Formal Specification and Implementation of Priority Queue using Stream Functions

Gongzhu Hu, Central Michigan University, Mount Pleasant, MI, USA
Jin Zhang, Hainan University, Haikou, China
Roger Lee, Central Michigan University, Mount Pleasant, MI, USA

ABSTRACT

Formal specification of software components, as a core research area in software engineering, has been widely studied for decades. Although quite a few formal models have been proposed for this purpose, specification of concrete software components is still a challenging task due to the complexity of the functionalities of the components. In this paper, the authors use the stream function model to specify the behavior of priority queue, a commonly used software component. This specification formally defines the regular behavior and fault tolerance behavior of priority queue. In particular, a priority-concatenation operator is defined to handle the ordering of data items to ensure the highest-priority item is removed first. A finite state machine is built based on this specification as an implementation of priority queue. In addition, the authors also discuss a priority upgrading approach to handle possible starvation situation of low-priority data items in the priority queue.

Keywords: Priority-Concatenation, Software Components, Starvation Prevention, State Transition Machine, Stream Function

INTRODUCTION

Over the last decade, software development has a tendency to build on models or components. From the software engineering point of view, model-based or component-based approaches provide an efficient way to improve the software development process in each stage of the life cycle of a software package.

The first and utmost critical aspect of component-based software development is the specification that describes the functionality and behavior of the component. There are three levels of component specifications - informal, self-formal and formal. Informal specification describes the component’s behavior in natural language that needs to be translated to code by individual programmer. Semi-formal specification uses some descriptive language or other representation to specify the component so that the interpretation of the component’s behavior is less ambiguous (if not all precise). Most
specification standards today are semi-formal. For example, UML, the industry-standard specification language, uses graphical notation, case notation and component notation (Object Management Group, 2003). The most desirable approach is formal specification by which the behavior or functionality of a software component is described in a formal language (algebraic, for example) to ensure the correctness of the software component when implemented according to the specification.

Several formal specification methods have been proposed in the past, such as meta-modeling framework (Övergaard, 2000), object-oriented paradigm (Lau & Ornaghi, 2001), Object-Z specification language (Smith, 2000), algebraic semantics (Bidot & Hennicker, 2008), and stream function (Stephens, 1997). With stream function, a software component is considered a “black box” interacting with its operational environment and its functionality is described as a mapping between the input and output streams. Such mapping can lead directly to an implementation of the component (Breitling & Philipps, 2000).

Several software components, such as stack and queue that are widely used data structures, have been formally specified as stream functions (Dosch & Hu, 2007, Hu, 2009). The items entered into these software components are simply data values (or objects) and identified by their positions (ordering) in the input stream. In this paper, we give a formal specification of priority queue as a mapping of stream function where the items are identified by their priorities, and by their positions in the input stream for those that have the same priority.

The main contribution of this work is an introduction of the priority-concatenation operator $\&_p$, that altered the ordering of the items of two streams being concatenated based on the priorities of the items. This is an enhancement of the regular concatenation operator for all previous software component specifications using streams.

**STREAM FUNCTION**

First, we briefly review the basic concept of stream function as formal model for component specification.

Given an alphabet $A$, the set $A^*$ comprises all streams $A = (a_1, a_2, ..., a_k)$ of length $|A| = k$ with element $a_i \in A \{ i \in \{1, k\}, k \geq 0 \}$.

Several operations can be applied to communication streams, one of the basic operator is concatenation defined as

$\& : A^* \times A^* \rightarrow A^*$

of two streams $A = (a_1, a_2, ..., a_k)$ and $B = (b_1, b_2, ..., b_l)$, that yields the stream

$A \& B = (a_1, a_2, ..., a_k, b_1, b_2, ..., b_l)$ (1)

A stream function $f : A^* \rightarrow B^*$ maps an input stream to an output stream (Stephens, 1997).

Let Input and Output, or simply $I$ and $O$, be the input and output alphabets, respectively. For an interactive software component $C$, its behavior can be specified as a stream function.

$C : I^* \rightarrow O^*$

where the domains of $I$ and $O$ depend on the characteristics of the component $C$.

**PRIORITY QUEUE**

Priority queue is a data structure for objects that are entered and removed based on the priority associated with the objects. That is, the object removed from the priority queue is the one with the highest priority. We shall give a formal specification of priority queue in this section.

**Basic Definition**

Data items and priorities. Let $D = \{d_1, d_2, \ldots\}$ be the set of items storable in a priority queue, $P = \{p_1, p_2, \ldots, p_m\}$ be a set of priority values with $m$ being the maximum priority associated
Designing Secure Software by Testing Application of Security Patterns
www.igi-global.com/chapter/designing-secure-software-by-testing-application-of-security-patterns/221715?camid=4v1a

Two Heads Are Better Than One: Leveraging Web 2.0 for Business Intelligence
www.igi-global.com/article/two-heads-better-than-one/44683?camid=4v1a