Abstract. We specify the black box behavior of dataflow components by characterizing the relation between their input and their output histories. We distinguish between three main classes of such specifications, namely time independent specifications, weakly time dependent specifications and strongly time dependent specifications. Dataflow components are semantically modeled by sets of timed stream processing functions. Specifications describe such sets by logical formulas. We emphasize the treatment of the well-known fair merge problem and the Brock/Ackermann anomaly. We give refinement rules which allow specifications to be decomposed modulo a feedback operator.

1 Introduction

Dataflow components can be specified by formulas with a free variable ranging over domains of so-called stream processing functions [7], [5]. Both time independent and time dependent components can be described this way. In the latter case, the functions are timed in the sense that the input/output streams may have occurrences of a special message representing a time signal. For such specifications elegant refinement calculi can be formulated.

Stream processing functions are required to be both monotonic and continuous with respect to the prefix ordering on domains of stream tuples. Unfortunately, there are certain weakly time dependent components, whose behaviors cannot be specified in terms of prefix monotonic stream processing functions, although explicit timing is not really needed in order to specify their black box behavior. A famous example of such a component is an agent which outputs a fair merge of the messages it receives on two input channels [7]. The behaviors of such components can of course be specified in terms of timed stream processing functions. However, this is a bit like shooting sparrows with a shot-gun.

In an attempt to abstract from unnecessary time-dependency, this paper advocates a technique, where the black box behavior of dataflow networks is specified by characterizing the relation between the input and the output streams. We distinguish between three main classes of such specifications, namely time independent specifications, weakly time dependent specifications and strongly

* This work is supported by the Sonderforschungsbereich 342 “Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen”.

Manfred Broy and Ketil Stølen
Institut für Informatik, TU München, Postfach 20 24 20
D-80290 München, Germany
time dependent specifications — from now on shortened to ti-specifications, wtd-specifications and std-specifications, respectively. For each class of specifications refinement rules are given, which allow specifications to be decomposed modulo a feedback operator. Rules, which allow a specification of one class to be translated into a specification of another class, are also given.

Section 2 describes the underlying formalism. In Sect. 3 we introduce the three main classes of specifications. The refinement of such specifications is the topic of Sect. 4. Then, so-called general specifications are introduced in Sect. 5, and the refinement of general specifications is discussed in Sect. 6. Finally, Sect. 7 contains a brief summary and draws some conclusions.

2 Underlying Formalism

Let \( D \) be the set of messages. A stream is a finite or infinite sequence of messages. It models the history of a communication channel by representing the sequence of messages sent along the channel. Given a set of messages \( D \), \( D^* \) denotes the set of all finite streams generated from \( D \); \( D^\infty \) denotes the set of all infinite streams generated from \( D \), and \( D^\omega \) denotes \( D^* \cup D^\infty \).

Let \( d \in D \), \( r, s \in D^\omega \), \( A \subseteq D \) and \( j \) be a natural number, then:

- \( \varepsilon \) denotes the empty stream;
- \( \{d_1, \ldots, d_j\} \) denotes a stream of length \( j \), whose first message is \( d_1 \), whose second message is \( d_2 \), etc.;
- \( \text{ft}(r) \) denotes the first element of \( r \) if \( r \) is not empty;
- \( \#r \) denotes the length of \( r \);
- \( d^n \), where \( n \in \mathbb{N} \cup \{\infty\} \), denotes a stream of length \( n \) consisting of only \( d \)'s;
- \( r|_j \) denotes the prefix of \( r \) of length \( j \) if \( j < \#r \), and \( r \) otherwise;
- \( d \& s \) denotes the result of appending \( d \) to \( s \);
- \( r \preceq s \) denotes \( r \) if \( r \) is infinite and the result of concatenating \( r \) with \( s \), otherwise;
- \( r \subseteq s \) holds if \( r \) is a prefix of \( s \).

Some of the stream operators defined above are overloaded to tuples of streams in a straightforward way. \( \varepsilon \) will also be used to denote tuples of empty streams when the size of the tuple is clear from the context. If \( d \) is an \( n \)-tuple of messages, \( j \) is a natural number and \( r, s \) are \( n \)-tuples of streams, then \( \#r \) denotes the length of the shortest stream in \( r \); \( d \& s \) denotes the result of applying \( \& \) pointwisely to the components of \( d \) and \( s \); \( r|_j \), \( r \preceq s \) and \( r \subseteq s \) are generalized in the same pointwise way.

A chain \( c \) is an infinite sequence of stream tuples \( c_1, c_2, \ldots \) such that for all \( j \geq 1 \), \( c_j \subseteq c_{j+1} \). \( \cup c \) denotes \( c \)'s least upper bound. Since streams may be infinite such least upper bounds always exist.

A Boolean function \( P : (D^\omega)^n \rightarrow \mathbb{B} \) is called admissible iff whenever \( P \) yields true for each element of a chain, then it yields true for the least upper bound of the chain. We write \( \text{adm}(P) \) iff \( P \) is admissible. \( P \) is prefix-closed iff whenever it yields true for a stream tuple, then it also yields true for any prefix of this