1 Concatenation Lemma Proof

Lemma 1.1. *If $\sigma_1 \overset{\mathbf{S}}{\sim} \sigma_2, \langle p, \sigma_1\rangle \downarrow^{\vec{c}_1} \sigma_1'$ by some derivation $\mathcal{P}_1$, $\langle p, \sigma_2\rangle \downarrow^{\vec{c}_2} \sigma_2'$ by some derivation $\mathcal{P}_2$, and $\mathbf{FV}(p) \cap \mathbf{S} = \emptyset$, then $\langle p, \sigma_1 \overset{\mathbf{S}}{\bowtie} \sigma_2\rangle \downarrow^{\vec{c}_1 +\!+ \vec{c}_2} \sigma_1' \overset{\mathbf{S}}{\bowtie} \sigma_2'$ by some derivation $\mathcal{P}$.*

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 compuation 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 **SId**s in all σs (i.e., $\mathbf{S}$) have no change throughout the process.

The proof of this lemma is by induction on the syntax of p. Three of the proof cases including $p = \epsilon \mid s := \psi(s_1, ..., s_k) \mid p_1; p_2$, are relatively regular. The tricky one is the case where $p = st :=$ $\texttt{WithCtrl}(s_0, p_1)$: how should we *split* a computation itself already being a parallel compuation?

1.1 Analysis of WithCtrl proof case

This subsection is some of my analysis of this case, should be removed after been solved.

In this lemma, the condition $\mathbf{FV}(p) \cap \mathbf{S} = \emptyset$ is indispensable. The free variable streams of p are those we are going to split and distribute on smaller degree compuations. At the first sight, we may think $\mathbf{FV}(st := \texttt{WithCtrl}(s_0, p_1)) = \mathbf{FV}(p_1)$, which are the supplier streams used in but defined before p_1. But there is another essential stream that should be splitted as well, that is the new control stream s_0. We should keep in mind that *in practice the control stream is used implicitly as the first supplier channel for all transducers to decide how much data they should consume and produce.* So we should fix $\mathbf{FV}(st := \texttt{WithCtrl}(s_0, p_1)) = \mathbf{FV}(p_1) \cup \{s_0\}$, which means $\sigma_1(s_0)$ can be different from $\sigma_2(s_0)$.

Now there are still three (or more precisely, four) subcases to discuss:

(a) both $\sigma_1(s_0)$ and $\sigma_2(s_0)$ are empty

(b) both $\sigma_1(s_0)$ and $\sigma_2(s_0)$ are non-empty

(c) one of them is empty and the other non-empty.

Again, the tricky one is the last subcase. Suppose $\sigma_1(s_0) = \langle\rangle, \sigma_2(s_0) = \langle ()|\vec{c}_2'\rangle$, then we may start the proof of this subcase as follows:

Proof. Case $p = st := \texttt{WithCtrl}(s_0, p_1)$

- Subcase $\sigma_1(s_0) = \langle\rangle, \sigma_2(s_0) = \langle ()|\vec{c}_2'\rangle$
  Must have

$$\mathcal{P}_1 = \frac{}{\langle st := \texttt{WithCtrl}(s_0, p_1), \sigma_1\rangle \downarrow^{\vec{c}_1} \sigma_1[st \rightarrowtail \langle\rangle]}$$

$$\mathcal{P}_2 = \frac{\begin{array}{c}\mathcal{P}_2' \\ \langle p_1, \sigma_2\rangle \downarrow^{\langle ()|\vec{c}_2'\rangle} \sigma_2''\end{array}}{\langle st := \texttt{WithCtrl}(s_0, p_1), \sigma_2\rangle \downarrow^{\vec{c}_2} \sigma_2[st \rightarrowtail \sigma_2''(st)]}$$

  Since $\sigma_1 \overset{\mathbf{S}}{\bowtie} \sigma_2(s_0) = \sigma_1(s_0) +\!+ \sigma_2(s_0) = \langle ()|\vec{c}_2'\rangle$, we should construct $\mathcal{P}$ as follows:

$$\frac{\begin{array}{c}\mathcal{P}' \\ \langle p_1, \sigma_1 \overset{\mathbf{S}}{\bowtie} \sigma_2\rangle \downarrow^{\langle ()|\vec{c}_2'\rangle} \sigma''\end{array}}{\langle st := \texttt{WithCtrl}(s_0, p_1), \sigma_1 \overset{\mathbf{S}}{\bowtie} \sigma_2\rangle \downarrow^{\vec{c}_1 +\!+ \vec{c}_2} \sigma_1 \overset{\mathbf{S}}{\bowtie} \sigma_2[st \rightarrowtail \sigma''(st)]}$$

  Then the problem is that we can not obtain $\mathcal{P}'$ just from $\mathcal{P}_2$, becasue we know nothing about how p_1 will behave with a start store σ_1:

$$\langle p_1, \sigma_1\rangle \downarrow^{\vec{c}_1} ???$$

□

One observation is that throughout the execution of any SVCODE program the control stream used in active/fired transducers is supposed non-empty in any cases for two reasons. First, at the beginning of the execution, we specify a stream $\langle()\rangle$ as the initial control stream. Second, during the execution, when we are going to switch to a new control stream, we always first detect whether it is empty or not. If not, we start running the subprogram with this new control stream as usual; otherwise we skip the execution. So empty control stream will never have a chance to be used in the real execution of any programs.

Q1: Then maybe we can modify the judgment of SVCODE evaluation from

$$\boxed{\langle p, \sigma\rangle \downarrow^{\vec{c}} \sigma'} \overset{to}{\Rightarrow} \boxed{\langle p, \sigma\rangle \downarrow^{\langle() | \vec{c}\rangle} \sigma'} \ ?$$

Also, as we can see from the semantics of transducers, if the control stream is not empty, then the parameter streams can not be empty either.

Q2: Is it better to modify the judgment of transducer semantics from

$$\boxed{\psi(\vec{a}_1, ..., \vec{a}_k) \downarrow^{\vec{c}} \vec{a}} \overset{to}{\Rightarrow} \boxed{\psi(\vec{c}, \vec{a}_1, ..., \vec{a}_k) \downarrow \vec{a}} \ ?$$

Now return back to the lemma, since $\vec{c}_1$ and $\vec{c}_2$ are not possible to be empty, then s_0 in $p = st :=$ `WithCtrl`(s_0, p_1) is non-empty either. Therefore we will only need to prove one possible subcase, that is the (b) subcase above, which should be provable.

But there is still one exception (although it does not exist in SNESL-Level0), that is, the `EmptyCtrl` instruction, which is used to generate empty sequences such as {}{`int`}. This instruction does not need any suppliers and only generate an empty stream $\langle\rangle$, so I even didn't write a Xducer for it. (More details about this instruction can be found at `Github: Issue 13`, item 2.)

For example, consider {{}`int: y in &3`}, in which p is $s_8 :=$ `WithCtrl`$(s_7, [])$ and $p_1 = []$.

```
 > :c {{}int: y in &3}
   S0  := Ctrl
   S1  := Const 3
   S2  := ToFlags 1
   S3  := Usum 2
   WithCtrl S3 (import []):
       S4  := Const 1
         Return: (IStr 4)
   S5  := SegscanPlus 4 2
   S6  := Usum 2
   WithCtrl S6 (import []):  -- S6 is non-empty
       S7  := EmptyCtrl
       WithCtrl S7 (import []):  -- but S7 is empty
           Return: (IStr 8)
       S9  := Const T
       Return: (SStr (IStr 8), 9)
   Return: (SStr (SStr (IStr 8), 9), 2)

-- stream values after execution
   (0,SBVal [False])
   (1,SIVal [3])
   (2,SBVal [False,False,False,True])
   (3,SBVal [False,False,False])
   (4,SIVal [1,1,1])
   (5,SIVal [0,1,2])
   (6,SBVal [False,False,False])
   (7,SBVal [])
   (8,SIVal [])
   (9,SBVal [True,True,True])
```

Q3 ?? I'm not sure what the best way to deal with it should be. As far as I can see, one fesible solution may be adding the following condition to the lemma to rule out this `EmptyCtrl` case:

$$\exists s \in \mathbf{FV}(p).\sigma_1(s) \neq \langle\rangle \ and \ \sigma_2(s) \neq \langle\rangle$$

It says that at least one free variable stream (then must be the new control stream) should be non-empty in both σ_1 and σ_2.