

# Formalizing the implementation of Streaming NESL

Dandan Xue

October 16, 2017

## **Abstract**

Streaming NESL (SNESL) is a first-order functional, nested data-parallel language, employing a streaming execution model and integrating with a cost model that can predict both time and space complexity. The experimentation has demonstrated good performance of SNESL's implementation and positive empirical evidence of the validity of the code model. In this thesis, we first present non-trivial extension to SNESL's target language, SVCODE, which enables SNESL to support recursion at the same time preserve the cost. Then the formalization of the semantics of this low-level streaming language is given. Finally, we present the proof of the correctness of SNESL's implementation model.

# Contents

<b>1</b>	<b>Formalization</b>	<b>2</b>
1.1	<b>SNESL<sub>0</sub></b>	2
1.1.1	Syntax	2
1.1.2	Typing rules	3
1.1.3	Semantics	4
1.2	<b>SVCODE<sub>0</sub></b>	4
1.2.1	Syntax	4
1.2.2	Instruction semantics	6
1.2.3	Xducer semantics	7
1.2.4	<b>SVCODE<sub>0</sub></b> determinism	8
1.3	Translation	12
1.4	Value representation	13
1.5	Correctness	14
1.5.1	Definitions	14
1.5.2	Correctness proof	22
1.6	Scaling up	31

# Chapter 1

## Formalization

In this chapter, we will present the formal proof of the correctness of the language  $\text{SNESL}_0$ , a subset of  $\text{SNESL}_1$ . First its definition and semantics will be given. Then  $\text{SVCODE}_0$ , the target language of  $\text{SNESL}_0$ , is defined, and proof of its determinism is given. The value representation and translation from  $\text{SNESL}_0$  to  $\text{SVCODE}_0$  are also formalized. Finally we show the proof of the main correctness theorem of this language.

### 1.1 $\text{SNESL}_0$

The language  $\text{SNESL}_0$  we will proof in this chapter is a subset of  $\text{SNESL}_1$ , with mainly the following simplifications:

- only one primitive type **int**
- no pairs
- selected built-in functions
- restricted comprehension is removed
- no user-defined functions

#### 1.1.1 Syntax

(1) The types of  $\text{SNESL}_0$  are:

$$\tau ::= \mathbf{int} \mid \{\tau_1\}$$

(2) The syntax of  $\text{SNESL}_0$  values :

$$n \in \mathbb{Z}$$

$$v ::= n \mid \{v_1, \dots, v_k\}$$

(3) The syntax of  $\text{SNESL}_0$  expressions and the built-in function are shown in Figure 1.1. Note that constants can be generated by calling the built-in function **const**<sub>*n*</sub>() for decreasing expression cases. Also, the arguments of built-in functions as well as the bound sequence in general comprehension are variables instead of expressions; we can simply convert them into their general forms by adding let-bindings of these variables.

$e ::= x$	(variable)
$\mid \text{let } x = e_1 \text{ in } e_2$	(let-binding)
$\mid \phi(x_1, \dots, x_k)$	(built-in function call)
$\mid \{e : x \text{ in } y \text{ using } x_1, \dots, x_j\}$	(general comprehension)
$\phi ::= \text{const}_n \mid \text{iota} \mid \text{plus}$	

**Figure 1.1:** SNESL<sub>0</sub> expressions and built-in functions

### 1.1.2 Typing rules

The typing environment  $\Gamma$  is a mapping from variables to types:

$$\Gamma = [x_1 \mapsto \tau_1, \dots, x_i \mapsto \tau_i]$$

- Expression typing rules:

**Judgment**  $\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma \vdash x : \tau} (\Gamma(x) = \tau) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma[x \mapsto \tau_1] \vdash e_2 : \tau}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau}$$

$$\frac{\phi : (\tau_1, \dots, \tau_k) \rightarrow \tau}{\Gamma \vdash \phi(x_1, \dots, x_k) : \tau} ((\Gamma(x_i) = \tau_i)_{i=1}^k)$$

$$\frac{[x \mapsto \tau_1, (x_i \mapsto \text{int})_{i=1}^j] \vdash e : \tau}{\Gamma \vdash \{e : x \text{ in } y \text{ using } x_1, \dots, x_j\} : \{\tau\}} (\Gamma(y) = \{\tau_1\}, (\Gamma(x_i) = \text{int})_{i=1}^j)$$

- Built-in functions typing rules:

**Judgment**  $\boxed{\phi : (\tau_1, \dots, \tau_k) \rightarrow \tau}$

$$\frac{}{\text{const}_n : () \rightarrow \text{int}} \qquad \frac{}{\text{iota} : (\text{int}) \rightarrow \{\text{int}\}} \qquad \frac{}{\text{plus} : (\text{int}, \text{int}) \rightarrow \text{int}}$$

- Value typing rules:

**Judgment**  $\boxed{v : \tau}$

$$\frac{}{n : \text{int}} \qquad \frac{(v_i : \tau)_{i=1}^k}{\{v_1, \dots, v_k\} : \{\tau\}}$$

### 1.1.3 Semantics

The evaluation environment  $\rho$  is a mapping from variables to values:

$$\rho = [x_1 \mapsto v_1, \dots, x_i \mapsto v_i]$$

- Expression evaluation rules:

**Judgment**  $\boxed{\rho \vdash e \downarrow v}$

$$\frac{}{\rho \vdash x \downarrow v} (\rho(x) = v)$$

$$\frac{\rho \vdash e_1 \downarrow v_1 \quad \rho[x \mapsto v_1] \vdash e_2 \downarrow v}{\rho \vdash \mathbf{let} \ e_1 = x \ \mathbf{in} \ e_2 \downarrow v}$$

$$\frac{\phi(v_1, \dots, v_k) \downarrow v}{\rho \vdash \phi(x_1, \dots, x_k) \downarrow v} ((\rho(x_i) = v_i)_{i=1}^k)$$

$$\frac{([x \mapsto v_i, (x_i \mapsto n_i)_{i=1}^j] \vdash e \downarrow v'_i)_{i=1}^k}{\rho \vdash \{e : x \ \mathbf{in} \ y \ \mathbf{using} \ x_1, \dots, x_j\} \downarrow \{v'_1, \dots, v'_k\}} (\rho(y) = \{v_1, \dots, v_k\}, (\rho(x_i) = n_i)_{i=1}^j)$$

- Built-in function evaluation rules:

**Judgment**  $\boxed{\phi(v_1, \dots, v_k) \downarrow v}$

$$\frac{}{\mathbf{const}_n() \downarrow n}$$

$$\frac{}{\mathbf{iota}(n) \downarrow \{0, 1, \dots, n-1\}} (n \geq 0)$$

$$\frac{}{\mathbf{plus}(n_1, n_2) \downarrow n_3} (n_3 = n_1 + n_2)$$

## 1.2 SVCODE<sub>0</sub>

The target language of SNESSL<sub>0</sub> is also a subset of SVCODE presented in the last chapter. We will call it SVCODE<sub>0</sub>.

### 1.2.1 Syntax

In this minimal language, a primitive stream  $\vec{a}$  can be a vector of booleans, integers or units, as the following grammar shows:

$$b \in \mathbb{B} = \{\mathbf{T}, \mathbf{F}\}$$

$$a ::= n \mid b \mid ()$$

$$\vec{b} = \langle b_1, \dots, b_i \rangle$$

$$\vec{c} = \langle (), \dots, () \rangle$$

$$\vec{a} = \langle a_1, \dots, a_i \rangle$$

The syntax of SVCODE<sub>0</sub> is given in Figure 1.2.

$$\begin{aligned}
p &::= \epsilon \\
&\quad | s := \psi(s_1, \dots, s_k) \\
&\quad | S_{out} := \text{WithCtrl}(s, S_{in}, p_1) \\
&\quad | p_1; p_2 \\
s &::= 0 \mid 1 \dots \in \mathbf{SId} = \mathbb{N} && \text{(stream ids)} \\
\psi &::= \text{Const}_a \mid \text{ToFlags} \mid \text{Usum} \mid \text{MapTwo}_{\oplus} \mid \text{ScanPlus}_{n_0} \mid \text{Distr} && \text{(Xducers)} \\
S &::= \{s_1, \dots, s_i\} \in \mathbb{S} && \text{(a set of stream ids)}
\end{aligned}$$

**Figure 1.2:** Abstract syntax of  $\text{SVCODE}_0$

The function  $\text{dv}$  returns the set of defined variables of a given  $\text{SVCODE}_0$  program.

$$\begin{aligned}
\text{dv}(\epsilon) &= \emptyset \\
\text{dv}(s := \psi(s_1, \dots, s_k)) &= \{s\} \\
\text{dv}(S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1)) &= S_{out} \\
\text{dv}(p_1; p_2) &= \text{dv}(p_1) \cup \text{dv}(p_2)
\end{aligned}$$

Correspondingly,  $\text{fv}$  returns the free variables set.

$$\begin{aligned}
\text{fv}(\epsilon) &= \emptyset \\
\text{fv}(s := \psi(s_1, \dots, s_i)) &= \{s_1, \dots, s_k\} \\
\text{fv}(S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1)) &= \{s_c\} \cup S_{in} \\
\text{fv}(p_1; p_2) &= \text{fv}(p_1) \cup (\text{fv}(p_2) - \text{dv}(p_1))
\end{aligned}$$

An immediate property of this language is that the defined variables of a *well-formed*  $\text{SVCODE}_0$  program are always fresh. In other words, there is no overlapping between the free variables and the newly generated ones.

**Definition 1.1 (Well-formedness).** *p is a well-formed  $\text{SVCODE}$  program, written as  $\Vdash p$ , if one of the following rules applies:*

**Judgment**  $\boxed{\Vdash p}$

$$\begin{array}{c}
\overline{\Vdash \epsilon} \qquad \overline{\Vdash s := \psi(s_1, \dots, s_k)} \quad (s \notin \{s_1, \dots, s_k\}) \\
\\
\overline{\Vdash p_1} \quad \overline{\Vdash S_{out} := \text{WithCtrl}(s, S_{in}, p_1)} \quad \left( \begin{array}{l} \text{fv}(p_1) \subseteq S_{in} \\ S_{out} \subseteq \text{dv}(p_1) \\ (\{s\} \cup S_{in}) \cap \text{dv}(p_1) = \emptyset \end{array} \right)
\end{array}$$

$$\frac{\Vdash p_1 \quad \Vdash p_2}{\Vdash (p_1; p_2)} \left( (\text{fv}(p_1) \cup \text{dv}(p_1)) \cap \text{dv}(p_2) = \emptyset \right)$$

**Lemma 1.2 (Freshness).** *If  $p$  is well-formed, then  $\text{fv}(p) \cap \text{dv}(p) = \emptyset$ .*

The proof is straightward by induction on the derivation of  $\Vdash p$ .

### 1.2.2 Instruction semantics

Before showing the semantics, we first introduce some notations and operations about streams for convenience.

**Notation 1.3.** *Let  $\langle a_0 | \vec{a} \rangle$  denote a non-empty stream  $\langle a_0, a_1, \dots, a_i \rangle$  for some  $\vec{a} = \langle a_1, \dots, a_i \rangle$ .*

**Notation 1.4 (Stream concatenation).**  $\langle a_1, \dots, a_i \rangle ++ \langle a'_1, \dots, a'_j \rangle = \langle a_1, \dots, a_i, a'_1, \dots, a'_j \rangle$

The operational semantics of  $\text{SVCODE}_0$  is given in Figure 1.3. The runtime environment or store  $\sigma$  is a mapping from stream variables to vectors:

$$\sigma = [s_1 \mapsto \vec{a}_1, \dots, s_i \mapsto \vec{a}_i]$$

The control stream  $\vec{c}$ , which is basically a vector of units representing an unary number, indicates the *parallel degree* of the computation. It is worth noting that only in the rule P-WC-NONEMP the control stream has a chance to get changed.

**Judgment**  $\boxed{\langle p, \sigma \rangle \Downarrow^{\vec{c}} \sigma'}$

$$\text{P-EMPTY: } \frac{}{\langle \epsilon, \sigma \rangle \Downarrow^{\vec{c}} \sigma}$$

$$\text{P-XDUCER: } \frac{\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\vec{c}} \vec{a}}{\langle s := \psi(s_1, \dots, s_k), \sigma \rangle \Downarrow^{\vec{c}} \sigma[s \mapsto \vec{a}]} \left( (\sigma(s_i) = \vec{a}_i)_{i=1}^k \right)$$

$$\text{P-WC-EMP: } \frac{}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1), \sigma \rangle \Downarrow^{\vec{c}} \sigma[(s_i \mapsto \langle \rangle)_{i=1}^l]} \left( \forall s \in \{s_c\} \cup S_{in}. \sigma(s) = \langle \rangle \right) \\ S_{out} = \{s_1, \dots, s_l\}$$

$$\text{P-WC-NONEMP: } \frac{\langle p_1, \sigma \rangle \Downarrow^{\vec{c}_1} \sigma''}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1), \sigma \rangle \Downarrow^{\vec{c}} \sigma[(s_i \mapsto \sigma''(s_i))_{i=1}^l]} \left( \sigma(s_c) = \vec{c}_1 \neq \langle \rangle \right) \\ S_{out} = \{s_1, \dots, s_l\}$$

$$\text{P-SEQ: } \frac{\langle p_1, \sigma \rangle \Downarrow^{\vec{c}} \sigma'' \quad \langle p_2, \sigma'' \rangle \Downarrow^{\vec{c}} \sigma'}{\langle p_1; p_2, \sigma \rangle \Downarrow^{\vec{c}} \sigma'}$$

**Figure 1.3:**  $\text{SVCODE}_0$  semantics

The rule P-EMPTY is trivial, empty program doing nothing on the store.



The rule P-XDUCER adds the store a new stream binding where the bound vector is generated by a specific Xducer. The detailed semantics of Xducers are defined in the next subsection.

The rules P-WC-EMP and P-WC-NONEMP together show two possibilities for interpreting a `WithCtrl` instruction:

- if the new control stream  $s_c$  and the streams in  $S_{in}$ , which includes the free variables of  $p_1$ , are all empty, then just bind empty vectors to the stream ids in  $S_{out}$ , which are part of the defined streams of  $p_1$ .
- otherwise execute the code of  $p_1$  as usual under the new control stream, ending in the store  $\sigma''$ ; then copy the bindings of  $S_{out}$  from  $\sigma''$  to the initial store.

### 1.2.3 Xducer semantics

The semantics of Xducers are abstracted into two levels: the *general* level and the *block* level. The general level summarizes the common property that all Xducers share, and the block level describes the specific behavior of each Xducer.

Figure 1.4 shows the semantics at the general level.

**Judgment**  $\boxed{\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\vec{c}} \vec{a}}$

$$\text{P-X-LOOP} : \frac{\psi(\vec{a}_{11}, \dots, \vec{a}_{k1}) \Downarrow \vec{a}_{01} \quad \psi(\vec{a}_{12}, \dots, \vec{a}_{k2}) \Downarrow^{\vec{c}_0} \vec{a}_{02}}{\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{(\langle \rangle | \vec{c}_0)} \vec{a}_0} ((\vec{a}_{i1} ++ \vec{a}_{i2} = \vec{a}_i)_{i=0}^k)$$

$$\text{P-X-TERM} : \frac{}{\psi(\langle \rangle_1, \dots, \langle \rangle_k) \Downarrow^{\langle \rangle} \langle \rangle} 1$$

**Figure 1.4:** General semantics of SVCODE<sub>0</sub> Xducers

There are only two rules for the general semantics. They together say that the output stream is computed in a “loop” fashion, where the iteration uses specific block semantics of the Xducer and the number of iteration is the unary number that the control stream represents, i.e., the length of the control stream. In the parallel setting, we prefer to call this iteration a *block*. Recall the control stream is a representation of the parallel degree of the computation, then a block consumes exact one degree. It is worth noting that all these blocks are data-independent, which means they can be executed in parallel. So the control stream indeed carries the theoretical maximum number of processors we need to execute the computation most efficiently, if the computation within the block cannot be parallelized further.

After abstracting the general semantics, the remaining work of formalizing the specific semantics of Xducers within a block becomes relatively clear and simple. The block semantics are defined in Figure 1.5.

<sup>1</sup>For notational convenience, in this thesis we add subscripts to a sequence of constants, such as  $\langle \rangle, \mathbf{F}, 1$ , to denote the total number of these constants.

**Judgment**  $\boxed{\psi(\vec{a}_1, \dots, \vec{a}_k) \downarrow \vec{a}}$

$$\begin{array}{l}
\text{P-X-CONST: } \frac{}{\text{Const}_a() \downarrow \langle a \rangle} \qquad \text{P-X-TOFLAGS: } \frac{}{\text{ToFlags}(\langle n \rangle) \downarrow \langle F_1, \dots, F_n, T \rangle} \\
\\
\text{P-X-MAPTWO: } \frac{}{\text{MapTwo}_{\oplus}(\langle n_1 \rangle, \langle n_2 \rangle) \downarrow \langle n_3 \rangle} \quad (n_3 = n_1 \oplus n_2) \\
\\
\text{P-X-USUMF: } \frac{\text{Usum}(\vec{b}) \downarrow \vec{a}}{\text{Usum}(\langle F | \vec{b} \rangle) \downarrow \langle () | \vec{a} \rangle} \qquad \text{P-X-USUMT: } \frac{}{\text{Usum}(\langle T \rangle) \downarrow \langle \rangle} \\
\\
\text{P-X-SCANF: } \frac{\text{ScanPlus}_{n_0+n}(\vec{b}, \vec{a}) \downarrow \vec{a}'}{\text{ScanPlus}_{n_0}(\langle F | \vec{b} \rangle, \langle n | \vec{a} \rangle) \downarrow \langle n_0 | \vec{a}' \rangle} \\
\\
\text{P-SCANT: } \frac{}{\text{ScanPlus}_{n_0}(\langle T \rangle, \langle \rangle) \downarrow \langle \rangle} \\
\\
\text{P-X-DISTRF: } \frac{\text{Distr}(\vec{b}, \langle n \rangle) \downarrow \vec{a}}{\text{Distr}(\langle F | \vec{b} \rangle, \langle n \rangle) \downarrow \langle n | \vec{a} \rangle} \qquad \text{P-X-DISTR T: } \frac{}{\text{Distr}(\langle T \rangle, \langle n \rangle) \downarrow \langle \rangle}
\end{array}$$

**Figure 1.5:** Block semantics of Xducers

As have discussed before, we consider a block as the minimum computing unit assigned to a single processor. This is reasonable for Xducers such as **Const<sub>a</sub>** and **MapTwo<sub>⊕</sub>**, because they are already sequential at the block level.

However, some other Xducers, such as **Usum**, can be parallelized further inside a block. As we extend the language with more Xducers, we could find that computations on unary numbers within blocks are common, which is mainly due to the value representation strategy we use, but also more difficult to be regularized. For the scope of this thesis, the block semantics we have shown are already relatively clear and simple enough to reason about, and the unary level parallelism can be investigated in future work.

#### 1.2.4 SVCODE<sub>0</sub> determinism

We now present how we prove a well-formed SVCODE<sub>0</sub> program is deterministic.

**Definition 1.5 (Stream prefix).**  $\vec{a}$  is a prefix of  $\vec{a}'$ , written  $\vec{a} \sqsubseteq \vec{a}'$ , if one of the following rules applies:

$$\begin{array}{l}
\text{Judgment } \boxed{\vec{a} \sqsubseteq \vec{a}'} \\
\\
\text{I-EMP: } \frac{}{\langle \rangle \sqsubseteq \vec{a}} \qquad \text{I-NONEMP: } \frac{\vec{a} \sqsubseteq \vec{a}'}{\langle a_0 \mid \vec{a} \rangle \sqsubseteq \langle a_0 \mid \vec{a}' \rangle}
\end{array}$$

**Lemma 1.6.** *If  $\vec{a}_1 ++ \vec{a}_2 = \vec{a}$ , then  $\vec{a}_1 \sqsubseteq \vec{a}$ .*

*Proof.* The proof is straightforward by induction on  $\vec{a}_1$ : case  $\vec{a}_1 = \langle \rangle$  and case  $\vec{a}_1 = \langle a_0 \mid \vec{a}'_1 \rangle$  for some  $\vec{a}'_1$ . ■

The following lemma says that a Xducer always knows how many elements it should consume and produce when it reads one element from the control stream, i.e., for one block, and these numbers are fixed in any block for the same Xducer.

**Lemma 1.7 (Blocks are self-delimiting).** *If*

- (i)  $(\vec{a}'_i \sqsubseteq \vec{a}_i \text{ by some derivation } \mathcal{I}_i)_{i=1}^k$  and  $\psi(\vec{a}'_1, \dots, \vec{a}'_k) \downarrow \vec{a}'$  by some  $\mathcal{P}$ ,
- (ii)  $(\vec{a}''_i \sqsubseteq \vec{a}_i \text{ by some derivation } \mathcal{I}'_i)_{i=1}^k$  and  $\psi(\vec{a}''_1, \dots, \vec{a}''_k) \downarrow \vec{a}''$  by some  $\mathcal{P}'$ .

then

- (i)  $(\vec{a}'_i = \vec{a}''_i)_{i=1}^k$
- (ii)  $\vec{a}' = \vec{a}''$ .

*Proof.* The proof can be done by induction on  $\mathcal{P}$ . We show three cases P-X-TOFLAGS, P-X-SCANT and P-X-SCANF here; the others are analogous.

- Case  $\mathcal{P}$  uses P-X-TOFLAGS.

Then

$$\mathcal{P} = \frac{}{\text{ToFlags}(\langle n_1 \rangle) \downarrow \langle F_1, \dots, F_{n_1}, T \rangle}$$

and

$$\mathcal{P}' = \frac{}{\text{ToFlags}(\langle n_2 \rangle) \downarrow \langle F_1, \dots, F_{n_2}, T \rangle}$$

so  $k=1$ ,  $\vec{a}'_1 = \langle n_1 \rangle$ ,  $\vec{a}' = \langle F_1, \dots, F_{n_1}, T \rangle$ , and  $\vec{a}''_1 = \langle n_2 \rangle$ ,  $\vec{a}'' = \langle F_1, \dots, F_{n_2}, T \rangle$ .

Since both  $\vec{a}'_1$  and  $\vec{a}''_1$  are nonempty,  $\mathcal{I}_1$  and  $\mathcal{I}'_1$  must both use the rule I-NONEMP, which implies  $n_1 = n_2$ . Then it is clear that  $\vec{a}'_1 = \vec{a}''_1$  and  $\vec{a}' = \vec{a}''$ , as required.

- Case  $\mathcal{P}$  uses P-X-SCANT.

Then

$$\mathcal{P} = \frac{}{\text{ScanPlus}_{n_0}(\langle T \rangle, \langle \rangle) \downarrow \langle \rangle}$$

so  $k=2$ ,  $\vec{a}'_1 = \langle T \rangle$ . Since  $\vec{a}'_1$  is nonempty, then  $\mathcal{I}_1$  must use I-NONEMP, which implies the first element of  $\vec{a}_1$  is T.

There are two possibilities for  $\mathcal{P}'$ :

- Subcase  $\mathcal{P}'$  uses P-X-SCANF.

This subcase is impossible, because it requires  $\vec{a}_1$  starts with a F, which is contradictory to what we already know.

- Subcase  $\mathcal{P}'$  uses P-X-SCANT.

Then

$$\mathcal{P}' = \frac{}{\text{ScanPlus}_{n_0}(\langle T \rangle, \langle \rangle) \downarrow \langle \rangle}$$

So  $\vec{a}''_1 = \langle T \rangle = \vec{a}'_1$ ,  $\vec{a}''_2 = \langle \rangle = \vec{a}'_2$ , and  $\vec{a}'' = \langle \rangle = \vec{a}'$ , as required.

- Case  $\mathcal{P}$  uses P-X-SCANF.

Then

$$\mathcal{P} = \frac{\mathcal{P}_0 \quad \text{ScanPlus}_{n_0+n}(\vec{a}'_{10}, \vec{a}'_{20}) \downarrow \vec{a}'_0}{\text{ScanPlus}_{n_0}(\langle \mathbf{F} | \vec{a}'_{10} \rangle, \langle n | \vec{a}'_{20} \rangle) \downarrow \langle n_0 | \vec{a}'_0 \rangle}$$

So  $k=2$ ,  $\vec{a}'_1 = \langle \mathbf{F} | \vec{a}'_{10} \rangle$ , and  $\vec{a}'_2 = \langle n | \vec{a}'_{20} \rangle$ .  $\mathcal{I}_1$  must use I-NONEMP, which implies the first element of  $\vec{a}'_1$  is F. So  $\vec{a}'_1 = \langle \mathbf{F} | \vec{a}'_{10} \rangle$  for some  $\vec{a}'_{10}$ . By the rule I-NONEMP, we have

$$\frac{\vec{a}'_{10} \sqsubseteq \vec{a}_{10}}{\langle \mathbf{F} | \vec{a}'_{10} \rangle \sqsubseteq \langle \mathbf{F} | \vec{a}_{10} \rangle}$$

Similarly, we can assume  $\vec{a}'_2 = \langle n | \vec{a}'_{20} \rangle$ , and we must have  $\vec{a}'_{20} \sqsubseteq \vec{a}_{20}$ . Thus,

$$(\vec{a}'_{i0} \sqsubseteq \vec{a}_{i0})_{i=1}^2 \quad (1.1)$$

There are two possibilities for  $\mathcal{P}'$ :

- Subcase  $\mathcal{P}'$  uses P-X-SCANT.

This subcase is impossible, because  $\vec{a}_1$  does not start with a T.

- Subcase  $\mathcal{P}'$  uses P-X-SCANF.

Then

$$\mathcal{P}' = \frac{\mathcal{P}'_0 \quad \text{ScanPlus}_{n_0+n}(\vec{a}''_{10}, \vec{a}''_{20}) \downarrow \vec{a}''_0}{\text{ScanPlus}_{n_0}(\langle \mathbf{F} | \vec{a}''_{10} \rangle, \langle n | \vec{a}''_{20} \rangle) \downarrow \langle n_0 | \vec{a}''_0 \rangle}$$

so  $\vec{a}''_1 = \langle \mathbf{F} | \vec{a}''_{10} \rangle$ ,  $\vec{a}''_2 = \langle n | \vec{a}''_{20} \rangle$ ,  $\vec{a}'' = \langle n_0 | \vec{a}''_0 \rangle$ , and it is easy to show

$$(\vec{a}''_{i0} \sqsubseteq \vec{a}_{i0})_{i=1}^2 \quad (1.2)$$

Here we first prove the following inner lemma:

if  $(\vec{a}'_{i0} \sqsubseteq \vec{a}_{i0})_{i=1}^2$  and  $\text{ScanPlus}_{n_0}(\vec{a}'_{10}, \vec{a}'_{20}) \downarrow \vec{a}'$  by some derivation  $\mathcal{P}_0$ ,  $(\vec{a}''_{i0} \sqsubseteq \vec{a}_{i0})_{i=1}^2$  and  $\text{ScanPlus}_{n_0}(\vec{a}''_{10}, \vec{a}''_{20}) \downarrow \vec{a}''$ , then  $\vec{a}'_{10} = \vec{a}''_{10}$ ,  $\vec{a}'_{20} = \vec{a}''_{20}$ , and  $\vec{a}' = \vec{a}''$ .

The proof is by induction on  $\mathcal{P}_0$ . There are two subcases: for the case  $\mathcal{P}_0$  uses P-X-SCANT, the proof is analogous to the outer proof case  $\mathcal{P}$  uses P-X-SCANT; for the case  $\mathcal{P}_0$  uses P-X-SCANF, the proof can be done by the inner IH.

Then, by this inner lemma on (1.1) with  $\mathcal{P}_0$ , (1.2),  $\mathcal{P}'_0$ , we get  $(\vec{a}'_{i0} = \vec{a}''_{i0})_{i=1}^2$ , and  $\vec{a}'_0 = \vec{a}''_0$ .

Thus  $\langle \mathbf{F} | \vec{a}'_{10} \rangle = \langle \mathbf{F} | \vec{a}''_{10} \rangle$ , i.e.,  $\vec{a}'_1 = \vec{a}''_1$ . Likewise,  $\vec{a}'_2 = \langle n | \vec{a}'_{20} \rangle = \langle n | \vec{a}''_{20} \rangle = \vec{a}''_2$ , and  $\vec{a}' = \langle n_0 | \vec{a}'_0 \rangle = \langle n_0 | \vec{a}''_0 \rangle = \vec{a}''$ , as required. ■

**Lemma 1.8 (Xducer determinism).** *If  $\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\vec{c}} \vec{a}_0$  by some derivation  $\mathcal{P}$ , and  $\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\vec{c}} \vec{a}'_0$  by some derivation  $\mathcal{P}'$ , then  $\vec{a}_0 = \vec{a}'_0$ .*

*Proof.* The proof is by induction on the structure of  $\vec{c}$ .

- Case  $\vec{c} = \langle \rangle$

Then both  $\mathcal{P}$  and  $\mathcal{P}'$  must use P-X-TERMI:

$$\mathcal{P} = \mathcal{P}' = \frac{}{\psi(\langle \rangle_1, \dots, \langle \rangle_k) \Downarrow^{\langle \rangle} \langle \rangle}$$

so  $\vec{a}_0 = \vec{a}'_0 = \langle \rangle$ , as required.

- Case  $\vec{c} = \langle () | \vec{c}_0 \rangle$ .  
 $\mathcal{P}$  must use P-X-LOOP:

$$\mathcal{P} = \frac{\overset{\mathcal{P}_1}{\psi(\vec{a}_{11}, \dots, \vec{a}_{k1}) \downarrow \vec{a}_{01}} \quad \overset{\mathcal{P}_2}{\psi(\vec{a}_{12}, \dots, \vec{a}_{k2}) \Downarrow^{\vec{c}_0} \vec{a}_{02}}}{\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\langle () | \vec{c}_0 \rangle} \vec{a}_0}$$

where

$$(\vec{a}_{i1} ++ \vec{a}_{i2} = \vec{a}_i)_{i=1}^k \quad (1.3)$$

$$\vec{a}_{01} ++ \vec{a}_{02} = \vec{a}_0 \quad (1.4)$$

Similarly,

$$\mathcal{P}' = \frac{\overset{\mathcal{P}'_1}{\psi(\vec{a}'_{11}, \dots, \vec{a}'_{k1}) \downarrow \vec{a}'_{01}} \quad \overset{\mathcal{P}'_2}{\psi(\vec{a}'_{12}, \dots, \vec{a}'_{k2}) \Downarrow^{\vec{c}_0} \vec{a}'_{02}}}{\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\langle () | \vec{c}_0 \rangle} \vec{a}'_0}$$

where

$$(\vec{a}'_{i1} ++ \vec{a}'_{i2} = \vec{a}_i)_{i=1}^k \quad (1.5)$$

$$\vec{a}'_{01} ++ \vec{a}'_{02} = \vec{a}'_0 \quad (1.6)$$

Using Lemma 1.6 on each of the  $k$  equations of (1.3), we have

$$(\vec{a}_{i1} \sqsubseteq \vec{a}_i)_{i=1}^k \quad (1.7)$$

Analogously, from (1.5),

$$(\vec{a}'_{i1} \sqsubseteq \vec{a}_i)_{i=1}^k \quad (1.8)$$

By Lemma 1.7 on (1.7) with  $\mathcal{P}_1$ , (1.8),  $\mathcal{P}'_1$ , we get

$$(\vec{a}_{i1} = \vec{a}'_{i1})_{i=1}^k \quad (1.9)$$

$$\vec{a}_{01} = \vec{a}'_{01} \quad (1.10)$$

It is easy to show that from (1.3), (1.5) and (1.9) we can get

$$(\vec{a}_{i2} = \vec{a}'_{i2})_{i=1}^k \quad (1.11)$$

Then by IH on  $\mathcal{P}_2$  with  $\mathcal{P}'_2$ , we obtain  $\vec{a}_{02} = \vec{a}'_{02}$ .

Therefore, with (1.4), (1.6), (1.10), we obtain  $\vec{a}_0 = \vec{a}_{01} ++ \vec{a}_{02} = \vec{a}'_{01} ++ \vec{a}'_{02} = \vec{a}'_0$ , as required. ■

**Theorem 1.9 (SVCODE<sub>0</sub> determinism).** *If  $\langle p, \sigma \rangle \Downarrow^{\vec{c}} \sigma'$  (by some derivation  $\mathcal{P}$ ) and  $\langle p, \sigma \rangle \Downarrow^{\vec{c}} \sigma''$  (by some derivation  $\mathcal{P}'$ ), then  $\sigma' = \sigma''$ .*

*Proof.* The proof is by induction on the syntax of  $p$ . There are four cases: the case for  $p = \epsilon$  is trivial; with the help of Lemma 1.8, the case for  $p = s := \psi(s_1, \dots, s_k)$  is immediate; proof of  $p = p_1; p_2$  can be done by IH; the only interesting case is  $p = S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1)$ .

- Case  $p = S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1)$ .  
 Assume  $S_{out} = \{s_1, \dots, s_l\}$ . There are two subcases by induction on  $\sigma(s_c)$ :

– Subcase  $\sigma(s_c) = \langle \rangle$ .

Then  $\mathcal{P}$  and  $\mathcal{P}'$  must both use P-WC-EMP, and they must be identical:

$$\mathcal{P} = \mathcal{P}' = \frac{}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1), \sigma \rangle \Downarrow^{\vec{c}} \sigma[(s_i \mapsto \langle \rangle)_{i=1}^l]}$$

with  $\forall s \in S_{in}. \sigma(s) = \langle \rangle$ . So  $\sigma' = \sigma'' = \sigma[(s_i \mapsto \langle \rangle)_{i=1}^l]$ , as required.

– Subcase  $\sigma(s_c) \neq \langle \rangle$ .

Then we must have

$$\mathcal{P} = \frac{\mathcal{P}_1 \quad \langle p_1, \sigma \rangle \Downarrow^{\vec{c}_1} \sigma_1}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1), \sigma \rangle \Downarrow^{\vec{c}} \sigma[(s_i \mapsto \sigma_1(s_i))_{i=1}^l]}$$

Also, we have

$$\mathcal{P}' = \frac{\mathcal{P}'_1 \quad \langle p_1, \sigma \rangle \Downarrow^{\vec{c}_1} \sigma'_1}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_1), \sigma \rangle \Downarrow^{\vec{c}} \sigma[(s_i \mapsto \sigma'_1(s_i))_{i=1}^l]}$$

So  $\sigma' = \sigma[(s_i \mapsto \sigma_1(s_i))_{i=1}^l]$ , and  $\sigma'' = \sigma[(s_i \mapsto \sigma'_1(s_i))_{i=1}^l]$ .

By IH on  $\mathcal{P}_1$  and  $\mathcal{P}'_1$ , we obtain

$$\sigma_1 = \sigma'_1$$

Then it is clear that  $\sigma' = \sigma''$ , as required. ■

### 1.3 Translation

(1) Since we do not have pairs, the stream tree type will be :

$$\mathbf{STree} \ni st ::= s \mid (st_1, s)$$

(2) Translation environment:

$$\delta = [x_1 \mapsto st_1, \dots, x_i \mapsto st_i]$$

The translation judgments shown below can be read as: “ in the environment  $\delta$ , the expression  $e$  (or the function call  $\phi(st_1, \dots, st_k)$ ) will be translated to an  $\text{SVCODE}_0$  program  $p$ , whose stream ids starts from  $s_0$  and ends at  $s_1 - 1$  (both included), and the evaluation result is represented as the streams in  $st$ ”.

• Expression translation rules:

$$\textbf{Judgment} \quad \boxed{\delta \vdash e \Rightarrow_{s_1}^{s_0} (p, st)}$$

$$\frac{}{\delta \vdash x \Rightarrow_{s_0}^{s_0} (\epsilon, st)} (\delta(x) = st)$$

$$\begin{array}{c}
\frac{\delta \vdash e_1 \Rightarrow_{s_0'}^{s_0} (p_1, st_1) \quad \delta[x \mapsto st_1] \vdash e_2 \Rightarrow_{s_1'}^{s_0'} (p_2, st)}{\delta \vdash \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2 \Rightarrow_{s_1}^{s_0} (p_1; p_2, st)} \\
\\
\frac{\phi(st_1, \dots, st_k) \Rightarrow_{s_1}^{s_0} (p, st)}{\delta \vdash \phi(x_1, \dots, x_k) \Rightarrow_{s_1}^{s_0} (p, st)} ((\delta(x_i) = st_i)_{i=1}^k) \\
\\
\frac{[x \mapsto st_1, (x_i \mapsto s_i)_{i=1}^j] \vdash e \Rightarrow_{s_1}^{s_0+1+j} (p_1, st)}{\delta \vdash \{e : x \ \mathbf{in} \ y \ \mathbf{using} \ x_1, \dots, x_j\} \Rightarrow_{s_1}^{s_0} (p, (st, s_b))} \left( \begin{array}{l} \delta(y) = (st_1, s_b) \\ (\delta(x_i) = s_i')_{i=1}^j \\ p = s_0 := \mathbf{Usum}(s_b); \\ (s_i := \mathbf{Distr}(s_b, s_i'))_{i=1}^j \\ S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1) \\ S_{in} = \mathbf{fv}(p_1) \\ S_{out} = \overline{st} \cap \mathbf{dv}(p_1) \\ s_{i+1} = s_i + 1, \forall i \in \{0, \dots, j-1\} \end{array} \right)
\end{array}$$

- Built-in function translation rules:

**Judgment**  $\boxed{\phi(st_1, \dots, st_k) \Rightarrow_{s_1}^{s_0} (p, st)}$

$$\frac{}{\mathbf{const}_n() \Rightarrow_{s_0+1}^{s_0} (s_0 := \mathbf{Const}_n(), s_0)}$$

$$\frac{}{\mathbf{plus}(s_1, s_2) \Rightarrow_{s_0+1}^{s_0} (s_0 := \mathbf{MapTwo}_+(s_1, s_2), s_0)}$$

$$\frac{}{\mathbf{iota}(s) \Rightarrow_{s_4}^{s_0} (p, (s_3, s_0))} \left( \begin{array}{l} s_{i+1} = s_i + 1, \forall i \in \{0, \dots, 3\} \\ p = s_0 := \mathbf{ToFlags}(s); \\ s_1 := \mathbf{Usum}(s_0); \\ \overline{s_2} := \mathbf{WithCtrl}(s_1, [s_1], s_2 := \mathbf{Const}_1()); \\ s_3 := \mathbf{ScanPlus}_0(s_0, s_2) \end{array} \right)$$

## 1.4 Value representation

- (1) SVCODE<sub>0</sub> values:

$$\mathbf{SvVal} \ni w ::= \vec{a} \mid (w, \vec{b})$$

- (2) We extend the operation ++ from primitive stream concatenation to SVCODE<sub>0</sub> values concatenation:

$$\begin{aligned}
& ++_\tau : \mathbf{SvVal} \rightarrow \mathbf{SvVal} \rightarrow \mathbf{SvVal} \\
& \langle \vec{a}_1, \dots, \vec{a}_i \rangle ++_{\mathbf{int}} \langle \vec{a}'_1, \dots, \vec{a}'_j \rangle = \langle \vec{a}_1, \dots, \vec{a}_i, \vec{a}'_1, \dots, \vec{a}'_j \rangle \\
& (w_1, \vec{b}_1) ++_{\{\tau\}} (w_2, \vec{b}_2) = (w_1 ++_\tau w_2, \vec{b}_1 ++_{\mathbf{int}} \vec{b}_2)
\end{aligned}$$

(3) SVCODE<sub>0</sub> value construction from a stream tree:

$$\begin{aligned}\sigma &: \mathbf{STree} \rightarrow \mathbf{SvVal} \\ \sigma(s) &= \vec{a} \\ \sigma((st, s)) &= (\sigma(st), \sigma(s))\end{aligned}$$

(4) Value representation rules:

• **Judgment**  $\boxed{v \triangleright_{\tau} w}$

$$\frac{}{n \triangleright_{\text{int}} \langle n \rangle} \quad \frac{(v_i \triangleright_{\tau} w_i)_{i=1}^k}{\{v_1, \dots, v_k\} \triangleright_{\{\tau\}} (w, \langle F_1, \dots, F_k, T \rangle)} (w = w_1 ++ \dots ++ w_k)$$

**Lemma 1.10 (Value translation backwards determinism).** *If  $v \triangleright_{\tau} w$ ,  $v' \triangleright_{\tau} w$ , then  $v = v'$ .*

## 1.5 Correctness

### 1.5.1 Definitions

We first define a binary relation  $\overset{S}{\sim}$  on stores to denote that two stores are *similar*: they have identical domains, and their bound values by  $S$  are the same. We call this  $S$  an *overlap* of these two stores.

**Definition 1.11 (Stores similarity).**  $\sigma_1 \overset{S}{\sim} \sigma_2$  iff

- (1)  $\text{dom}(\sigma_1) = \text{dom}(\sigma_2)$
- (2)  $\forall s \in S. \sigma_1(s) = \sigma_2(s)$

According to this definition, it is only meaningful to have  $S \subseteq \text{dom}(\sigma_1)$  ( $= \text{dom}(\sigma_2)$ ). When  $S = \text{dom}(\sigma_1) = \text{dom}(\sigma_2)$ ,  $\sigma_1$  and  $\sigma_2$  are identical. It is easy to show that this relation  $\overset{S}{\sim}$  is symmetric and transitive.

- If  $\sigma_1 \overset{S}{\sim} \sigma_2$ , then  $\sigma_2 \overset{S}{\sim} \sigma_1$ .
- If  $\sigma_1 \overset{S}{\sim} \sigma_2$  and  $\sigma_2 \overset{S}{\sim} \sigma_3$ , then  $\sigma_1 \overset{S}{\sim} \sigma_3$ .

We also define a binary operation  $\overset{S}{\bowtie}$  on stores to denote a kind of special concatenation of two similar stores: the *concatenation* of two similar stores is a new store, in which the bound values by  $S$  are from any of the parameter stores, and the others are the concatenation of the values from the two stores. In other words, a *concatenation* of two similar stores is only a concatenation of the bound values that *maybe* different in these stores.

**Definition 1.12 (Store Concatenation).** For  $\sigma_1 \overset{S}{\sim} \sigma_2$ ,  $\sigma_1 \overset{S}{\bowtie} \sigma_2 = \sigma$  where

$$\sigma(s) = \begin{cases} \sigma_1(s) (= \sigma_2(s)), & s \in S \\ \sigma_1(s) ++ \sigma_2(s), & s \notin S \end{cases}$$

**Lemma 1.13.** *If  $\sigma_1 \overset{S}{\bowtie} \sigma_2 = \sigma$ , then  $\sigma_1 \overset{S}{\sim} \sigma$  and  $\sigma_2 \overset{S}{\sim} \sigma$ .*



This lemma says that the concatenation result of two similar stores is still similar to each of them.

**Lemma 1.14.** *If*

(i)  $\psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\vec{c}_1} \vec{a}_0$  *by some derivation*  $\mathcal{P}_1$

(ii)  $\psi(\vec{a}'_1, \dots, \vec{a}'_k) \Downarrow^{\vec{c}_2} \vec{a}'_0$  *by some*  $\mathcal{P}_2$ ,

*then*  $\psi(\vec{a}_1 ++ \vec{a}'_1, \dots, \vec{a}_k ++ \vec{a}'_k) \Downarrow^{\vec{c} ++ \vec{c}'} \vec{a} ++ \vec{a}'$  *by some*  $\mathcal{P}_3$ .

*Proof.* There are two possibilities:

- Case  $\mathcal{P}_1$  uses P-X-TERMI.

We must have  $(\vec{a}_i = \langle \rangle)_{i=0}^k$  and  $\vec{c}_1 = \langle \rangle$ . Then  $(\vec{a}_i ++ \vec{a}'_i = \vec{a}'_i)_{i=0}^k$ , and  $\vec{c}_1 ++ \vec{c}_2 = \vec{c}_2$ . Take  $\mathcal{P}_3 = \mathcal{P}_2$  and we are done.

- Case  $\mathcal{P}_1$  uses P-X-LOOP.

Assume  $\vec{c}_1 = \langle () | \vec{c}'_1 \rangle$ , then we have

$$\mathcal{P}_1 = \frac{\mathcal{P}_{11} \quad \mathcal{P}_{12}}{\psi(\vec{a}_{11}, \dots, \vec{a}_{k1}) \downarrow \vec{a}_{01} \quad \psi(\vec{a}_{12}, \dots, \vec{a}_{k2}) \Downarrow^{\vec{c}'_1} \vec{a}_{02} \over \psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\langle c_0 | \vec{c}'_1 \rangle} \vec{a}_0}$$

with  $(\vec{a}_i = \vec{a}_{i1} ++ \vec{a}_{i2})_{i=0}^k$ .

By IH on  $\mathcal{P}_{12}$  with  $\mathcal{P}_2$ , we get a derivation  $\mathcal{P}_4$  of

$$\psi(\vec{a}_{12} ++ \vec{a}'_1, \dots, \vec{a}_{k2} ++ \vec{a}'_k) \Downarrow^{\vec{c}'_1 ++ \vec{c}_2} \vec{a}_{02} ++ \vec{a}'_0$$

Then using the rule P-X-LOOP we can build a derivation  $\mathcal{P}_5$  as follows:

$$\frac{\mathcal{P}_{11} \quad \mathcal{P}_4}{\psi(\vec{a}_{11}, \dots, \vec{a}_{k1}) \downarrow \vec{a}_{01} \quad \psi(\vec{a}_{12} ++ \vec{a}'_1, \dots, \vec{a}_{k2} ++ \vec{a}'_k) \Downarrow^{\vec{c}'_1 ++ \vec{c}_2} \vec{a}_{02} ++ \vec{a}'_0 \over \psi(\vec{a}_{11} ++ (\vec{a}_{12} ++ \vec{a}'_1), \dots, \vec{a}_{k1} ++ (\vec{a}_{k2} ++ \vec{a}'_k)) \Downarrow^{\langle () | \vec{c}'_1 ++ \vec{c}_2 \rangle} \vec{a}_{01} ++ (\vec{a}_{02} ++ \vec{a}'_0)}$$

Since it is clear that

$$\forall i \in \{0, \dots, k\}. \vec{a}_{i1} ++ (\vec{a}_{i2} ++ \vec{a}'_i) = (\vec{a}_{i1} ++ \vec{a}_{i2}) ++ \vec{a}'_i = \vec{a}_i ++ \vec{a}'_i$$

$$\langle () | \vec{c}'_1 ++ \vec{c}_2 \rangle = \langle () | \vec{c}'_1 \rangle ++ \vec{c}_2 = \vec{c}_1 ++ \vec{c}_2$$

so take  $\mathcal{P}_3 = \mathcal{P}_5$  and we are done. ■

**Lemma 1.15.** *If*

(i)  $\sigma_1 \stackrel{S}{\sim} \sigma_2$

(ii)  $\langle p, \sigma_1 \rangle \Downarrow^{\vec{c}} \sigma$

(iii)  $\text{fv}(p) \cap S = \emptyset$

(iv)  $\forall s \in \text{fv}(p). \sigma_2(s) = \langle \rangle$

then

$$(v) \langle p, \sigma_1 \overset{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}} \sigma'$$

$$(vi) \forall s' \in \mathbf{dv}(p). \sigma(s') = \sigma'(s')$$

**Lemma 1.16.** *If*

$$(i) \sigma_1 \overset{S}{\sim} \sigma_2$$

$$(ii) \langle p, \sigma_2 \rangle \Downarrow^{\vec{c}} \sigma$$

$$(iii) \mathbf{fv}(p) \cap S = \emptyset$$

$$(iv) \forall s \in \mathbf{fv}(p). \sigma_1(s) = \langle \rangle$$

then

$$(v) \langle p, \sigma_1 \overset{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}} \sigma'$$

$$(vi) \forall s' \in \mathbf{dv}(p). \sigma(s') = \sigma'(s')$$

**Lemma 1.17 (Stores concatenation lemma).** *If*

$$(i) \sigma_1 \overset{S}{\sim} \sigma_2$$

$$(ii) \langle p, \sigma_1 \rangle \Downarrow^{\vec{c}_1} \sigma'_1 \text{ (by some derivation } \mathcal{P}_1)$$

$$(iii) \langle p, \sigma_2 \rangle \Downarrow^{\vec{c}_2} \sigma'_2 \text{ (by some derivation } \mathcal{P}_2)$$

$$(iv) \mathbf{fv}(p) \cap S = \emptyset$$

$$\text{then } \langle p, \sigma_1 \overset{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma'_1 \overset{S}{\bowtie} \sigma'_2 \text{ (by } \mathcal{P}).$$

We need this lemma to prove that the results of single computations inside a comprehension body (i.e.  $p$  in the lemma) can be concatenated to express a parallel computation. From the other direction, we can consider this process as distributing or splitting the computation  $p$  on even smaller degree of parallel computations, in which all the supplier streams, i.e.,  $\mathbf{fv}(p)$ , are splitted to feed the transducers. The splitted parallel degrees are specified by the control streams, i.e.,  $\vec{c}_1$  and  $\vec{c}_2$  in the lemma. Other untouched **SI**ds in all  $\sigma$ s (i.e.,  $S$ ) have no change throughout the process.

*Proof.* By induction on the syntax of  $p$ .

- Case  $p = \epsilon$ .  
 $\mathcal{P}_1$  must be  $\overline{\langle \epsilon, \sigma_1 \rangle \Downarrow^{\vec{c}_1} \sigma_1}$ , and  $\mathcal{P}_2$  must be  $\overline{\langle \epsilon, \sigma_2 \rangle \Downarrow^{\vec{c}_2} \sigma_2}$ .  
 So  $\sigma'_1 = \sigma_1$ , and  $\sigma'_2 = \sigma_2$ , thus  $\sigma'_1 \overset{S}{\bowtie} \sigma'_2 = \sigma_1 \overset{S}{\bowtie} \sigma_2$ .

By P-EMPTY, we take  $\mathcal{P} = \overline{\langle \epsilon, \sigma_1 \overset{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma_1 \overset{S}{\bowtie} \sigma_2}$  and we are done.

- Case  $p = s_l := \psi(s_1, \dots, s_k)$ .  
 $\mathcal{P}_1$  must look like

$$\frac{\mathcal{P}'_1 \quad \psi(\vec{a}_1, \dots, \vec{a}_k) \Downarrow^{\vec{c}_1} \vec{a}}{\langle s_l := \psi(s_1, \dots, s_k), \sigma_1 \rangle \Downarrow^{\vec{c}_1} \sigma_1[s_l \mapsto \vec{a}]}$$

and we have

$$(\sigma_1(s_i) = \vec{a}_i)_{i=1}^k \quad (1.12)$$

Similarly,  $\mathcal{P}_2$  must look like

$$\frac{\mathcal{P}'_2 \quad \psi(\vec{a}'_1, \dots, \vec{a}'_k) \Downarrow^{\vec{c}_2} \vec{a}'}{\langle s_l := \psi(s_1, \dots, s_k), \sigma_2 \rangle \Downarrow^{\vec{c}_2} \sigma_2[s_l \mapsto \vec{a}']}$$

and we have

$$(\sigma_2(s_i) = \vec{a}'_i)_{i=1}^k \quad (1.13)$$

So  $\sigma'_1 = \sigma_1[s_l \mapsto \vec{a}]$ ,  $\sigma'_2 = \sigma_2[s_l \mapsto \vec{a}']$ .

From assumption (iv) we have  $\mathbf{fv}(s_l := \psi(s_1, \dots, s_k)) \cap S = \emptyset$ , that is,

$$\{s_1, \dots, s_k\} \cap S = \emptyset \quad (1.14)$$

By Lemma 1.14 on  $\mathcal{P}'_1$ ,  $\mathcal{P}'_2$ , we get a derivation  $\mathcal{P}'$  of

$$\psi(\vec{a}_1 ++ \vec{a}'_1, \dots, \vec{a}_k ++ \vec{a}'_k) \Downarrow^{\vec{c}_1 ++ \vec{c}_2} \vec{a} ++ \vec{a}'$$

Since  $\sigma_1 \stackrel{S}{\sim} \sigma_2$ , with (1.12), (1.13) and (1.14), by Definition 1.12 we have

$$\forall i \in \{1, \dots, k\}. \sigma_1 \stackrel{S}{\bowtie} \sigma_2(s_i) = \sigma_1(s_i) ++ \sigma_2(s_i) = \vec{a}_i ++ \vec{a}'_i \quad (1.15)$$

Also, it is easy to prove that  $\sigma_1[s_l \mapsto \vec{a}] \stackrel{S}{\bowtie} \sigma_2[s_l \mapsto \vec{a}'] \stackrel{S}{\sim} \sigma_1 \stackrel{S}{\bowtie} \sigma_2[s_l \mapsto \vec{a} ++ \vec{a}']$  and

$$\sigma_1[s_l \mapsto \vec{a}] \stackrel{S}{\bowtie} \sigma_2[s_l \mapsto \vec{a}'] = \sigma_1 \stackrel{S}{\bowtie} \sigma_2[s_l \mapsto \vec{a} ++ \vec{a}'] \quad (1.16)$$

Using the rule P-XDUCER with (1.15), we can build  $\mathcal{P}''$  as follows

$$\frac{\mathcal{P}' \quad \psi(\vec{a}_1 ++ \vec{a}'_1, \dots, \vec{a}_k ++ \vec{a}'_k) \Downarrow^{\vec{c}_1 ++ \vec{c}_2} \vec{a} ++ \vec{a}'}{\left\langle s_l := \psi(s_1, \dots, s_k), \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \right\rangle \Downarrow^{\vec{c}_1 ++ \vec{c}_2} \sigma_1 \stackrel{S}{\bowtie} \sigma_2[s_l \mapsto \vec{a} ++ \vec{a}']}$$

Replacing  $\sigma_1 \stackrel{S}{\bowtie} \sigma_2[s_l \mapsto \vec{a} ++ \vec{a}']$  in  $\mathcal{P}''$  with the left-hand side of (1.16) gives us  $\mathcal{P}$

$$\frac{\mathcal{P}' \quad \psi(\vec{a}_1 ++ \vec{a}'_1, \dots, \vec{a}_k ++ \vec{a}'_k) \Downarrow^{\vec{c}_1 ++ \vec{c}_2} \vec{a} ++ \vec{a}'}{\left\langle s_l := \psi(s_1, \dots, s_k), \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \right\rangle \Downarrow^{\vec{c}_1 ++ \vec{c}_2} \sigma_1[s_l \mapsto \vec{a}] \stackrel{S}{\bowtie} \sigma_2[s_l \mapsto \vec{a}']}$$

as required.

- Case  $p = S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0)$  where

$$\text{fv}(p_0) \subseteq S_{in} \quad (1.17)$$

$$S_{out} \subseteq \text{dv}(p_0) \quad (1.18)$$

From the assumption (iv), we have

$$\begin{aligned} \text{fv}(S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0)) \cap S &= \emptyset \\ (\{s_c\} \cup S_{in}) \cap S &= \emptyset \quad (\text{by definition of } \text{fv}()) \end{aligned}$$

thus

$$\{s_c\} \cap S = \emptyset \quad (1.19)$$

$$S_{in} \cap S = \emptyset \quad (1.20)$$

Since (1.17) with (1.20), we also have

$$\text{fv}(p_0) \cap S = \emptyset \quad (1.21)$$

Assume  $S_{out} = [s_1, \dots, s_j]$ .

There are four possibilities:

- Subcase both  $\mathcal{P}_1$  and  $\mathcal{P}_2$  use P-WC-EMP.

So  $\mathcal{P}_1$  must look like

$$\overline{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \rangle \Downarrow^{\tilde{c}_1} \sigma_1[s_1 \mapsto \langle \rangle, \dots, s_j \mapsto \langle \rangle]}$$

and we have

$$\forall s \in \{s_c\} \cup S_{in}. \sigma_1(s) = \langle \rangle \quad (1.22)$$

thus

$$\sigma_1(s_c) = \langle \rangle \quad (1.23)$$

$$\forall s \in \text{fv}(p_0). \sigma_1(s) = \langle \rangle \quad (1.24)$$

Similarly,  $\mathcal{P}_2$  must look like

$$\overline{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_2 \rangle \Downarrow^{\tilde{c}_2} \sigma_2[s_1 \mapsto \langle \rangle, \dots, s_j \mapsto \langle \rangle]}$$

and we have

$$\forall s \in \{s_c\} \cup S_{in}. \sigma_2(s) = \langle \rangle \quad (1.25)$$

thus

$$\sigma_2(s_c) = \langle \rangle \quad (1.26)$$

$$\forall s \in \text{fv}(p_0). \sigma_2(s) = \langle \rangle \quad (1.27)$$

So  $\sigma'_1 = \sigma_1[(s_i \mapsto \langle \rangle)_{i=1}^j]$ ,  $\sigma'_2 = \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]$ .

Since  $\sigma_1 \stackrel{S}{\sim} \sigma_2$ , by Definition 1.12 with (1.19), (1.20), and (1.22), (1.25), we have

$$\forall s \in \{s_c\} \cup S_{in}. \sigma_1 \stackrel{S}{\bowtie} \sigma_2(s) = \sigma_1(s) ++ \sigma_2(s) = \langle \rangle \quad (1.28)$$

Also, it is easy to show that  $\sigma_1[(s_i \mapsto \langle \rangle)_{i=1}^j] \stackrel{S}{\sim} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]$  and

$$\sigma_1[(s_i \mapsto \langle \rangle)_{i=1}^j] \stackrel{S}{\bowtie} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j] = \sigma_1 \stackrel{S}{\bowtie} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j] \quad (1.29)$$

Using P-WC-EMP with (1.28), we build  $\mathcal{P}'$  as follows

$$\frac{}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} (\sigma_1 \stackrel{S}{\bowtie} \sigma_2)[(s_i \mapsto \langle \rangle)_{i=1}^j]}$$

Then replcaing  $\sigma_1 \stackrel{S}{\bowtie} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]$  in  $\mathcal{P}'$  with the left-hand side of (1.29) gives us  $\mathcal{P}$  of

$$\frac{}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma_1[(s_i \mapsto \langle \rangle)_{i=1}^j] \stackrel{S}{\bowtie} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]} \text{ as required.}$$

- Subcase  $\mathcal{P}_1$  uses P-WC-NOMEMP,  $\mathcal{P}_2$  uses P-WC-EMP.  
 $\mathcal{P}_1$  must look like

$$\frac{\mathcal{P}'_1 \quad \langle p_0, \sigma_1 \rangle \Downarrow^{\vec{c}_1} \sigma'_1}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \rangle \Downarrow^{\vec{c}_1} \sigma_1[(s_i \mapsto \sigma'_1(s_i))_{i=1}^j]}$$

and we have

$$\sigma_1(s_c) = \vec{c}_1 = \langle () | \dots \rangle \quad (1.30)$$

$\mathcal{P}_2$  must look like

$$\frac{}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_2 \rangle \Downarrow^{\vec{c}_2} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]}$$

and we have  $\forall s \in \{s_c\} \cup S_{in}. \sigma_2(s) = \langle \rangle$  thus

$$\sigma_2(s_c) = \langle \rangle \quad (1.31)$$

$$\forall s \in \text{fv}(p_0). \sigma_2(s) = \langle \rangle \quad (1.32)$$

So  $\sigma'_1 = \sigma_1[(s_i \mapsto \sigma''_1(s_i))_{i=1}^j]$ ,  $\sigma'_2 = \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]$ .

By Lemma 1.15 on  $\sigma_1 \stackrel{S}{\sim} \sigma_2$  with  $\mathcal{P}'_1$ , (1.32), (1.21), we obtain a derivation  $\mathcal{P}_0$  of

$$\langle p_0, \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1} \sigma_0$$

for some  $\sigma_0$ , and

$$\forall s \in \text{dv}(p_0). \sigma_0(s) = \sigma''_1(s),$$

thus, with (1.18), we have

$$(\sigma_0(s_i) = \sigma''_1(s_i))_{i=1}^j \quad (1.33)$$

Since  $\sigma_1 \stackrel{S}{\sim} \sigma_2$ , by Definition 1.12 with (1.30), (1.31), we have

$$\sigma_1 \stackrel{S}{\bowtie} \sigma_2(s_c) = \sigma_1(s_c) ++ \sigma_2(s_c) = \vec{c}_1 = \langle () | \dots \rangle \quad (1.34)$$

and it is also easy to prove  $\sigma_1[(s_i \mapsto \sigma_1''(s_i))_{i=1}^j] \stackrel{S}{\sim} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]$  and

$$\sigma_1[(s_i \mapsto \sigma_1''(s_i))_{i=1}^j] \stackrel{S}{\bowtie} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j] = \sigma_1 \stackrel{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma_1''(s_i))_{i=1}^j] \quad (1.35)$$

Using the rule P-WC-NONEMP with (1.34) we can build a derivation  $\mathcal{P}'$  as follows

$$\frac{\begin{array}{c} \mathcal{P}_0 \\ \langle p_0, \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1} \sigma_0 \end{array}}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} (\sigma_1 \stackrel{S}{\bowtie} \sigma_2)[(s_i \mapsto \sigma_0(s_i))_{i=1}^j]}$$

With (1.33), we replace  $\sigma_0(s_i)$  with  $\sigma_1''(s_i)$  for  $\forall i \in \{1, \dots, j\}$  in  $\mathcal{P}'$ , obtaining

$$\frac{\begin{array}{c} \mathcal{P}_0 \\ \langle p_0, \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1} \sigma_0 \end{array}}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} (\sigma_1 \stackrel{S}{\bowtie} \sigma_2)[(s_i \mapsto \sigma_1''(s_i))_{i=1}^j]}$$

Then replacing  $(\sigma_1 \stackrel{S}{\bowtie} \sigma_2)[(s_i \mapsto \sigma_1''(s_i))_{i=1}^j]$  in  $\mathcal{P}'$  with the left-hand side of (1.35), we get  $\mathcal{P}$  of

$$\frac{\begin{array}{c} \mathcal{P}_0 \\ \langle p_0, \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1} \sigma_0 \end{array}}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \stackrel{S}{\bowtie} \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma_1[(s_i \mapsto \sigma_1''(s_i))_{i=1}^j] \stackrel{S}{\bowtie} \sigma_2[(s_i \mapsto \langle \rangle)_{i=1}^j]} \text{ as required.}$$

- Subcase  $\mathcal{P}_1$  uses P-WC-EMP and  $\mathcal{P}_2$  uses P-WC-NONEMP.  
This subcase is symmetric to the second one, so the proof is analogous except that this subcase uses Lemma 1.16 rather than Lemma 1.15.
- Subcase both  $\mathcal{P}_1$  and  $\mathcal{P}_2$  use P-WC-NONEMP.  
 $\mathcal{P}_1$  must look like

$$\frac{\begin{array}{c} \mathcal{P}'_1 \\ \langle p_0, \sigma_1 \rangle \Downarrow^{\vec{c}_1} \sigma_1'' \end{array}}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \rangle \Downarrow^{\vec{c}_1} \sigma_1[(s_i \mapsto \sigma_1''(s_i))_{i=1}^j]}$$

and

$$\sigma_1(s_c) = \vec{c}_1 = \langle () | \dots \rangle \quad (1.36)$$

Similarly,  $\mathcal{P}_2$  must look like

$$\frac{\begin{array}{c} \mathcal{P}'_2 \\ \langle p_0, \sigma_2 \rangle \Downarrow^{\vec{c}_2} \sigma_2'' \end{array}}{\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_2 \rangle \Downarrow^{\vec{c}_2} \sigma_2[(s_i \mapsto \sigma_2''(s_i))_{i=1}^j]}$$

and

$$\sigma_2(s_c) = \vec{c}_2 = \langle () | \dots \rangle \quad (1.37)$$

So  $\sigma'_1 = \sigma_1[(s_i \mapsto \sigma''_1(s_i))_{i=1}^j]$ ,  $\sigma'_2 = \sigma_2[(s_i \mapsto \sigma''_2(s_i))_{i=1}^j]$ .

By IH on  $\mathcal{P}'_1, \mathcal{P}'_2$  with (1.21), we get a derivation  $\mathcal{P}_0$  of

$$\left\langle p_0, \sigma_1 \overset{S}{\bowtie} \sigma_2 \right\rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma''_1 \overset{S}{\bowtie} \sigma''_2$$

Since  $\forall i \in \{1, \dots, j\}. s_i \notin S$ , then by Definition 1.12, we know

$$\sigma''_1 \overset{S}{\bowtie} \sigma''_2(s_i) = \sigma''_1(s_i) ++ \sigma''_2(s_i) \quad (1.38)$$

Also, it is easy to show that  $\sigma_1[(s_i \mapsto \sigma''_1(s_i))_{i=1}^j] \overset{S}{\sim} \sigma_2[(s_i \mapsto \sigma''_2(s_i))_{i=1}^j]$ , and

$$\begin{aligned} \sigma_1[(s_i \mapsto \sigma''_1(s_i))_{i=1}^j] \overset{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma''_2(s_i))_{i=1}^j] \\ = \sigma_1 \overset{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma''_1(s_i) ++ \sigma''_2(s_i))_{i=1}^j] \end{aligned} \quad (1.39)$$

thus, with (1.38),

$$\begin{aligned} \sigma_1[(s_i \mapsto \sigma''_1(s_i))_{i=1}^j] \overset{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma''_2(s_i))_{i=1}^j] \\ = \sigma_1 \overset{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma''_1 \overset{S}{\bowtie} \sigma''_2(s_i))_{i=1}^j] \end{aligned} \quad (1.40)$$

Since (1.19) with (1.36), (1.37), we know  $\sigma_1 \overset{S}{\bowtie} \sigma_2(s_c) = \vec{c}_1 ++ \vec{c}_2 = \langle () | \dots \rangle$ , therefore we can use the rule P-WC-NONEMP to build a derivation  $\mathcal{P}'$  as follows:

$$\frac{\mathcal{P}_0 \quad \left\langle p_0, \sigma_1 \overset{S}{\bowtie} \sigma_2 \right\rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma''_1 \overset{S}{\bowtie} \sigma''_2}{\left\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \overset{S}{\bowtie} \sigma_2 \right\rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma_1 \overset{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma''_1 \overset{S}{\bowtie} \sigma''_2(s_i))_{i=1}^j]} .$$

Then replacing  $\sigma_1 \overset{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma''_1 \overset{S}{\bowtie} \sigma''_2(s_i))_{i=1}^j]$  with the left-hand side of (1.40), we obtain  $\mathcal{P}$  of

$$\frac{\mathcal{P}_0 \quad \left\langle p_0, \sigma_1 \overset{S}{\bowtie} \sigma_2 \right\rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma''_1 \overset{S}{\bowtie} \sigma''_2}{\left\langle S_{out} := \text{WithCtrl}(s_c, S_{in}, p_0), \sigma_1 \overset{S}{\bowtie} \sigma_2 \right\rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma_1[(s_i \mapsto \sigma''_1(s_i))_{i=1}^j] \overset{S}{\bowtie} \sigma_2[(s_i \mapsto \sigma''_2(s_i))_{i=1}^j]} .$$

as required.

- Case  $p = p_1; p_2$

We must have

$$\mathcal{P}_1 = \frac{\mathcal{P}'_1 \quad \mathcal{P}''_1}{\left\langle p_1; p_2, \sigma_1 \right\rangle \Downarrow^{\vec{c}_1} \sigma'_1} \quad \text{where} \quad \begin{aligned} \mathcal{P}'_1 &= \left\langle p_1, \sigma_1 \right\rangle \Downarrow^{\vec{c}_1} \sigma''_1 \\ \mathcal{P}''_1 &= \left\langle p_2, \sigma''_1 \right\rangle \Downarrow^{\vec{c}_1} \sigma'_1 \end{aligned}$$

and

$$\mathcal{P}_2 = \frac{\mathcal{P}'_2 \quad \mathcal{P}''_2}{\langle p_1, \sigma_2 \rangle \Downarrow^{\vec{c}_2} \sigma''_2 \quad \langle p_2, \sigma'_2 \rangle \Downarrow^{\vec{c}_2} \sigma'_2 \quad \langle p_1; p_2, \sigma_2 \rangle \Downarrow^{\vec{c}_1} \sigma'_2}$$

Since  $\mathbf{fv}(p_1; p_2) \cap S = \emptyset$ , we have  $(\mathbf{fv}(p_1) \cup \mathbf{fv}(p_2) - \mathbf{dv}(p_1)) \cap S = \emptyset$ , thus

$$\mathbf{fv}(p_1) \cap S = \emptyset \quad (1.41)$$

$$\mathbf{fv}(p_2) \cap S = \emptyset \quad (1.42)$$

By IH on  $\mathcal{P}'_1, \mathcal{P}'_2$ , (1.41), we get  $\mathcal{P}'$  of

$$\langle p_1, \sigma_1 \bowtie^S \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma''_1 \bowtie^S \sigma''_2$$

By Definition 1.12, we must have  $\sigma''_1 \stackrel{S}{\sim} \sigma''_2$ .

Then by IH on  $\mathcal{P}''_1, \mathcal{P}''_2$  with (1.42), we get  $\mathcal{P}''$  of

$$\langle p_2, \sigma''_1 \bowtie^S \sigma''_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma'_1 \bowtie^S \sigma'_2$$

Therefore, we use the rule P-SEQ to build  $\mathcal{P}$  as follows:

$$\frac{\mathcal{P}' \quad \mathcal{P}''}{\langle p_1, \sigma_1 \bowtie^S \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma''_1 \bowtie^S \sigma''_2 \quad \langle p_2, \sigma''_1 \bowtie^S \sigma''_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma'_1 \bowtie^S \sigma'_2 \quad \langle p_1; p_2, \sigma_1 \bowtie^S \sigma_2 \rangle \Downarrow^{\vec{c}_1 + \vec{c}_2} \sigma'_1 \bowtie^S \sigma'_2}$$

and we are done. ■

Let  $\sigma_1 \stackrel{\leq s}{=} \sigma_2$  denote  $\forall s' < s. \sigma_1(s') = \sigma_2(s')$ .

**Lemma 1.18.** *If  $\sigma_1 \stackrel{S_1}{\sim} \sigma'_1$ ,  $\sigma_2 \stackrel{S_2}{\sim} \sigma'_2$ ,  $\sigma_1 \stackrel{\leq s}{=} \sigma_2$ , and  $\sigma'_1 \stackrel{\leq s}{=} \sigma'_2$  then  $\sigma_1 \bowtie^{S_1} \sigma'_1 \stackrel{\leq s}{=} \sigma_2 \bowtie^{S_2} \sigma'_2$ .*

### 1.5.2 Correctness proof

**Lemma 1.19.** *If*

- (i)  $\phi : (\tau_1, \dots, \tau_k) \rightarrow \tau$  (by some derivation  $\mathcal{T}$ )
- (ii)  $\phi(v_1, \dots, v_k) \downarrow v$  (by  $\mathcal{E}$ )
- (iii)  $\phi(st_1, \dots, st_k) \Rightarrow_{s_1}^{s_0} (p, st)$  (by  $\mathcal{C}$ )
- (iv)  $(v_i \triangleright_{\tau_i} \sigma(st_i))_{i=1}^k$
- (v)  $\bigcup_{i=1}^k \mathbf{sids}(st_i) \leq s_0$

then

- (vi)  $\langle p, \sigma \rangle \Downarrow^{(\emptyset)} \sigma'$  (by  $\mathcal{P}$ )



(vii)  $v \triangleright_{\tau} \sigma'(st)$  (by  $\mathcal{R}$ )

(viii)  $\sigma' \stackrel{\leq s_0}{=} \sigma$

(ix)  $s_0 \leq s_1$

(x)  $\mathbf{sids}(st) \prec s_1$

*Proof.* By induction on the syntax of  $\phi$ .

- Case  $\phi = \mathbf{const}_n$

There is only one possibility for each of  $\mathcal{T}$ ,  $\mathcal{E}$  and  $\mathcal{C}$ :

$$\begin{aligned}\mathcal{T} &= \overline{\mathbf{const}_n : () \rightarrow \mathbf{int}} \\ \mathcal{E} &= \overline{\vdash \mathbf{const}_n() \downarrow n} \\ \mathcal{C} &= \overline{\mathbf{const}_n() \Rightarrow_{s_0+1}^{s_0} (s_0 := \mathbf{Const}_n(), s_0)}\end{aligned}$$

So  $k = 0, \tau = \mathbf{int}, v = n, p = s_0 := \mathbf{Const}_n(), s_1 = s_0 + 1$ , and  $st = s_0$

By P-XDUCER, P-X-LOOP, P-X-TERMI and P-X-CONST, we can construct  $\mathcal{P}$  as follows:

$$\mathcal{P} = \frac{\frac{\overline{\mathbf{Const}_n() \downarrow \langle n \rangle} \quad \overline{\mathbf{Const}_n() \Downarrow^{\langle \rangle} \langle \rangle}}{\mathbf{Const}_n() \Downarrow^{\langle \rangle} \langle n \rangle}}{\langle s_0 := \mathbf{Const}_n(), \sigma \rangle \Downarrow^{\langle \rangle} \sigma[s_0 \mapsto \langle n \rangle]}$$

So  $\sigma' = \sigma[s_0 \mapsto \langle n \rangle]$ .

Then we take  $\mathcal{R} = \overline{n \triangleright_{\mathbf{int}} \sigma'(s_0)}$ .

Also clearly,  $\sigma' \stackrel{\leq s_0}{=} \sigma$ ,  $s_0 \leq s_0 + 1$ ,  $\mathbf{sids}(s_0) \prec s_0 + 1$ , and we are done.

- Case  $\phi = \mathbf{plus}$

We must have

$$\begin{aligned}\mathcal{T} &= \overline{\mathbf{plus} : (\mathbf{int}, \mathbf{int}) \rightarrow \mathbf{int}} \\ \mathcal{E} &= \overline{\vdash \mathbf{plus}(n_1, n_2) \downarrow n_3}\end{aligned}$$

where  $n_3 = n_2 + n_1$ , and

$$\mathcal{C} = \overline{\mathbf{plus}(s_1, s_2) \Rightarrow_{s_0+1}^{s_0} (s_0 := \mathbf{MapTwo}_+(s_1, s_2), s_0)}$$

So  $k = 2, \tau_1 = \tau_2 = \tau = \mathbf{int}, v_1 = n_1, v_2 = n_2, v = n_3, st_1 = s_1, st_2 = s_2, st = s_0, s_1 = s_0 + 1$  and  $p = s_0 := \mathbf{MapTwo}_+(s_1, s_2)$ .

Assumption (iv) gives us  $\overline{n_1 \triangleright_{\mathbf{int}} \sigma(s_1)}$  and  $\overline{n_2 \triangleright_{\mathbf{int}} \sigma(s_2)}$ , which implies  $\sigma(s_1) = \langle n_1 \rangle$  and  $\sigma(s_2) = \langle n_2 \rangle$  respectively.

For (v) we have  $s_1 < s_0$  and  $s_2 < s_0$ .

Then using P-XDUCER with  $\sigma(s_1) = \langle n_1 \rangle$  and  $\sigma(s_2) = \langle n_2 \rangle$ , and using P-X-LOOP and P-X-TERMI, we can build  $\mathcal{P}$  as follows:

$$\frac{\frac{\text{MapTwo}_+(\langle n_1 \rangle, \langle n_2 \rangle) \downarrow \langle n_3 \rangle \quad \text{MapTwo}_+(\langle \rangle, \langle \rangle) \Downarrow^\diamond \langle \rangle}{\text{MapTwo}_+(\langle n_1 \rangle, \langle n_2 \rangle) \Downarrow^{\langle () \rangle} \langle n_3 \rangle}}{\langle s_0 := \text{MapTwo}_+(s_1, s_2), \sigma \rangle \Downarrow^{\langle () \rangle} \sigma[s_0 \mapsto \langle n_3 \rangle]}$$

Therefore,  $\sigma' = \sigma[s_0 \mapsto \langle n_3 \rangle]$ .

Now we can take  $\mathcal{R} = n_3 \triangleright_{\text{int}} \sigma'(s_0)$ , and it is clear that  $\sigma' \stackrel{\leq s_0}{=} \sigma$ ,  $s_0 \leq s_0 + 1$  and  $\text{sids}(s_0) \leq s_0 + 1$  as required.

- Case  $\phi = \text{iota}$

■

**Theorem 1.20.** *If*

- (i)  $\Gamma \vdash e : \tau$  (by some derivation  $\mathcal{T}$ )
  - (ii)  $\rho \vdash e \downarrow v$  (by some  $\mathcal{E}$ )
  - (iii)  $\delta \vdash e \Rightarrow_{s_1}^{s_0} (p, st)$  (by some  $\mathcal{C}$ )
  - (iv)  $\forall x \in \text{dom}(\Gamma). \vdash \rho(x) : \Gamma(x)$
  - (v)  $\forall x \in \text{dom}(\Gamma). \overline{\delta(x)} \leq s_0$
  - (vi)  $\forall x \in \text{dom}(\Gamma). \rho(x) \triangleright_{\Gamma(x)} \sigma(\delta(x))$
- then
- (vii)  $\langle p, \sigma \rangle \Downarrow^{\langle () \rangle} \sigma'$  (by some derivation  $\mathcal{P}$ )
  - (viii)  $v \triangleright_\tau \sigma'(st)$  (by some  $\mathcal{R}$ )
  - (ix)  $\sigma' \stackrel{\leq s_0}{=} \sigma$
  - (x)  $s_0 \leq s_1$
  - (xi)  $\overline{st} \leq s_1$

*Proof.* By induction on the syntax of  $e$ .

- Case  $e = \{e_1 : x \text{ in } y \text{ using } x_1, \dots, x_j\}$ .

We must have:

(i)

$$\mathcal{T} = \frac{\mathcal{T}_1 \quad [x \mapsto \tau_1, x_1 \mapsto \text{int}, \dots, x_j \mapsto \text{int}] \vdash e_1 : \tau_2}{\Gamma \vdash \{e_1 : x \text{ in } y \text{ using } x_1, \dots, x_j\} : \{\tau_2\}}$$

with

$$\begin{aligned} \Gamma(y) &= \{\tau_1\} \\ (\Gamma(x_i) = \text{int})_{i=1}^j \end{aligned}$$

(ii)

$$\mathcal{E} = \frac{\left( \frac{\mathcal{E}_i}{[x \mapsto v_i, x_1 \mapsto n_1, \dots, x_j \mapsto n_j] \vdash e_1 \downarrow v'_i} \right)_{i=1}^k}{\rho \vdash \{e_1 : x \text{ in } y \text{ using } x_1, \dots, x_j\} \downarrow \{v'_1, \dots, v'_k\}}$$

with

$$\begin{aligned} \rho(y) &= \{v_1, \dots, v_k\} \\ (\rho(x_i) = n_i)_{i=1}^j \end{aligned}$$

(iii)

$$\mathcal{C} = \frac{\mathcal{C}_1}{\delta \vdash \{e_1 : x \text{ in } y \text{ using } x_1, \dots, x_j\} \Rightarrow_{s_1}^{s_0+1+j} (p_1, st_2)}$$

with

$$\begin{aligned} \delta(y) &= (st_1, s_b) \\ (\delta(x_i) = s'_i)_{i=1}^j \\ p &= s_0 := \text{Usum}(s_b); \\ (s_i &:= \text{Distr}(s_b, s'_i))_{i=1}^j \\ S_{out} &:= \text{WithCtrl}(s_0, S_{in}, p_1) \\ S_{in} &= \text{fv}(p_1) \\ S_{out} &= \overline{st_2} \cap \text{dv}(p_1) \\ s_{i+1} &= s_i + 1, \forall i \in \{0, \dots, j-1\} \end{aligned} \tag{1.43}$$

So  $\tau = \{\tau_2\}, v = \{v'_1, \dots, v'_k\}, st = (st_2, s_b)$ .(iv)  $\vdash \rho(y) : \Gamma(y)$  gives us  $\vdash \{v_1, \dots, v_k\} : \{\tau_1\}$ , which must have the derivation:

$$\frac{(\vdash v_i : \tau_1)_{i=1}^k}{\vdash \{v_1, \dots, v_k\} : \{\tau_1\}} \tag{1.44}$$

and clearly for  $\forall i \in \{1, \dots, j\}$ ,  $\vdash \rho(x_i) : \Gamma(x_i)$ , that is

$$(\vdash n_i : \mathbf{int})_{i=1}^j \tag{1.45}$$

(v)  $\overline{\delta(y)} \triangleleft s_0$  gives us

$$\overline{\delta(y)} = \overline{(st_1, s_b)} = \overline{st_1} ++ [s_b] \triangleleft s_0 \tag{1.46}$$

and  $(\overline{\delta(x_i)})_{i=1}^j \triangleleft s_0$  implies  $[s'_1, \dots, s'_j] \triangleleft s_0$ .(vi) Since  $\rho(y) \triangleright_{\Gamma(y)} \sigma(\delta(y)) = \{v_1, \dots, v_k\} \triangleright_{\{\tau_1\}} \sigma((st_1, s_b))$ , which must have the derivation:

$$\frac{\left( \frac{\mathcal{R}_i}{v_i \triangleright_{\tau_1} w_i} \right)_{i=1}^k}{\{v_1, \dots, v_k\} \triangleright_{\{\tau_1\}} (w, \langle \mathbf{F}_1, \dots, \mathbf{F}_k, \mathbf{T} \rangle)} \tag{1.47}$$

where  $w = w_1 ++ \dots ++ w_k$ , therefore we have

$$\sigma(st_1) = w \quad (1.48)$$

$$\sigma(s_b) = \langle F_1, \dots, F_k, T \rangle. \quad (1.49)$$

Also, for  $\forall i \in \{1, \dots, j\}$ ,  $\rho(x_i) \triangleright_{\Gamma(x_i)} \sigma(\delta(x_i)) = n_i \triangleright_{\mathbf{int}} \sigma(s'_i)$ , which implies

$$(\sigma(s'_i) = \langle n_i \rangle)_{i=1}^j \quad (1.50)$$

First we shall show:

- (vii)  $\left\langle \begin{array}{l} s_0 := \mathbf{Usum}(s_b); \\ (s_i := \mathbf{Distr}(s_b, s'_i);)_{i=1}^j, \sigma \\ S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1) \end{array} \right\rangle \Downarrow^{(\langle \rangle)} \sigma' \text{ by some } \mathcal{P}$
- (viii)  $\{v'_1, \dots, v'_k\} \triangleright_{\{\tau_2\}} \sigma'((st_2, s_b)) \text{ by some } \mathcal{R}$

Using P-SEQ ( $j+1$ ) times, we can build  $\mathcal{P}$  as follows:

$$\frac{\begin{array}{c} \mathcal{P}_0 \\ \langle s_0 := \mathbf{Usum}(s_b), \sigma \rangle \Downarrow^{(\langle \rangle)} \sigma_0 \end{array} \quad \frac{\begin{array}{c} \mathcal{P}_1 \\ \langle s_1 := \mathbf{Distr}(s_b, s'_1), \sigma_0 \rangle \Downarrow^{(\langle \rangle)} \sigma_1 \end{array} \quad \frac{\begin{array}{c} \mathcal{P}_{j+1} \\ \langle S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1), \sigma_j \rangle \Downarrow^{(\langle \rangle)} \sigma' \end{array} \quad \vdots}{\left\langle \begin{array}{l} (s_i := \mathbf{Distr}(s_b, s'_i))_{i=2}^j; \\ S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1) \end{array} \right\rangle \Downarrow^{(\langle \rangle)} \sigma'}}{\left\langle \begin{array}{l} (s_i := \mathbf{Distr}(s_b, s'_i);)_{i=1}^j, \sigma_0 \\ S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1) \end{array} \right\rangle \Downarrow^{(\langle \rangle)} \sigma'} \quad \frac{\left\langle \begin{array}{l} s_0 := \mathbf{Usum}(s_b); \\ (s_i := \mathbf{Distr}(s_b, s'_i);)_{i=1}^j, \sigma \\ S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1) \end{array} \right\rangle \Downarrow^{(\langle \rangle)} \sigma'}{\left\langle \begin{array}{l} s_0 := \mathbf{Usum}(s_b); \\ (s_i := \mathbf{Distr}(s_b, s'_i);)_{i=1}^j, \sigma \\ S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1) \end{array} \right\rangle \Downarrow^{(\langle \rangle)} \sigma'}$$

in which for  $\forall i \in \{1, \dots, j\}$ ,  $\mathcal{P}_i$  is a derivation of  $\langle s_i := \mathbf{Distr}(s_b, s'_i), \sigma_{i-1} \rangle \Downarrow^{(\langle \rangle)} \sigma_i$ .

For  $\mathcal{P}_0$ , with  $\sigma(s_b) = \langle F_1, \dots, F_k, T \rangle$ , we can build it as follows:

$$\begin{array}{c} \text{by P-X-USUMT} \quad \overline{\mathbf{Usum}(\langle T \rangle) \downarrow \langle \rangle} \\ \vdots \\ \text{by P-X-USUMF} \quad \frac{\mathbf{Usum}(\langle F_2, \dots, F_k, T \rangle) \downarrow \langle ()_2, \dots, ()_k \rangle}{\mathbf{Usum}(\langle F_1, \dots, F_k, T \rangle) \downarrow \langle ()_1, \dots, ()_k \rangle} \quad \text{by P-X-TERMI} \quad \frac{}{\mathbf{Usum}(\langle \rangle) \Downarrow^{\langle \rangle} \langle \rangle} \\ \text{by P-X-LOOP} \quad \frac{}{\mathbf{Usum}(\langle F_1, \dots, F_k, T \rangle) \downarrow \langle ()_1, \dots, ()_k \rangle} \\ \text{by P-XDUCER} \quad \frac{\mathbf{Usum}(\langle F_1, \dots, F_k, T \rangle) \downarrow \langle ()_1, \dots, ()_k \rangle}{\langle s_0 := \mathbf{Usum}(s_b), \sigma \rangle \Downarrow^{(\langle \rangle)} \sigma[s_0 \mapsto \langle ()_1, \dots, ()_k \rangle]} \end{array}$$

So  $\sigma_0 = \sigma[s_0 \mapsto \langle ()_1, \dots, ()_k \rangle]$ .

Similarly, with  $\sigma(s_b) = \langle F_1, \dots, F_k, T \rangle$  and  $(\sigma(s'_i) = \langle n_i \rangle)_{i=1}^j$  from (1.50), we can build each  $\mathcal{P}_i$  for  $\forall i \in \{1, \dots, j\}$  as follows:

$$\begin{array}{c}
\text{by P-X-DISTR T} \frac{\text{Distr}(\langle T \rangle, \langle n_i \rangle) \downarrow \langle \rangle}{\vdots} \\
\text{by P-X-DISTR F} \frac{\text{Distr}(\langle F_2, \dots, F_k, T \rangle, \langle n_i \rangle) \downarrow \overbrace{\langle n_i, \dots, n_i \rangle}^{k-1}}{\vdots} \\
\text{by P-X-LOOP} \frac{\text{Distr}(\langle F_1, \dots, F_k, T \rangle, \langle n_i \rangle) \downarrow \overbrace{\langle n_i, \dots, n_i \rangle}^k}{\vdots} \quad \text{by P-X-TERMI} \frac{\text{Distr}(\langle \rangle, \langle \rangle) \Downarrow \langle \rangle \langle \rangle}{\vdots} \\
\text{by P-XDUCER} \frac{\text{Distr}(\langle F_1, \dots, F_k, T \rangle, \langle n_i \rangle) \Downarrow \langle \rangle \overbrace{\langle n_i, \dots, n_i \rangle}^k}{\langle s_i := \text{Distr}(s_b, s'_i), \sigma_{i-1} \rangle \Downarrow \langle \rangle \sigma_{i-1}[s_i \mapsto \overbrace{\langle n_i, \dots, n_i \rangle}^k]}
\end{array}$$

So  $\forall i \in \{1, \dots, j\}. \sigma_i = \sigma_{i-1}[s_i \mapsto \overbrace{\langle n_i, \dots, n_i \rangle}^k]$ .

Thus  $\sigma_j = \sigma[s_0 \mapsto \langle ()_1, \dots, ()_k \rangle, s_1 \mapsto \overbrace{\langle n_1, \dots, n_1 \rangle}^k, \dots, s_j \mapsto \overbrace{\langle n_j, \dots, n_j \rangle}^k]$ .

Now it remains to build  $\mathcal{P}_{j+1}$ .

Since we have

$$\begin{aligned}
\mathcal{T}_1 &= [x \mapsto \tau_1, x_1 \mapsto \mathbf{int}, \dots, x_j \mapsto \mathbf{int}] \vdash e_1 : \tau_2 \\
(\mathcal{E}_i &= [x \mapsto v_i, x_1 \mapsto n_1, \dots, x_j \mapsto n_j] \vdash e_1 \downarrow v'_i)_{i=1}^k \\
\mathcal{C}_1 &= [x \mapsto st_1, x_1 \mapsto s_1, \dots, x_j \mapsto s_j] \vdash e_1 \Rightarrow_{s_1}^{s_0+1+j} (p_1, st_2)
\end{aligned}$$

Let  $\Gamma_1 = [x \mapsto \tau_1, x_1 \mapsto \mathbf{int}, \dots, x_j \mapsto \mathbf{int}]$ ,  $\rho_i = [x \mapsto v_i, x_1 \mapsto n_1, \dots, x_j \mapsto n_j]$  and  $\delta_1 = [x \mapsto st_1, x_1 \mapsto s_1, \dots, x_j \mapsto s_j]$ .

For  $\forall i \in \{1, \dots, k\}$ , we show the following three conditions, which allows us to use IH with  $\mathcal{T}_1, \mathcal{E}_i, \mathcal{C}_1$  later.

- (a)  $\forall x \in \text{dom}(\Gamma_1). \vdash \rho_i(x) : \Gamma_1(x)$
- (b)  $\forall x \in \text{dom}(\Gamma_1). \overline{\delta_1(x)} \leq s_0 + 1 + j$
- (c)  $\forall x \in \text{dom}(\Gamma_1). \rho_i(x) \triangleright_{\Gamma_1(x)} \sigma_{ji}(\delta_1(x))$

TS: (a)

From (1.44) and (1.45) it is clear that

$$\forall x \in \text{dom}(\Gamma_1). \vdash \rho_i(x) : \Gamma_1(x)$$

TS: (b)

From (1.46), it is clear that  $\overline{\delta_1(x)} = \overline{st_1} \leq s_0 + 1 + j$ . From (1.43), for  $\forall i \in \{1, \dots, j\}. \delta_1(x_i) = s_0 + i < s_0 + 1 + j$ . Therefore,

$$\forall x \in \text{dom}(\Gamma_1). \overline{\delta_1(x)} \leq s_0 + 1 + j$$

TS: (c)

For  $\forall i \in \{1, \dots, k\}$ , we take  $\sigma_{ji} \stackrel{S}{\sim} \sigma_j$  where  $S = \text{dom}(\sigma_j) - (\overline{st_1} \cup \{s_1, \dots, s_j\})$ ,

such that

$$\begin{aligned}\sigma_{ji}(st_1) &= w_i \\ \sigma_{ji}(s_1) &= \langle n_1 \rangle \\ &\vdots \\ \sigma_{ji}(s_j) &= \langle n_j \rangle\end{aligned}$$

It is easy to show that

$$\sigma_{j1} \stackrel{S}{\sim} \sigma_{j2} \stackrel{S}{\sim} \dots \stackrel{S}{\sim} \sigma_{jk} \stackrel{S}{\sim} \sigma_j \quad (1.51)$$

$$\sigma_{j1} \stackrel{S}{\boxtimes} \sigma_{j2} \stackrel{S}{\boxtimes} \dots \stackrel{S}{\boxtimes} \sigma_{jk} = \sigma_j \quad (1.52)$$

Also note that

$$S_{in} = \mathbf{fv}(p_1) \subseteq (\overline{st_1} \cup \{s_1, \dots, s_j\}) \cap S = \emptyset \quad (1.53)$$

$$\overline{st_2} \subseteq (\overline{st_1} \cup \{s_1, \dots, s_j\} \cup \mathbf{dv}(p_1)) \cap S = \emptyset \quad (1.54)$$

From  $\mathcal{R}_i$  in (1.47) we have  $\rho_i(x) \triangleright_{\Gamma_1(x)} \sigma_{ji}(\delta_1(x))$

and it is clear that

$$\begin{aligned}\rho_i(x_1) &\triangleright_{\Gamma_1(x_1)} \sigma_{ji}(\delta_1(x_j)) \\ &\vdots \\ \rho_i(x_j) &\triangleright_{\Gamma_1(x_j)} \sigma_{ji}(\delta_1(x_j))\end{aligned}$$

Therefore,  $\forall x \in \text{dom}(\Gamma_1). \rho_i(x) \triangleright_{\Gamma_1(x)} \sigma_{ji}(\delta_1(x))$ .

Then by IH ( $k$  times) on  $\mathcal{T}_1$  with  $\mathcal{E}_i, \mathcal{C}_1$  we obtain the following result:

$$(\langle p_1, \sigma_{ji} \rangle \Downarrow^{(\emptyset)} \sigma'_{ji})_{i=1}^k \quad (1.55)$$

$$(v'_i \triangleright_{\tau_2} \sigma'_{ji}(st_2))_{i=1}^k \quad (1.56)$$

$$(\sigma'_{ji} \stackrel{\leq s_0+j+1}{=} \sigma_{ji})_{i=1}^k \quad (1.57)$$

$$s_0 + 1 + j \leq s_1 \quad (1.58)$$

$$\overline{st_2} \leq s_1 \quad (1.59)$$

Assume  $S_{out} = \{s_{j+1}, \dots, s_{j+l}\}$ . (Note here  $s_{j+i}$  is not necessary equal to  $s_j + i$ , but must be  $\geq s_j$ ).

There are two possibilities for  $\mathcal{P}_{j+1}$ :

– Subcase  $\sigma_j(s_0) = \langle \rangle$ , i.e.,  $k = 0$ .

Then  $(\sigma_j(s_i) = \langle \rangle)_{i=1}^j$ . Also, with (3.4) and (3.5), we have  $\forall s \in \overline{st_1}. \sigma_j(s) = \langle \rangle$ ; with (3.6),  $\sigma_j(s_b) = \langle \mathbf{T} \rangle$ . Thus

$$\forall s \in (\{s_0\} \cup S_{in}). \sigma_j(s) = \langle \rangle$$

Then we can use the rule P-WC-EMP to build  $\mathcal{P}$  as follows:

$$\overline{\langle S_{out} := \mathbf{WithCtrl}(s_0, S_{in}, p_1), \sigma_j \rangle \Downarrow^{(\emptyset)} \sigma_j[(s_{j+i} \mapsto \langle \rangle)_{i=1}^l]}$$

So in this subcase, we take

$$\sigma' = \sigma_j[(s_{j+i} \mapsto \langle \rangle)_{i=1}^l] = \sigma[s_0 \mapsto \langle \rangle, s_1 \mapsto \langle \rangle, \dots, s_{j+l} \mapsto \langle \rangle] \quad (1.60)$$

TS: (viii)

Since  $k = 0$ , then  $v = \{\}$ . Also, we have

$$\begin{aligned} \sigma'(s_b) &= \sigma(s_b) = \langle T \rangle \\ \forall s \in \overline{st_2}. \sigma'(s) &= \langle \rangle \end{aligned}$$

Therefore,  $\sigma'((st_2, s_b)) = (\sigma'(st_2), \sigma'(s_b))$ , with which we construct

$$\mathcal{R} = \overline{\{\} \triangleright_{\{\tau_2\}} ((\dots(\langle \rangle), \dots), \langle T \rangle)}$$

as required.

– Subcase  $\sigma_j(s_0) = \langle () | \dots \rangle$ , i.e.,  $k > 0$ .

Since we have (1.51), (1.55) and  $\text{fv}(()p_1) \cap S = \emptyset$  from (1.53), it is easy to show that using Lemma 1.17 at most  $(k-1)$  times we can obtain

$$\langle p_1, (\boxtimes \sigma_{ji})_{i=1}^k \rangle \Downarrow^{\langle ()_1, \dots, ()_k \rangle} (\boxtimes \sigma'_{ji})_{i=1}^k \quad (1.61)$$

Let  $\sigma'' = (\boxtimes \sigma'_{ji})_{i=1}^k$ . Also with (1.52), we replace both the start and ending stores in (1.61), giving us a derivation  $\mathcal{P}'_{j+1}$  of

$$\langle p_1, \sigma_j \rangle \Downarrow^{\langle ()_1, \dots, ()_k \rangle} \sigma''$$

Now we build  $\mathcal{P}_{j+1}$  using the rule P-WC-NONEMP as follows:

$$\frac{\mathcal{P}'_{j+1} \quad \langle p_1, \sigma_j \rangle \Downarrow^{\langle ()_1, \dots, ()_k \rangle} \sigma''}{\langle S_{out} := \text{WithCtrl}(s_0, S_{in}, p_1), \sigma_j \rangle \Downarrow^{\langle () \rangle} \sigma_j[(s_{j+i} \mapsto \sigma''(s_{j+i}))_{i=1}^l]}$$

So in this subcase we take

$$\begin{aligned} \sigma' &= \sigma_j[(s_{j+i} \mapsto \sigma''(s_{j+i}))_{i=1}^l] \\ &= \sigma[s_0 \mapsto \langle ()_1, \dots, ()_k \rangle, s_1 \mapsto \langle \overbrace{n_1, \dots, n_1}^k \rangle, \dots, s_j \mapsto \langle \overbrace{n_j, \dots, n_j}^k \rangle, \\ &\quad s_{j+1} \mapsto \sigma''(s_{j+1}), \dots, s_{j+l} \mapsto \sigma''(s_{j+l})] \end{aligned} \quad (1.62)$$

TS : (viii)

Let  $\sigma'(st_2) = w'$ , and  $\sigma'_{ji}(st_2) = w'_i$ .

For  $\forall i \in \{1, \dots, k\}$ , by Definition 1.12 with (1.54), we get

$$w' = \sigma''(st_2) = w'_1 ++ \dots ++ w'_k$$

Also,  $\sigma'(s_b) = \sigma(s_b) = \langle F_1, \dots, F_k, T \rangle$ , we now have  $\sigma'((st_2, s_b)) = (\sigma'(st_2), \sigma'(s_b)) = (w', \langle F_1, \dots, F_k, T \rangle)$ . With (1.56), we can construct  $\mathcal{R}$  as follows:

$$\frac{(v'_i \triangleright_{\tau_2} w'_i)_{i=1}^k}{\{v'_1, \dots, v'_k\} \triangleright_{\{\tau_2\}} (w', \langle F_1, \dots, F_k, T \rangle)}$$

as required.

(ix) TS:  $\sigma' \stackrel{\leq s_0}{=} \sigma$

Since  $\forall s \in \{s_0\} \cup \{s_1, \dots, s_j\} \cup \{s_{j+1}, \dots, s_{j+l}\}. s \geq s_0$ , with (1.60) and (1.62), it is clear  $\forall s < s_0. \sigma'(s) = \sigma(s)$ , i.e.,  $\sigma' \stackrel{\leq s_0}{=} \sigma$  as required.

(x) TS:  $s_0 \leq s_1$

From (1.58) we immediately get  $s_0 \leq s_1 - 1 - j < s_1$ .

(xi) TS:  $\overline{(st_2, s_b)} \triangleleft s_1$

From (1.46) we know  $s_b < s_0$ , thus  $s_b < s_0 \leq s_1$ . And we already have (1.59). Therefore,

$$\overline{(st_2, s_b)} = \overline{st_2} ++ [s_b] \triangleleft s_1.$$

- Case  $e = x$ .

We must have

$$\begin{aligned} \mathcal{T} &= \frac{}{\Gamma \vdash x : \tau} (\Gamma(x) = \tau) \\ \mathcal{E} &= \frac{}{\rho \vdash x \downarrow v} (\rho(x) = v) \\ \mathcal{C} &= \frac{}{\delta \vdash x \Rightarrow_{s_0}^{s_0} (\epsilon, st)} (\delta(x) = st) \end{aligned}$$

So  $p = \epsilon$ .

Immediately we have  $\mathcal{P} = \overline{\langle \epsilon, \sigma \rangle \Downarrow^{(\cdot)} \sigma}$

So  $\sigma' = \sigma$ , which implies  $\sigma' \stackrel{\leq s_0}{=} \sigma$ .

From the assumptions (iv),(v) and (vi) we already have  $v \triangleright_{\tau} \sigma(st)$ , and  $\overline{st} \triangleleft s_0$ . Finally it's clear that  $s_0 \leq s_0$ , and we are done.

- Case  $e = \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2$ .

We must have:

$$\begin{aligned} \mathcal{T} &= \frac{\mathcal{T}_1 \quad \mathcal{T}_2}{\Gamma \vdash e_1 : \tau_1 \quad \Gamma[x \mapsto \tau_1] \vdash e_2 : \tau} \Gamma \vdash \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2 : \tau \\ \mathcal{E} &= \frac{\mathcal{E}_1 \quad \mathcal{E}_2}{\rho \vdash e_1 \downarrow v_1 \quad \rho[x \mapsto v_1] \vdash e_2 \downarrow v} \rho \vdash \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2 \downarrow v \\ \mathcal{C} &= \frac{\mathcal{C}_1 \quad \mathcal{C}_2}{\delta \vdash e_1 \Rightarrow_{s_0}^{s_0} (p_1, st_1) \quad \delta[x \mapsto st_1] \vdash e_2 \Rightarrow_{s_1}^{s'_0} (p_2, st)} \delta \vdash \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2 \Rightarrow_{s_1}^{s_0} (p_1; p_2, st) \end{aligned}$$

So  $p = p_1; p_2$ .

By IH on  $\mathcal{T}_1$  with  $\mathcal{E}_1, \mathcal{C}_1$ , we get

(a)  $\mathcal{P}_1$  of  $\langle p_1, \sigma \rangle \Downarrow^{(\cdot)} \sigma_1$

(b)  $\mathcal{R}_1$  of  $v_1 \triangleright_{\tau_1} \sigma_1(st_1)$



- (c)  $\sigma_1 \xrightarrow{\leq s_0} \sigma$
- (d)  $s_0 \leq s'_0$
- (e)  $\overline{st_1} \triangleleft s'_0$

From (b), we know  $\rho[x \mapsto v_1](x) : \Gamma[x \mapsto \tau_1](x)$  and  $\rho[x \mapsto v_1](x) \triangleright_{\Gamma[x \mapsto \tau_1](x)} \sigma_1(\delta[x \mapsto st_1](x))$  must hold. From (e), we have  $\overline{\delta[x \mapsto st_1](x)} \triangleleft s'_0$ .

Then by IH on  $\mathcal{T}_2$  with  $\mathcal{E}_2, \mathcal{C}_2$ , we get

- (f)  $\mathcal{P}_2$  of  $\langle p_2, \sigma_1 \rangle \Downarrow^{(\cdot)} \sigma_2$
- (g)  $\mathcal{R}_2$  of  $\sigma_2 \triangleright_{\tau} \sigma_2(st)$
- (h)  $\sigma_2 \xrightarrow{\leq s'_0} \sigma_1$
- (i)  $s'_0 \leq s_1$
- (j)  $\overline{st} \triangleleft s_1$

So we can construct:

$$\mathcal{P} = \frac{\mathcal{P}_1 \quad \mathcal{P}_2}{\langle p_1; p_2, \sigma \rangle \Downarrow^{(\cdot)} \sigma_2} \quad \begin{array}{l} \langle p_1, \sigma \rangle \Downarrow^{(\cdot)} \sigma_1 \quad \langle p_2, \sigma_1 \rangle \Downarrow^{(\cdot)} \sigma_2 \end{array}$$

From (c), (d) and (h), it is clear that  $\sigma_2 \xrightarrow{\leq s_0} \sigma_1 \xrightarrow{\leq s_0} \sigma$ . From (d) and (i),  $s_0 \leq s_1$ .

Take  $\sigma' = \sigma_2$  (thus  $\mathcal{R} = \mathcal{R}_2$ ) and we are done.

- Case  $e = \phi(x_1, \dots, x_k)$

We must have

$$\begin{aligned} \mathcal{T} &= \frac{\mathcal{T}_1}{\Gamma \vdash \phi(x_1, \dots, x_k) : \tau} ((\Gamma(x_i) = \tau_i)_{i=1}^k) \\ \mathcal{E} &= \frac{\mathcal{E}_1}{\rho \vdash \phi(x_1, \dots, x_k) \Downarrow v} ((\rho(x_i) = v_i)_{i=1}^k) \\ \mathcal{C} &= \frac{\mathcal{C}_1}{\delta \vdash \phi(x_1, \dots, x_k) \Rightarrow_{s_1}^{s_0} (p, st)} ((\delta(x_i) = st_i)_{i=1}^k) \end{aligned}$$

From the assumptions (iv), (v) and (vi), for all  $i \in \{1, \dots, k\}$ :

- (iv)  $\vdash \rho(x_i) : \Gamma(x_i)$ , that is,  $\vdash v_i : \tau_i$
- (v)  $\overline{\delta(x_i)} \triangleleft s_0$ , that is,  $\overline{st_i} \triangleleft s_0$
- (vi)  $\rho(x_i) \triangleright_{\Gamma(x_i)} \sigma(st_i)$ , that is,  $v_i \triangleright_{\tau_i} \sigma(st_i)$

So using Lemma 1.19 on  $\mathcal{T}_1, \mathcal{E}_1, \mathcal{C}_1, (a), (b)$  and (c) gives us exactly what we shall show. ■

## 1.6 Scaling up

# Bibliography

- [Ble89] Guy E Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, 1989.
- [Ble95] Guy E Blelloch. Nesl: A nested data-parallel language.(version 3.1). Technical Report CMU-CS-95-170, School of Computer Science, Carnegie Mellon University, 1995.
- [Ble96] Guy E Blelloch. Programming parallel algorithms. *Communications of the ACM*, 39(3):85–97, 1996.
- [Mad13] Frederik M. Madsen. A streaming model for nested data parallelism. Master’s thesis, Department of Computer Science (DIKU), University of Copenhagen, March 2013.
- [Mad16] Frederik M. Madsen. *Streaming for Functional Data-Parallel Languages*. PhD thesis, Department of Computer Science (DIKU), University of Copenhagen, September 2016.
- [PP93] Jan F Prins and Daniel W Palmer. Transforming high-level data-parallel programs into vector operations. In *Proceedings of the Fourth SIG-PLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP’93, pages 119–128. ACM, 1993.
- [PPCF95] Daniel W Palmer, Jan F Prins, Siddhartha Chatterjee, and Rickard E Faith. Piecewise execution of nested data-parallel programs. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 346–361. Springer, 1995.