A Denotational Semantics of a Probabilistic Stream-Processing Language

Yohei Miyamoto\textsuperscript{1} Kohei Suenaga\textsuperscript{1,2} Koji Nakazawa\textsuperscript{3}
\textsuperscript{1}: Kyoto University \textsuperscript{2}: PRESTO, JST \textsuperscript{3}: Nagoya University
\{ymiyamoto,ksuenaga,knak\}@fos.kuis.kyoto-u.ac.jp

1. Introduction
Stream-processing programming languages are used to specify computation on streams (i.e., infinite sequences of values). Since a stream can naturally express time-series data, these languages are widely used in industry to model the behavior of dynamical systems.

The formal semantics of such languages is not only a theoretically interesting topic; it is practically important as a foundation of the simulation of and reasoning about a model. To this end, several formal semantics of stream-processing languages have been proposed [1–3].

This paper describes a denotational semantics of a stream-processing language extended with probabilistic behavior. Although probabilistic behavior is useful in modeling uncertainty consisting in a model, to the best of our knowledge, the formal semantics of a probabilistic stream-processing language has been less investigated.

Our semantics extends one of our previous denotational semantics of a stream-processing language [5]. In the previous semantics, the denotation of streams was given by the domain of value streams. We now adapt it by using the set of probability distributions over streams as the domain. In our theoretical development, we rely on a theorem introduced by Saheb-Djahromi [4] stating that the set of probability distributions on a cpo constitutes a cpo with respect to a natural partial order.

As a first step toward a semantics of a full-fledged probabilistic stream-processing language, we consider a simple language, the probabilistic behavior of which is determined only by the term \( B^p \). This term produces a stream consisting of \( p \), where \( p \) is determined by the probabilistic behavior of the term.

2. Language
We designate the set of stream variables \( \text{Var} \) ranging over by \( x, y, z, \ldots \); a stream variable is bound to a stream of values. We also designate the set of node names \( \text{NdName}_{m,n} \) ranging over by \( f, g, h, \ldots \); a node name represents a function from an \( m \)-tuple of streams to an \( n \)-tuple of streams.

**Definition 1.** The syntax of language \( \text{SPROC}_p \) is defined by the following BNF:

\[
\begin{align*}
  e &::= x \mid e_1 \text{op} \ e_2 \mid \text{if } b \text{ then } e_1 \text{ else } e_2 \\
  &\quad \mid \pi_k(f(e_1, \ldots, e_m)) \quad \text{where } f \in \text{NdName}_{m,n} \text{ and } k \in \{1, \ldots, n\} \\
  b &::= \text{true} \mid \text{false} \mid b_1 \land b_2 \mid \neg b \mid e_1 \text{ rop } e_2 \\
  d &::= \text{node } f(x_1, \ldots, x_n) \text{ returns}(y_1, \ldots, y_n) \mid \text{with}(y_1, \ldots, y_n, z_1, \ldots, z_l) = (e_1, \ldots, e_n, e'_1, \ldots, e'_l) \quad \text{where } f \in \text{NdName}_{m,n} \\
  p # &::= d_1, \ldots, d_n; e
\end{align*}
\]

\textsuperscript{1}For uncountable \( D \), the probability distribution over \( D \) does not form a cpo in general; see Saheb-Djahromi [4].
\[ \text{Figure 1. Denotation of stream expressions.} \]

SPROC \( P \) is an extension of the SPROC given by Suenaga and Hasuo [5] with expression \( B^P \); for a detailed explanation, see their paper. The metavariable \( e \) ranges over arithmetic stream expressions, the denotation of which are given by \( D^+_\mathbb{R} \). The symbol \( e \) represents constant streams. The symbol \( \text{aop} \) ranges over pointwise arithmetic operations between two streams. The expression \( e_1 \text{ aop } e_2 \) evaluates to a stream, the head of which is \( e_1 \). The expression \( \sum_{s_1 \in D^+} P_{e_1 \text{ aop } e_2}(s_1, s_2) \) is the denotation of \( \sum_{s_1 \in D^+} P_{e_1 \text{ aop } e_2}(s_1, s_2) \).

The metavariable \( b \) is for Boolean stream expressions that evaluate to streams of Boolean values. The symbol \( \text{rop} \) is for operations between arithmetic values.

The metavariable \( d \) is for node definitions; an \((m, n)\)-node (or simply node) is a function from an \( m \)-tuple of arithmetic streams to an \( n \)-tuple of arithmetic streams. The body \((y_1, \ldots, y_m, z_1, \ldots, z_n) = (e_1, \ldots, e_m, e'_1, \ldots, e'_n)\) of a node definition is mutually recursive definitions of stream variables.

\textbf{Example 1.} The following node \( \text{Sum} \) takes a stream of rational numbers \((r_i)_{i \in \mathbb{N}}\) and returns \((\sum_{j=0}^n r_j)_{j \in \mathbb{N}}\):

\begin{verbatim}
node \( \text{Sum}(x) \) returns \( y \) with \( y = 0 \text{ fby } (x + y) \).
\end{verbatim}

\section{3. Semantics}

We abuse our notation of the language constructors as operations on finite sequences. For example, the following equations hold: \((1, 2, 3) + (4, 5, 6) = (5, 7, 9); (1, 2, 3) \text{ fby } (4, 5) = (1, 4, 5); (1, 2, 3) \text{ fby } (x, y, z) = (x, y, z); (1, 2, 3) < (4, 5, 6) \iff (1 < 4) \land (2 < 5) \land (3 < 6)\).

Figure 1 defines the denotation \( [e]_{\mathbb{R},d} \) for arithmetic stream expression \( e \), \([b]_{\mathbb{R},d} \) for Boolean stream expression \( b \), and \([d]_{\mathbb{R},d} \) for node definition \( d \). Here, \( \phi \) is a node environment, a map from node names to their denotation; \( \delta \) is a stream environment, a map from variables to their denotation. The denotation \( [e]_{\mathbb{R},d} \) (resp. \([b]_{\mathbb{R},d} \), \([d]_{\mathbb{R},d} \)) is given by a probability distribution on \( D^+_\mathbb{R} \) (resp. \( B^+ \), \( D^+_\mathbb{R} \)).

\textbf{Example 2.} Consider the following node definition:

\begin{verbatim}
node \( \text{SumP}() \) returns \( y \) with
\( s = \text{ if } B^+ \text{ then } 1 \text{ else } -1 \)
\( y = \text{ Sum}(s) \).
\end{verbatim}

We show that the probability of \( \text{SumP}() \) generating a prefix 01, which is given by \( \text{SumP}() \text{ fby } B^+ \text{ fby } B^+ \text{ fby } B^+ \text{ fby } B^+ \), is \( \frac{1}{2} \). By definition, this is equal to \( \text{Sum}(s)(01) \): in order to calculate this value, one needs to evaluate \([0 \text{ fby } (x + y)]_{\phi, \delta} \text{ fby } (0 \text{ fby } (x + y)]_{\phi, \delta} \text{ fby } (0 \text{ fby } (x + y)]_{\phi, \delta} \text{ fby } (0 \text{ fby } (x + y)]_{\phi, \delta} \text{ fby } (0 \text{ fby } (x + y)]_{\phi, \delta} \). This is equal to \( \text{delta}(x)(1) \), since \( \text{delta}(y)(0) = \text{delta}(x)(0) \). By definition, \( \text{SumP}() \text{ fby } B^+ \text{ fby } B^+ \text{ fby } B^+ \text{ fby } B^+ \) is equal to \( \frac{1}{2} \).

\textbf{Example 3.} Consider the following definition of the node \( \text{Geom} \):

\begin{verbatim}
node \( \text{Geom}(n) \) returns \( y \) with \( y = n \text{ fby } (\text{if } B^+ \text{ then } \text{Geom}(n) \text{ else } \bot) \).
\end{verbatim}

where \( \bot \) is the shorthand for an expression having the denotation \( \bot \). We show \( \text{Geom}(n) \) is \( (p_1, \ldots, p_m) = \frac{1}{n} \) if \( n = p_1 = \cdots = p_m \) by induction on \( m \). The base case \( \text{Geom}(n) \) is \( 1 \).
because $\epsilon^* = D^c$ by definition. The induction case is

$$
\begin{align*}
\text{Geom}(n) & (p\vec{p}) \\
= \quad n \textbf{fby} (\text{if } B \vec{=} \text{then } \text{Geom}(n) \text{ else } \bot) (p\vec{p}) \\
\text{because } p = n \text{ and } [n] \text{ is independent of the other distributions} \\
= \quad \text{if } B \vec{=} \text{then } \text{Geom}(n) \text{ else } \bot (p\vec{p}) \\
\text{because } [B \vec{=} ] \text{ and } [\bot] \text{ are independent of the other distributions} \\
= \quad \frac{1}{2} \text{Geom}(n) (p\vec{p}) \\
\text{by I.H.} \\
= \quad \frac{1}{2}\text{I.H.} \\
\end{align*}
$$

4. Conclusion

We presented a denotational semantics of a probabilistic stream-processing programming language. The semantics exploits the cpo-structure of $D(D^c)$.

We recognize that the current semantics is a cleaver for cracking a nut; for a node definition that contains $B^p$, by replacing it with a stream variable $x$ and adding it as an argument of the definition, we can hoist every probabilistic expression to the top level. For such a translated program, one can give denotations without using the cpo structure of $D(D^c)$. However, we believe that the current idea extends well for more complex probabilistic expressions, such as those having parameters that change dynamically.

References


