1$ |
| | $enum^c(nil) = nil$<br>$enum^c(h::t) = snoc^c(enum^c(t),length^c(t))$<br>where $snoc^c(nil,x) = x::nil \mid snoc^c(h::t,x) = h::snoc^c(t,x)$ |

Figure 3: Mixed translation of $\mathcal{NSC}$ term-constructors

| | | | |
|---|---|---|---|
| $split^c$ | $l, k{:}\, L\tau, LN \to T(L(L\tau))$ | | |
| $split^r$ | $L'(\tau, \tau', l) \to T'([x{:}\, L(L\tau)]L'([y{:}\, L\tau]L'(\tau, \tau', y), x), split^c(l, k))$ | | |
| | $split^c(nil, nil)$ | $=$ | $[nil]$ |
| | $split^r(nil, nil, [])$ | $=$ | $[]$ |
| | $split^c(h::t, nil)$ | $=$ | $err^c$ |
| | $split^r(h::t, nil, [h', t'])$ | $=$ | $err^r$ |
| | $split^c(x, 0::p)$ | $=$ | let $y \Leftarrow split^c(x, p)$ in $[nil::y]$ |
| | $split^r(x, 0::p, x')$ | $=$ | let$'$ $y, y' \Leftarrow split^c(x, p), split^r(x, p, x')$ in $[[], y']$ |
| | $split^c(nil, sn::p)$ | $=$ | $err^c$ |
| | $split^r(nil, sn::p, [])$ | $=$ | $err^r$ |
| | $split^c(h::t, sn::p)$ | $=$ | let $y \Leftarrow split^c(t, n::p)$ in $cons'(h, y)$ |
| | $split^r(h::t, sn::p, [h', t'])$ | $=$ | let$'$ $y, y' \Leftarrow split^c(t, n::p), split^r(t, n::p, t')$ in $cons'^r(h, y, h', y')$ |
| | where $cons'^c(x, nil) = err^c \mid cons'^c(x, h::t) = [(x::h)::t]$ | | |
| | where $cons'^r(x, nil, x', []) = err^r \mid cons'^r(x, h::t, x', [h', t']) = [[x', h'], t']$ | | |

Figure 4: Mixed translation of $\mathcal{NSC}$ term-constructors (cont.)

One cannot expect that array bound-checking can all be done at compile-time. Indeed shape analysis proposes a more pragmatic approach, where execution and analysis alternate (see [15]). Existential types should provide a clean way of expressing when shape information is available only at run-time.

In shape theory one can distinguish between arrays and lists (see [13]): the elements of an array have the same shape, those of a list may have different shapes. This suggests a different translation of $\mathcal{NSC}$ into $S2$ worth studying, namely: $[\tau]^c = N \times \tau^c$ and $[\tau]^r(\langle n, x\rangle) = n \Rightarrow \tau^r(x)$.

# References


[1] Lars Birkedal, Mads Tofte, and Magnus Vejlstrup. From Region Inference to von Neumann Machines via Region Representation Inference. In *Proceedings from the 23rd annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1996.

[2] G.E. Blelloch, S. Chatterjee, J.C. Hardwick, J. Sipelstein, and M. Zagha. Implementation of a portable nested data-parallel language. *Journal of Parallel and Distributed Computing*, 21(1), April 1994.

[3] Guy E. Blelloch and John Greiner. A provable time and space efficient implementation of NESL. In *ACM SIGPLAN International Conference on Functional Programming*, pages 213–225, May 1996.

[4] Aurelio Carboni, Stephen Lack, and R.F.C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84:145–158, 1993.

[5] T. Coquand and C. Paulin-Mohring. Inductively defined types. volume 389, LNCS, 1989.

[6] Rowan Davies. A temporal-logic approach to binding-time analysis. In *LICS*, July 1996.

[7] Healfdene Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, 1994.

[8] R. Harper, J. Mitchell, and E. Moggi. Higher-order modules and the phase distinction. In *17th POPL*. ACM, 1990.

[9] R. Harper and G. Morrisett. Compiling polymorphism using intensional type analysis. In *Conference Record of POPL '95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 130–141, San Francisco, California, January 1995.

[10] Martin Hofmann. Dependent types:syntax, semantics, and applications. Summer School on Semantics and Logics of Computation, September 1995.

[11] B. Jacobs, E. Moggi, and T. Streicher. Relating models of impredicative type theories. In *Proceedings of the Conference on Category Theory and Computer Science, Manchester, UK, Sept. 1991*, volume 389 of *LNCS*. Springer Verlag, 1991.

[12] Bart Jacobs. *Categorical Type Theory*. PhD thesis, University of Nijmegen, 1991.

[13] C.B. Jay. Matrices, monads and the fast fourier transform. In *Proceedings of the Massey Functional Programming Workshop 1994*, pages 71–80, 1994.

[14] C.B. Jay. A semantics for shape. *Science of Computer Programming*, 25:251–283, 1995.

[15] C.B. Jay. Shape in computing. *ACM Computing Surveys*, 1996. to appear in Symposium on Models of Programming Languages and Computation in June.

[16] C.B. Jay and M. Sekanina. Shape checking of array programs. Technical Report 96.09, University of Technology, Sydney, 1996.

[17] Eugenio Moggi. A category-theoretic account of program modules. *Math. Struct. in Computer Science*, 1:103–139, 1991.

[18] F. Nielson and H.R. Nielson. *Two-Level Functional Languages*. Number 34 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[19] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's type theory:an introduction*. Number 7 in International series of monographs on computer science. Oxford University Press, New York, 1990.

[20] S. Peyton Jones. Unboxed values as first-class citizens. In *Functional Programming and Computer Architecture*, volume 523 of *LNCS*, 1991.

[21] Andrew M. Pitts. Notes on categorical logic. University of Cambridge, Computer Laboratory, Lent Term 1989.

[22] Dan Suciu and Val Tannen. Efficient compilation of high-level data parallel algorithms. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, June 1994.

[23] Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. *Information and Computation*, 111(2):245–296, June 1994.

[24] Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value lambda-calculus using a stack of regions. In *Proceedings from the 21st annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1994.

# A The 2-level intensional type-theory $S2$

In presenting the rules we follow [19, 10]. The elimination rules for inductive types are over sorts (since $S2$ has no universes).

## A.1 Rules for $c$-types

### A.1.1 $\Pi$-types

$$(\Pi)\ \frac{\Gamma, x{:}\,A{:}\,c \vdash B{:}\,c}{\Gamma \vdash (\Pi x{:}\,A.B){:}\,c} \qquad (\Pi\text{-I})\ \frac{\Gamma, x{:}\,A{:}\,c \vdash M{:}\,B{:}\,c}{\Gamma \vdash (\lambda x{:}\,A.M){:}\,(\Pi x{:}\,A.B){:}\,c}$$

$$(\Pi\text{-E})\ \frac{\Gamma \vdash M{:}\,(\Pi x{:}\,A.B){:}\,c \quad \Gamma \vdash N{:}\,A{:}\,c}{\Gamma \vdash MN{:}\,[N/x]B{:}\,c}$$

### A.1.2 $\Sigma$-types

$$(\Sigma)\ \frac{\Gamma, x{:}\,A{:}\,c \vdash B{:}\,c}{\Gamma \vdash (\Sigma x{:}\,A.B){:}\,c} \qquad (\Sigma\text{-I})\ \frac{\begin{array}{c}\Gamma, x{:}\,A{:}\,c \vdash B(x){:}\,c \\ \Gamma \vdash M{:}\,A{:}\,c \quad \Gamma \vdash N{:}\,B(M){:}\,c\end{array}}{\Gamma \vdash \langle M, N\rangle{:}\,(\Sigma x{:}\,A.B){:}\,c}$$

$$(\Sigma\text{-E})\ \frac{\begin{array}{l}\Gamma, z{:}\,(\Sigma x{:}\,A.B){:}\,c \vdash C(z){:}\,\alpha \\ \Gamma, x{:}\,A{:}\,c, y{:}\,B{:}\,c \vdash M{:}\,C(\langle x, y\rangle){:}\,\alpha \\ \Gamma \vdash N{:}\,(\Sigma x{:}\,A.B){:}\,c\end{array}}{\Gamma \vdash R^{\Sigma}([x{:}\,A, y{:}\,B]M, N){:}\,C(N){:}\,\alpha} \qquad (\Sigma\text{-E-}\alpha)\ \frac{\begin{array}{c}\Gamma, x{:}\,A{:}\,c, y{:}\,B{:}\,c \vdash C{:}\,\alpha \\ \Gamma \vdash N{:}\,(\Sigma x{:}\,A.B){:}\,c\end{array}}{\Gamma \vdash R^{\Sigma}([x{:}\,A, y{:}\,B]C, N){:}\,\alpha}$$

### A.1.3 Sums

$$(+)\ \frac{\Gamma \vdash A_i{:}\,c \quad (i = 0, 1)}{\Gamma \vdash A_0 + A_1{:}\,c} \qquad (+\text{-I})\ \frac{\begin{array}{c}\Gamma \vdash A_i{:}\,c \quad (i = 0, 1) \\ \Gamma \vdash M{:}\,A_i{:}\,c\end{array}}{\Gamma \vdash in_i(M){:}\,A_0 + A_1{:}\,c}$$

$$(+\text{-E})\ \frac{\begin{array}{l}\Gamma, z{:}\,A_0 + A_1{:}\,c \vdash C(z){:}\,\alpha \\ \Gamma, x{:}\,A_i{:}\,c \vdash M_i{:}\,C(in_i(x)){:}\,\alpha \quad (i = 0, 1) \\ \Gamma \vdash N{:}\,A_0 + A_1{:}\,c\end{array}}{\Gamma \vdash R^{+}([x{:}\,A_0]M_0, [x{:}\,A_1]M_1, N){:}\,C(N){:}\,\alpha}$$

$$(+\text{-E-}\alpha)\ \frac{\begin{array}{c}\Gamma, x{:}\,A_i{:}\,c \vdash C_i{:}\,\alpha \quad (i = 0, 1) \\ \Gamma \vdash N{:}\,A_0 + A_1{:}\,c\end{array}}{\Gamma \vdash R^{+}([x{:}\,A_0]C_0, [x{:}\,A_1]C_1, N){:}\,\alpha}$$

### A.1.4 Unit

$$(1)\ \frac{\Gamma \vdash}{\Gamma \vdash 1{:}\,c} \qquad (1\text{-I})\ \frac{\Gamma \vdash}{\Gamma \vdash *{:}\,1{:}\,c} \qquad (1\text{-E})\ \frac{\begin{array}{l}\Gamma, z{:}\,1{:}\,c \vdash C(z){:}\,\alpha \\ \Gamma \vdash M{:}\,C(*){:}\,\alpha \\ \Gamma \vdash N{:}\,1{:}\,c\end{array}}{\Gamma \vdash R^{1}(M, N){:}\,C(N){:}\,\alpha}$$

### A.1.5 Natural number object

$$(N)\ \frac{\Gamma \vdash}{\Gamma \vdash N{:}\,c} \qquad (N\text{-}0)\ \frac{\Gamma \vdash}{\Gamma \vdash 0{:}\,N{:}\,c} \qquad (N\text{-}s)\ \frac{\Gamma \vdash M{:}\,N{:}\,c}{\Gamma \vdash s(M){:}\,N{:}\,c}$$

$$(N\text{-E})\ \frac{\begin{array}{l}\Gamma, n\colon N\colon c \vdash A(n)\colon \alpha \\ \Gamma \vdash M_0\colon A(0)\colon \alpha \\ \Gamma, n\colon N\colon c, x\colon A(n)\colon \alpha \vdash M_s\colon A(sn)\colon \alpha \\ \Gamma \vdash m\colon N\colon c\end{array}}{\Gamma \vdash R^N(M_0, [n\colon N, x\colon A(n)]M_s, m)\colon A(m)\colon \alpha}$$

#### A.1.6 Finite cardinals

$$(n)\ \frac{\Gamma \vdash n\colon N\colon c}{\Gamma \vdash n\colon c} \qquad (s\text{-}0)\ \frac{\Gamma \vdash n\colon N\colon c}{\Gamma \vdash 0\colon sn\colon c} \qquad (s\text{-}s)\ \frac{\begin{array}{l}\Gamma \vdash n\colon N\colon c \\ \Gamma \vdash M\colon n\colon c\end{array}}{\Gamma \vdash sM\colon sn\colon c}$$

$$(0\text{-E})\ \frac{\begin{array}{l}\Gamma, i\colon 0\colon c \vdash C(i)\colon \alpha \\ \Gamma \vdash M\colon 0\colon c\end{array}}{\Gamma \vdash R^0(M)\colon C(M)\colon \alpha} \qquad (s\text{-E})\ \frac{\begin{array}{l}\Gamma \vdash n\colon N\colon c \\ \Gamma, i\colon sn\colon c \vdash C(i)\colon \alpha \\ \Gamma \vdash M_0\colon C(0)\colon \alpha \\ \Gamma, i\colon n\colon c \vdash M_s\colon C(si)\colon \alpha \\ \Gamma \vdash P\colon sn\colon c\end{array}}{\Gamma \vdash R^s(M_0, [i\colon n]M_s, P)\colon C(P)\colon \alpha}$$

#### A.1.7 Arrays

$$(V)\ \frac{\begin{array}{l}\Gamma \vdash n\colon N\colon c \\ \Gamma \vdash A\colon c\end{array}}{\Gamma \vdash V(n, A)\colon c} \qquad (V_0\text{-I})\ \frac{\Gamma \vdash A\colon c}{\Gamma \vdash []\colon V(0, A)\colon c}$$

$$(V_s\text{-I})\ \frac{\begin{array}{l}\Gamma \vdash n\colon N\colon c \\ \Gamma \vdash M\colon A\colon c \\ \Gamma \vdash N\colon V(n, A)\colon c\end{array}}{\Gamma \vdash [M, N]\colon V(sn, A)\colon c} \qquad (V_0\text{-E})\ \frac{\begin{array}{l}\Gamma, z\colon V(0, A)\colon c \vdash C(z)\colon \alpha \\ \Gamma \vdash M\colon C([])\colon \alpha \\ \Gamma \vdash N\colon V(0, A)\colon c\end{array}}{\Gamma \vdash R^{V_0}(M, N)\colon C(N)\colon \alpha}$$

$$(V_s\text{-E})\ \frac{\begin{array}{l}\Gamma \vdash n\colon N\colon c \\ \Gamma, z\colon V(sn, A)\colon c \vdash C(z)\colon \alpha \\ \Gamma, x\colon A\colon c, y\colon V(n, A)\colon c \vdash M\colon C([x, y])\colon \alpha \\ \Gamma \vdash N\colon V(sn, A)\colon c\end{array}}{\Gamma \vdash R^{V_s}([x\colon A, y\colon V(n, A)]M, N)\colon C(N)\colon \alpha}$$

**Remark A.1** In a stronger version of *intensional* $S2$, where sort $c$ and $r$ are universes (i.e. types of some bigger sort), one could have defined finite cardinals and arrays by induction on the natural numbers

$$F(0) = 0 \mid F(sn) = 1 + F(n) \qquad V(0, A) = 1 \mid V(sn, A) = A \times V(n, A)$$

and derived the corresponding introduction and elimination rules. We have not used the stronger version of intensional $S2$, because its categorical semantics is more involved. On the other hand, the categorical models of $S2$ are *extensional* (see Section 1), and the interpretation of finite cardinals and arrays is defined in terms of the NNO by exploiting extensionality.

### A.2 Rules for $r$-types

#### A.2.1 $\forall$-types

$$(\forall)\ \frac{\Gamma, x\colon A\colon c \vdash B\colon r}{\Gamma \vdash (\forall x\colon A.B)} \qquad (\forall\text{-I})\ \frac{\Gamma, x\colon A\colon c \vdash M\colon B\colon r}{\Gamma \vdash (\Lambda x\colon A.M)\colon (\forall x\colon A.B)\colon r}$$

$$(\forall\text{-E})\ \frac{\Gamma \vdash M\colon (\forall x\colon A.B)\colon r \quad \Gamma \vdash N\colon A\colon c}{\Gamma \vdash MN\colon [N/x]B\colon r}$$

### A.2.2 $\exists$-types

$$(\exists)\ \frac{\Gamma, x\colon A\colon c \vdash B\colon r}{\Gamma \vdash (\exists x\colon A.B)\colon r} \quad (\exists\text{-I})\ \frac{\begin{array}{l}\Gamma, x\colon A\colon c \vdash B(x)\colon r\\ \Gamma \vdash M\colon A\colon c\\ \Gamma \vdash N\colon B(M)\colon r\end{array}}{\Gamma \vdash (\langle M, N\rangle)\colon (\exists x\colon A.B)\colon r}$$

$$(\exists\text{-E})\ \frac{\begin{array}{l}\Gamma \vdash C\colon r\\ \Gamma, x\colon A\colon c, y\colon B\colon r \vdash N\colon C\colon r\\ \Gamma \vdash M\colon (\exists x\colon A.B)\colon r\end{array}}{\Gamma \vdash R^{\exists}([x\colon A, y\colon B]M, N)\colon C\colon r}$$

### A.2.3 $\times$-types

$$(\times)\ \frac{\Gamma \vdash A_i\colon r \quad (i = 0, 1)}{\Gamma \vdash (A \times B)\colon r} \quad (\times\text{-I})\ \frac{\Gamma \vdash M_i\colon A_i\colon r \quad (i = 0, 1)}{\Gamma \vdash \langle M_0, M_1\rangle\colon A_0 \times A_1\colon r}$$

$$(\times\text{-E})\ \frac{\Gamma \vdash M\colon A_0 \times A_1\colon r}{\Gamma \vdash \pi_i(M)\colon A_i\colon r}$$

### A.2.4 Sums

$$(+)\ \frac{\Gamma \vdash A_i\colon r \quad (i = 0, 1)}{\Gamma \vdash A_0 + A_1\colon r} \quad (+\text{-I})\ \frac{\begin{array}{l}\Gamma \vdash A_i\colon r \quad (i = 0, 1)\\ \Gamma \vdash M\colon A_i\colon r\end{array}}{\Gamma \vdash in_i(M)\colon A_0 + A_1\colon r}$$

$$(+\text{-E})\ \frac{\begin{array}{l}\Gamma, x\colon A_i\colon c \vdash M_i\colon C\colon r \quad (i = 0, 1)\\ \Gamma \vdash N\colon A_0 + A_1\colon r\end{array}}{\Gamma \vdash R^{+}([x\colon A_0]M_0, [x\colon A_1]M_1, N)\colon C\colon r}$$

### A.2.5 $\rightarrow$-types

$$(\rightarrow)\ \frac{\Gamma \vdash A, B\colon r}{\Gamma \vdash (A \rightarrow B)\colon r} \quad (\rightarrow\text{-I})\ \frac{\Gamma, x\colon A\colon r \vdash M\colon B\colon r}{\Gamma \vdash (\lambda x\colon A.M)\colon (A \rightarrow B)\colon r}$$

$$(\rightarrow\text{-E})\ \frac{\Gamma \vdash M\colon (A \rightarrow B)\colon r \quad \Gamma \vdash N\colon A\colon r}{\Gamma \vdash MN\colon B\colon r}$$

### A.2.6 Unit

$$(1)\ \frac{\Gamma \vdash}{\Gamma \vdash 1\colon r} \quad (1\text{-I})\ \frac{\Gamma \vdash}{\Gamma \vdash *\colon 1\colon r}$$

**Remark A.2** Sums, products and unit types of sort $r$ could have been defined in terms of finite cardinals, universal and existential types.

## A.3 Computational rules

This section summarizes the computational rules on raw terms:

- $(\lambda x\colon A.M)N = [N/x]M$
- $R^{\Sigma}([x\colon A, y\colon B]M, \langle P, Q\rangle) = [P, Q/x, y]M$
- $R^{+}([x\colon A_0]M_0, [x\colon A_1]M_1, in_i(N)) = [N/x]M_i$
- $R^{1}(M, *) = M$

- $R^N(M_0, [n\colon N, x\colon A(n)]M_s, 0) = M_0$
  $R^N(M_0, [n\colon N, x\colon A(n)]M_s, s(N)) = [N, R^N(M_0, [n\colon N, x\colon A(n)]M_s, N)/n, x]M_s$
- $R^s(M_0, [x\colon n]M_s, 0) = M_0$
  $R^s(M_0, [x\colon n]M_s, s(P)) = [P/x]M_s$
- $R^{V_0}(M, []) = M$
  $R^{V_s}([x\colon A, y\colon V(n, A)]M, [P, Q]) = [P, Q/x, y]M$
- $(\Lambda x\colon A.M)N = [N/x]M$
- $R^{\exists}([x\colon A, y\colon B]M, \langle P, Q\rangle) = [P, Q/x, y]M$
- $\pi_i(\langle M_0, M_1\rangle) = M_i$

## A.4 Auxiliary notation and notational conventions

This section introduces auxiliary notations and notational conventions for $S2$.

### A.4.1 Auxiliary notation for types

- $A \Rightarrow B$ stands for $\forall \_\colon A.B$
- $A \to B$ stands for $\Pi \_\colon A.B$ when $A$ and $B$ have sort $c$
- $A \times B$ stands for $\Sigma \_\colon A.B$ when $A$ and $B$ have sort $c$
- $A^n$ stands for $V(n, A)$
- $N^r$ stands for $\exists n\colon N.1$ (the NNO of sort $r$)
- $L(A)$ stands for $\Sigma n\colon N.V(n, A)$ (the list object of sort $c$)
- $V^r(n, [i\colon n]A)$ with $n\colon N\colon c$ and $i\colon n\colon c \vdash A\colon r$ stands for $\forall i\colon n.A(i)$, i.e. the $r$-type of heterogeneous arrays of size $n$
- $L^r(A)$ stands for $\exists n\colon N.n \Rightarrow A$ (the list object of sort $r$)

When there is no ambiguity with sorts the superscript $r$ is omitted.

### A.4.2 ML-style notation for function definitions

- $f(\langle x, y\rangle) = M(x, y)$ stands for $f(z) = R^{\Sigma}([x\colon A, y\colon B]M, z)$ or $f(z) = R^{\exists}([x\colon A, y\colon B]M, z)$ or $f(z) = M(\pi_0(z), \pi_1(z))$ depending on the domain of $f$
- $f(in_i(x)) = M_i(x) \quad (i = 0, 1)$ stands for $f(z) = R^{+}([x\colon A_0]M_0, [x\colon A_1]M_1, z)$
- $f(*) = M$ stands for $f(z) = R^1(M, z)$
- $\begin{cases} f(0) & = & M_0 \\ f(sn) & = & M_s(n, f(n)) \end{cases}$ stands for $f(z) = R^N(M_0, [n\colon N, x\colon A(n)]M_s, z)$
- $(M_0, [x\colon n]M_s)$ stands for $f(z\colon sn) = R^s(M_0, [x\colon n]M_s, z)$ and $()$ stands for $R^0$
- $f([]) = M$ stands for $f(z) = R^{V_0}(M, z)$
  $f([x, v]) = M$ stands for $f(z) = R^{V_s}([x\colon A, v\colon V(n, A)]M, z)$ when $n$ is clear from the context.
- array selection $select\colon \Pi n\colon N.V(n, A) \to n \to A$ is given by
  $$\begin{array}{lcl} select(0, []) & = & () \\ select(sn, [a, v]) & = & (a, select(n, v)) \end{array}$$
  and we write $MN$ for $select(n, M, N)$ when $M\colon V(n, A)\colon c$ and $N\colon n\colon c$.

### A.4.3 Auxiliary notation for derived universal objects

- NNO $N^r$ of sort $r$

| | |
|---|---|
| $0^r$ | $N^r$ |
| | $0^r = \langle 0, * \rangle$ |
| $s^r$ | $N^r \to N^r$ |
| | $s^r(\langle n, * \rangle) = \langle sn, * \rangle$ |
| $R^{N^r}$ | $A, (N^r, A \to A), N^r \to A$ |
| | $R^{N^r}(x, f, \langle 0, * \rangle) = x$<br>$R^{N^r}(x, f, \langle sn, * \rangle) = f(n, R^{N^r}(x, f, \langle n, * \rangle))$ |

- $r$-type of heterogeneous arrays $V^r(n, [i{:}\, n]A)$ with $n{:}\, N {:}\, c$ and $i{:}\, n{:}\, c \vdash A{:}\, r$

| | |
|---|---|
| $[]^r$ | $V^r(0, [i{:}\, 0]A)$ |
| | $[]^r = ()$ |
| $[\_, \_]^r$ | $A(0), V^r(n, [i{:}\, n]A(si)) \to V^r(sn, [i{:}\, sn]A)$ |
| | $[a, v]) = (a, v)$ |
| $R^{V_0^r}$ | $B, V^r(0, [i{:}\, 0]A) \to B$ |
| | $R^{V_0^r}(b, \_) = b$ |
| $R^{V_s^r}$ | $(A(0), V^r(n, i{:}\, nA(si)) \to B), V^r(sn, [i{:}\, sn]A) \to B$ |
| | $R^{V_s^r}(f, v) = f(v_0, v_s)$<br>where $v_0 = v(0)$ and $v_s = \Lambda i{:}\, n.v(si)$ |

- list object $L(A)$ of sort $c$

| | |
|---|---|
| nil | $L(A)$ |
| | $0 = \langle 0, [] \rangle$ |
| *cons* | $A, L(A) \to L(A)$ |
| | $cons(a, \langle n, v \rangle) = \langle sn, [a, v] \rangle$ |
| $R^L$ | $B([]), (l{:}\, L(A), a{:}\, A, B(l) \to B([a, l])), l{:}\, L(A) \to B(l)$ |
| | $R^L(b, f, \langle 0, [] \rangle) = b$<br>$R^L(b, f, \langle sn, [a, v] \rangle) = f(\langle n, v \rangle, a, R^L(b, f, \langle n, v \rangle))$ |

we may write $M :: N$ for $cons(M, N)$

- list object $L^r(A)$ of sort $r$

| | |
|---|---|
| $\text{nil}^r$ | $L^r(A)$ |
| | $0^r = \langle 0, () \rangle$ |
| $cons^r$ | $A, L^r(A) \to L^r(A)$ |
| | $cons^r(a, \langle n, l \rangle) = \langle sn, (a, l) \rangle$ |
| $R^{L^r}$ | $B, (L^r(A), A, B \to B), L^r(A) \to B$ |
| | $R^{L^r}(b, f, \langle 0, \_ \rangle) = b$<br>$R^{L^r}(b, f, \langle sn, l \rangle) = f(\langle n, l_s \rangle, l_0, R^{L^r}(b, f, \langle n, l_s \rangle))$<br>where $l_0 = l(0)$ and $l_s = \Lambda i{:}\, n.l(si)$ |

we may write $M :: N$ for $cons^r(M, N)$

ML-style notation for function definitions will be used also for these derived types. When there is no ambuguity, we may drop the superscript $r$.

# Contents