

# Formal Methods in Information Management System Specifications: A Case Study\*

*S. Ramaswamy, Tsz S. Cheng*

*The Laboratory of Software and Data Engineering, School of Computer and Applied Sciences*

*Georgia Southwestern College, Americus, GA 31709, USA*

*Y. Zhang*

*Portable Graphics, Austin, TX 78758*

## **Abstract**

*The formal specification of information management systems provide long term advantages in terms of systems development and maintenance. In this paper, algebraic specification techniques are used for the formal design specification of a library information management system. The derived specification has been used as a basis for the implementation of an object-oriented library information management system. The use of formal techniques in information system specifications greatly help the construction of easily verifiable system designs and lead to highly adaptable system implementations. The derived specification led to the easy development of class hierarchies. Also, it was noted that modifications and changes in the search criteria were easy to accommodate.*

## **I. Introduction**

Modern information management systems are highly complex and they undergo rapid and constant changes in their functionality and usage. Therefore, it is important that these mission critical systems be accurate, verifiable and flexible to constant changes in their operational environment. In recent years, the use of formal specification techniques has grown to be a highly desirable means of designing, developing and maintaining large and complex information systems. In this paper, the simplicity and appropriateness of formal specification techniques is illustrated using the algebraic specification technique to specify a Library Information Management System (LIMS). This LIMS specification has not only led to the easy development of class hierarchies, but the LIMS system is well adapted to modifications and changes in the search procedures.

The LIMS system is a database of all information regarding books and other reading material in a library. This database is stored as a booklist, which is an unordered sequence of book informations. Any new information that results from the addition of a book to the library is appended to the end of

---

\* Address all correspondences to Dr. S. Ramaswamy. Phone: (912)-931-2100, Fax: (912)-931-2270, email: srini@gswrs6k1.gsw.peachnet.edu

this booklist. The system is expected to answer queries related to the status of the book in the library. For this reason, the system has different types of ordered element lists that index on certain specified search strings and provides an index into the booklist. In this specification, since the search terms have been restricted to author name, title, and call number, there are three element lists. A removal of a book from the library reflects the deletion of information related to the book from this booklist, and the other element lists.

An entry in such an element list is a tuple  $\{string, index\}$ , where string represents an author name in case of an author list, a title in case of a title list, or a call number in case of a call number list and index is an index (*book id. number*) into the booklist for the particular book information. For example, consider a book titled “HCI Design” written by “Alan Dix” and its call number being “123456” . Now if this book is added to the library, then information pertaining to this book is added to the booklist with a book identification number, say 1000. Then, the three element lists, the author, title and call number list are updated such that the tuples  $\{Dix, 1000\}$ ,  $\{HCI\ Design, 1000\}$ ,  $\{123456, 1000\}$  are added to the three element lists in an appropriate order.

## II. Algebraic Specification of the LIMS System

The algebraic specification of the LIMS system has been done systematically with emphasis on the generalization of the various search methods. For example, although a search for a particular book in the library may be carried out in various ways (viz. search on author name, search on title, etc.), the specification has been done in such a way that a search for a particular book is defined irrespective of the type (author name, title, etc.) of the search string.

### A. Basic Constructs

A string, an integer list and a tuple of information such as:  $\{(name, index\ number), (title, index\ number)\}$  etc., are the basic constructs in this specification. For easier understanding and completeness purposes, the LIMS specifications of these building blocks and operations on them, before the actual specification of the system.

**A.1. Specification of a String** Some important functions for the specification of a string include: (i) **HeadC**: This is used to extract the head (first character) of the string, (ii) **TailC**: This function is used to return a string  $s$  without its first element, (iii) **LengthC**: This function calculates the length of the string, (iv) **Match**: This function matches two strings and checks for a string match. If the two strings are of different lengths then the shorter string is recursively matched with the longer string with its last character deleted until the strings are of equal length. Once the strings are of equal length every

character position in both the strings are checked for a match, and, (v) **Greater**: This function checks whether a given string is greater or lesser than another string and can be used to create an ordering.

<b>String</b>	
Import:	Integer, Char, Boolean
<b>Constructors for String:</b>	
CreateCS:	$\rightarrow String$
ConsC:	$String \times Char \rightarrow String$
<b>Observers for String:</b>	
HeadC:	$String \rightarrow Char$
TailC:	$String \rightarrow String$
DeleteC:	$String \rightarrow String$
LengthC:	$String \rightarrow Integer$
Greater:	$String \times String \rightarrow Boolean$
Match:	$String \times String \rightarrow Boolean$
$\forall s, s_1, s_2 \in String, c, c_1, c_2 \in Char$	
HeadC(CreateCS) =	Undefined
HeadC(ConsC(s, c)) =	If (s = CreateCS) then c else Head(s)
TailC(CreateCS) =	CreateCS
TailC(ConsC(s, c)) =	If (s = CreateCS) then CreateCS else ConsC(TailC(s), c)
DeleteC(CreateCS) =	CreateCS
DeleteC(ConsC(s, c)) =	s
LengthC(CreateCS) =	0
LengthC(ConsC(s, c)) =	LengthC(s) + 1
Greater(CreateCS, CreateCS) =	False
Greater(s, CreateCS) =	True, when s != CreateCS
Greater(CreateCS, s) =	False, when s != CreateCS
Greater(ConsC(s1, c1), ConsC(s2, c2)) =	If (HeadC(ConsC(s1, c1)) > HeadC(ConsC(s2, c2))) then True else if (HeadC(ConsC(s1, c1)) = HeadC(ConsC(s2, c2))) then Greater(TailC(ConsC(s1, c1)), TailC(ConsC(s2, c2))) else False
Match(CreateCS, CreateCS) =	True
Match(ConsC(s1, c1), ConsC(s2, c2)) =	If (LengthC(s1) = LengthC(s2)) then { if (c1 = c2) then Match(s1, s2) else False} else if (LengthC(s1) > LengthC(s2)) then Match(s1, ConsC(s2, c2)) else if (LengthC(s1) < LengthC(s2)) then Match(ConsC(s1, c1), s2)

**Figure 1. Trait 1: String**

**A.2. Specification of an Integer List** LIMS uses integers as indexes to book information. Therefore, the definition of a list of integers (hereafter called an intlist) becomes essential. All books are uniquely identified by a book identification number called bookid, and LIMS is assumed to recover the book information using this unique identification number. During informal discussions, bookid and index are used interchangeably. However, as functions they have been defined and treated separately. An intlist is a list of integers. Figure 2 represents the algebraic specification of an Intlist, with the necessary axioms<sup>2</sup>. Operations include: (i) **HeadI**: This function returns the element at the head of the integer list, (ii) **TailI**: This function is used to remove the head of a given list and return its remaining elements, (iii) **GetI**: Given an integer value that indicates a position and an integer list this function returns the

<sup>2</sup> Null indicates that this operation returns a null element that is not an integer

integer element at that position within the list, (iv) **OpsI**: This function defines the logical combination operations of two elements in a list, the operations being AND, OR, or BUTNOT. Contrary to the logical operation of NOT, the operation BUTNOT takes two lists and constructs a third list consisting of elements that are present in one but not in the other. This operation is useful when elements of a list are also lists.

Intlist	
Import:	Integer
<b>Constructors for Intlist:</b>	
CreateIL:	$\rightarrow Intlist$
AddI:	$Intlist \times Integer \rightarrow Intlist$
<b>Observers for Intlist:</b>	
HeadI:	$Intlist \rightarrow Integer$
TailI:	$Intlist \rightarrow Intlist$
GetI	$Integer \times Intlist \rightarrow Integer$
OpsI:	$Intlist \times Intlist \rightarrow Intlist$
OpsI = { AND, OR, BUTNOT }	
$\forall il, l1, l2 \in Intlist, i \in integer$	
HeadI(CreateIL) =	Undefined
HeadI(AddI(il, i)) =	If (il = CreateIL) then i else HeadI(il)
TailI(CreateIL) =	CreateIL
TailI(AddI(il, i)) =	If(il = CreateIL) then CreateIL
	else AddI(Tail(il), i)
GetI(k, CreateIL) =	Null
GetI(k, AddI(il, i)) =	If ( k = 1) then HeadI(AddI(il, i))
	else GetI(k - 1, TailI(AddI(il, i)))
AND(l1, l2) =	$\{ l3 \mid \forall x \in l1 \wedge x \in l2 \rightarrow x \in l3 \}$
OR(l1, l2) =	$\{ l3 \mid \forall x \in l1 \wedge y \in l2 \rightarrow x, y \in l3 \}$
BUTNOT(l1, l2) =	$\{ l3 \mid \forall x \in l1 \wedge x \ni l2 \rightarrow x \in l3 \}$

Figure 2. Trait 2: Integer List

**A.3. Specification of an Element Information** An element information (called eleminfo) is a tuple  $\{string, Index\}$ . The string could be either an author name, title, or a call number, depending on which element list this information belongs to. A sequence of eleminfo is a element list. A search for a library resource initiates a search into this element list depending on the search request and returns an index into the booklist where information pertaining to a book is stored. Figure 3 illustrates the algebraic specification of an eleminfo with the necessary axioms. Functions that are required to manipulate eleminfo include: (i) **Name**: This function extracts the string from eleminfo. (ii) **Index**: This function extracts the index from eleminfo.

Eleminfo	
Import:	String, Integer
<b>Constructors for Eleminfo:</b>	
CreateEI:	$String \times Integer \rightarrow Eleminfo$
<b>Observers for Eleminfo:</b>	

Figure 3. Trait 3: Element Information (Continued) . . .

Eleminfo	
Name:	$Eleminfo \rightarrow String$
Index:	$Eleminfo \rightarrow Integer$
$\forall s \in String, i \in Integer$	
Name(CreateEI(s, i)) =	s
Index(CreateEI(s, i)) =	i

Figure 3. Trait 3: Element Information

**A.4. Specification of an Element list** The element list (called elemList) is an ordered sequence of element information. The elemList is the starting point for the specification of the LIMS. A search for a particular book is initiated by finding an appropriate shorter list of integers which contains the index information of all elemInfo that match the search string in the search request. This may be a list of books written by one author, or a list of books with the same title. Figure 4 gives the algebraic specification of elemList with the necessary axioms. Since the call number of a book is unique, a search on a call number returns a single element list. Operations that are required on the elemList include: (i) **LengthE**: This function is used to find the length of the element list, (ii) **PosE**: This function returns the position of a string in the element list, the first position being position number one. When the list is empty, PosE returns  $+\infty$ , (iii) **ValE**: Given the position of an element in the list, find the elemInfo at that position, (iv) **AddE**: This is a function to add an elemInfo into the elemList in an appropriate position, so that the ordering of the elemList is maintained, (v) **RemoveE**: Given a bookid k in an element list, this function removes an elemInfo e, which has Index(e) = k, from the elemList and returns the updated elemList, (vi) **FindE**: The input string for this function might be the original input string or a shortened input string created by SearchE. FindE searches for a match from the position specified by SearchE in the elemList and creates a list of book ids that correspond to all the matches in the list to the input string. This is done by recursively calling FindE until a string that does not match is encountered, (vii) **SearchE**: SearchE looks for a string match within an element list to the given string. If a match is not found the last character of the search string is deleted and SearchE called recursively until a match is found. Once a match is found then SearchE calls FindE to create a list of indexes that match the input string.

ElemList	
Import:	ElemInfo, Integer, String, IntList
<b>Constructors for ElemList:</b>	
CreateEL:	$\rightarrow ElemList$
ConsE:	$ElemInfo \times ElemList \rightarrow ElemList$
<b>Observers for ElemList:</b>	
LengthE:	$ElemList \rightarrow Integer$
PosE:	$String \times ElemList \rightarrow Integer$
ValE:	$Integer \times ElemList \rightarrow ElemInfo$
AddE:	$ElemInfo \times ElemList \rightarrow ElemList$
RemoveE:	$Integer \times ElemList \rightarrow ElemList$

Figure 4. Trait 4: Element List (Continued) . . .

Elemlist	
FindE:	$String \times Elemlist \times Intlist$ $X Integer \rightarrow Intlist$
SearchE:	$String \times Elemlist \rightarrow Intlist$
$\forall l \in Elemlist, il \in Intlist$ $i, k \in Integer, s \in String$ $e, e' \in Eleminfo$ LengthE(CreateEL) = 0 LengthE(ConsE(e, l)) = if (e $\exists$ l) then LengthE(l) + 1 PosE(s, CreateEL) = + $\infty$ PosE(s, ConsE(e, l)) = If (match(name(e), s)) then 1 else PosE(s, l) + 1 ValE(i, CreateEL) = Undefined ValE(i, ConsE(e, l)) = If (i = 1) then e else ValE(i - 1, l) AddE(e, ConsE(e', l)) = If (greater(name(e'), name(e))) then ConsE(e, ConsE(e', l)) else ConsE(e', AddE(e, l)) AddE(e, CreateEL) = ConsE(e, CreateEL) RemoveE(k, ConsE(e, l)) = If (k = Index(e)) then l else ConsE(e, RemoveE(k, l)) RemoveE(k, CreateEL) = CreateEL FindE(s, l, il, k) = If (match(Name(ValE(k+1)), s) then FindE(s, l, AddI(il, Index(ValE(k)), k+ 1) else AddI(il, Index(ValE(k))) SearchE(CreateS, l) = Null SearchE(s, l) = If (PosE(s, l) $\leq$ LengthE(l)) then FindE(s, l, CreateIL, PosE(s, l)) else SearchE(DeleteC(s), l)	

Figure 4. Trait 4: Element List

**A.5. Specification of a Book Information** Information pertaining to a particular book in the LIMS is stored as a book information (bookinfo). It consists of a book identification (bookid), the author name, the title of the book, the call number of the book, detailed information regarding the book (the fourth string in the signature) and the status of the book (called copy) in the library (whether it has been checked out by some user or it is available).

Every book in the library can be referenced by a unique bookid and this is the key used in searching for a particular book in the library. Figure 5 gives the algebraic specification of bookinfo and the required axioms for it. Operations that need to be defined on bookinfo include: (i) **Author**: To retrieve the author name from the given information. (ii) **Title**: To retrieve the title of the book from bookinfo. (iii) **Callnum**: To retrieve the call number from bookinfo. (iv) **Bookid**: To retrieve the bookid from bookinfo. (v) **Copy**: To retrieve the status of the book from bookinfo.

Bookinfo	
Import:	String, Integer
<b>Constructors for Bookinfo:</b>	
CreateBI:	$Integer \times String \times String \times String$ $X String \times Integer \rightarrow Bookinfo$

Figure 5. Trait 5: Book Information (Continued) . . .

Bookinfo	
<b>Observers for Bookinfo:</b>	
Author:	$Bookinfo \rightarrow String$
Title:	$Bookinfo \rightarrow String$
Callnum:	$Bookinfo \rightarrow String$
Bookid:	$Bookinfo \rightarrow Integer$
Copy:	$Bookinfo \rightarrow Integer$
$\forall n, k \in Integer, a, t, c, I \in String$	
Author(CreateBI (k, a, t, c, I, n)) =	a
Title(CreateBI (k, a, t, c, I, n)) =	t
Callnum(CreateBI (k, a, t, c, I, n)) =	c
Bookid(CreateBI (k, a, t, c, I, n)) =	k
copy(CreateBI (k, a, t, c, I, n)) =	n

**Figure 5. Trait 5: Book Information**

**A.6. Specification of a Book List** The specification of a book list (booklist) is the most important part of the specification of the LIMS system. The booklist is actually the database of information regarding all the books in the library. Figure 6 gives the algebraic specification of a booklist and the necessary axioms. Functions that can be defined on the booklist include: (i) **AddB**: This function is a constructor that is used similar to a Cons function used in the previous traits. It is used to add a new bookinfo to the booklist, (ii) **RemoveB**: Given a bookid, this function is used to remove a book from the booklist, (iii) **UpdateB**: This function is used in updating the booklist with information pertaining to the status of a book, while it is either checked in or checked out, (iv) **SearchB**: This function is used to Search for a particular book in the booklist given a bookid.

Booklist	
Import:	Bookinfo
<b>Constructors for Booklist:</b>	
CreateBL:	$\rightarrow Booklist$
AddB:	$Bookinfo \times Booklist \rightarrow Booklist$
<b>Observers for Booklist:</b>	
RemoveB:	$Integer \times Booklist \rightarrow Booklist$
SearchB:	$Integer \times Booklist \rightarrow Bookinfo$
UpdateB:	$Integer \times Integer \times Booklist \rightarrow Booklist$
$\forall n, k \in Integer, l \in Booklist$	
$b \in Bookinfo$	
RemoveB(k, CreateBL) =	CreateBL
RemoveB(k, AddB(b, l)) =	If (k = bookid(b) $\wedge$ Copy(b) = 1) then l else AddB(b, RemoveB(k, l))
SearchB(k, CreateBL) =	Null
SearchB(k, AddB(b, l)) =	If (k = Bookid(b)) then b else SearchB(k, l)
UpdateB(n, k, CreateBL) =	CreateBL
UpdateB(n, k, AddB(b, l)) =	If (k = Bookid(b)) then AddB(CreateBI(Bookid(b), Author(b), Title(b), Callnum(b), n), l) else AddB(b, UpdateB(n, k, l))

**Figure 6. Trait 6: Book List**

**A.7. Specification of a Save List** The save list (savelist) contains information pertaining to a particular search operation. Information retrieved during a search operation may be saved in a saved list. This information is of type *intlist*, that is, it is a list of integers that give the index numbers of books in the booklist which match a particular search term. Figure 7 gives the algebraic specification of a savelist and the necessary axioms. Operations that need to be defined for manipulation of the savelist include: (i) **HeadV**: This function returns the head (first entry) of the savelist, (ii) **TailV**: This function removes the first element from the savelist, (iii) **LengthV**: This function is used in counting the number of elements in the savelist, (iv) **RemoveV**: This function is used to remove an element from the saved list from the given position and return the updated savelist, (v) **GetV**: This function returns the element from a given position in the savelist.

Savelist	
Import:	Intlist, Integer
<b>Constructors for Savelist:</b>	
CreateSL:	$\rightarrow Savelist$
AddV:	$Savelist \times Intlist \rightarrow Savelist$
<b>Observers for Savelist:</b>	
HeadV:	$Savelist \rightarrow Intlist$
TailV:	$Savelist \rightarrow Savelist$
LengthV:	$Savelist \rightarrow Integer$
RemoveV:	$Integer \times Savelist \rightarrow Savelist$
GetV:	$Integer \times Savelist \rightarrow Intlist$
$\forall i \in Integer, l \in Savelist$  $e \in Intlist$ HeadV(CreateSL) = Undefined HeadV(AddV(l, e)) = If (l = CreateSL) then e  else HeadV(l) TailV(CreateSL) = CreateSL TailV(AddV(l, e)) = If (l = CreateSL) then CreateSL  else AddV(TailV(l), e) LengthV(CreateSL) = 0 LengthV(AddV(l, e)) = LengthV(l) + 1 RemoveV(i, CreateSL) = CreateSL RemoveV(i, AddV(l, e)) = If (i = LengthV(AddV(l, e))) then l  else AddV(RemoveV(i, l), e) GetV(i, CreateSL) = Undefined GetV(i, AddV(l, e)) = If (i = 1) then HeadV(AddV(l, e))  else GetV(i - 1, TailV(AddV(l, e)))	

**Figure 7. Trait 7: Save List**

**A.8. Specification of a State** The state is the LIMS user interface specification. All user specified commands are defined by the state. The state of the LIMS system consists of a booklist, three element lists and a savelist. The operations that need to be performed by the system are defined in the state. These operations are:

1. **Addbook**: This function is used for adding a book to the library and updating the LIMS system with

information regarding the new arrival. It can also be viewed as a constructor for the LIMS booklist.

2. **Removebook:** Given a bookid k, his function is used to remove a book from the booklist.
3. **BookL:** Given a state, this function returns the current booklist.
4. **AuthorL:** Given a state, this function returns the current author list.
5. **TitleL:** Given a state, this function returns the current title list.
6. **CallnumL:** Given a state, this function returns the current callnum list.
7. **SaveL:** Given a state, this function returns the current savelist.
8. **Checkout:** This function allows the user to check out a book from the library and updates the booklist regarding the status of the book accordingly.
9. **Checkin:** This is a function pertaining to the operation of the library which updates the status of book, when the book is returned by the user.
10. **SearchAT:** Searching the LIMS system for a particular book in the library, the search being carried out by the author name. This function returns a list that might be zero or more elements long and which contains the books id's of all successful searches.
11. **SearchTL:** Searching the LIMS system for a particular book in the library, the search being carried out by the book title name. This function returns a list that might be zero or more elements long and which contains the books id's of all successful searches.
12. **SearchCN:** Searching the LIMS system for a particular book in the library, the search being carried out by the call number of the book. This function returns a list that is one element in length in case there is only one copy of the book in the library. Otherwise it returns a list of book id's that have the same call number and represent multiple copies of the book in the library.
13. **Extract:** This function is used to extract an element from a given position in the savelist. The element in the savelist is an integer list of bookid numbers.
14. **Select:** This function is used in selecting a particular bookid from the list that is generated by SearchAT, SearchTL, or SearchCN. It is also used for returning a bookid from a integer list, which is generated by the Extract function that operates on a savelist.
15. **Save:** This function is used to add the smaller list generated by functions SearchAT, SearchTL, and SearchCN to the savelist.
16. **Delete:** This function is used to delete a element from a given position in the savelist.
17. **Oper:** This function specifies operations on the savelist. All the operations specified involve two elements (denoted by means of their positions in the savelist) of the savelist and result in a new element being added to the savelist. Operations can be the logical operations defined in *intlist*.

Some of the above functions are actually compositions of two or more functions that perform operations on some state variable, the variable being the booklist, savelist or one of the elemlists, or operations on the incoming information from the outside environment. For example, there are functions that retrieve certain elements of information from the information that is supplied while adding a book to the library and creating new pieces of element information that is appropriate to update other variables of the state. Figure 8 gives the algebraic specification of the state and defines axioms that relate to the specified operations of the LIMS system, that is, search by author name, search by title, and search by call number for a particular book in the library.

State	
Import:	Booklist, Elemlist, Savelist, Intlist
<b>Constructors for State:</b>	
CreateST:	$Booklist \times Elemlist \times Elemlist$ $\times Elemlist \times Savelist \rightarrow State$
<b>Observers for State:</b>	
BookL:	$State \rightarrow Booklist$
AuthorL:	$State \rightarrow Elemlist$
TitleL:	$State \rightarrow Elemlist$
CallnumL:	$State \rightarrow Elemlist$
Savel:	$State \rightarrow Savelist$
Addbook:	$Bookinfo \times State \rightarrow State$
Removebook:	$Integer \times State \rightarrow State$
Checkin:	$Integer \times State \rightarrow State$
Checkout:	$Integer \times State \rightarrow State$
SearchAT:	$String \times State \rightarrow Intlist$
SearchTL:	$String \times State \rightarrow Intlist$
SearchCN:	$String \times State \rightarrow Intlist$
Select:	$Integer \times Intlist \times State \rightarrow Bookinfo$
Save:	$Intlist \times State \rightarrow Savelist$
Delete:	$Integer \times State \rightarrow State$
Extract:	$Integer \times State \rightarrow Intlist$
Oper:	$Integer \times Integer \times State \rightarrow State$
$\forall b, b1, b2 \in Bookinfo, st \in State$	
$i, j, k \in Integer, il \in Intlist$	
BookL(CreateST(bl, al, tl, cl, sl)) =	bl, where bl is the booklist
AuthorL(CreateST(bl, al, tl, cl, sl)) =	al, where al is the author list
TitleL(CreateST(bl, al, tl, cl, sl)) =	tl, where tl is the title list
CallnumL(CreateST(bl, al, tl, cl, sl)) =	cl, where cl is the callnum list
Savel(CreateST(bl, al, tl, cl, sl)) =	sl, where sl is the savelist
Addbook(b, CreateST(CreateBL, CreateEL, CreateEL, CreateEL, CreateSL)) =	CreateST( AddB(b, CreateBL), AddE(CreateEI(Author(b), Bookid(b)), CreateEL), AddE(CreateEI(Title(b), Bookid(b)), CreateEL), AddE(CreateEI(Callnum(b), Bookid(b)), CreateEL), CreateSL)

Figure 8. Trait 8: State (Continued) . . .

State	
Addbook(b, CreateST(bl, al, tl, cl, sl)) =	If (SearchB(Bookid(b), bl) = Null) then CreateST( AddB(b, bl), AddE(CreateEI(Author(b), Bookid(b)), al), AddE(CreateEI(Title(b), Bookid(b)), tl), AddE(CreateEI(Callnum(b), Bookid(b)), cl), sl)
Removebook(k, CreateST(CreateBL, CreateEL, CreateEL, CreateEL, CreateSL)) =	CreateST(CreateBL, CreateEL, CreateEL, CreateEL, CreateSL)
Removebook(k, CreateST(bl, al, tl, cl, sl)) =	CreateST( RemoveB(k, bl), RemoveE(k, al), RemoveE(k, tl), RemoveE(k, cl), sl)
Checkin(k, CreateST(bl, al, tl, cl, sl)) =	If(SearchB(k, bl) != Null $\wedge$ Copy(SearchB(k, bl)) = 0) then CreateST(UpdateB(1, k, bl), al, tl, cl, sl)
Checkout(k, CreateST(bl, al, tl, cl, sl)) =	If(SearchB(k, bl) != Null $\wedge$ Copy(SearchB(k, bl)) = 1) then CreateST(UpdateB(0, k, bl), al, tl, cl, sl)
SearchAT(s, CreateST(bl, al, tl, cl, sl)) =	SearchE(s, al)
SearchTL(s, CreateST(bl, al, tl, cl, sl)) =	SearchE(s, tl)
SearchCN(s, CreateST(bl, al, tl, cl, sl)) =	SearchE(s, cl)
Select(k, il, CreateST(bl, al, tl, cl, sl)) =	SearchB(GetI(k, il), bl)
Save(l, CreateST(CreateBL, CreateEL, CreateEL, CreateEL, CreateSL)) =	CreateST(CreateBL, CreateEL, CreateEL, CreateEL, AddV(CreateSL, l))
Save(l, CreateST(bl, al, tl, cl, sl)) =	CreateST(bl, al, tl, cl, AddV(sl, l))
Delete(k, CreateST(CreateBL, CreateEL, CreateEL, CreateEL, CreateSL)) =	CreateST(CreateBL, CreateEL, CreateEL, CreateEL, CreateSL)
Delete(k, CreateST(bl, al, tl, cl, sl)) =	CreateST(bl, al, tl, cl, RemoveV(k, sl))

**Figure 8. Trait 8: State (Continued) . . .**

State	
Extract(k, CreateST(bl, al, tl, cl, sl)) =	GetV(k, sl)
Oper(i, j, CreateST(bl, al, tl, cl, sl)) =	CreateST(bl, al, tl, cl, AddV(sl, OpsI(GetV(i, sl), GetV(j, sl))))

Figure 8. Trait 8: State

## B. Properties for Correct System Operation

The easy verification of certain basic system properties is the most important advantage of using formal specification techniques. Some such properties include: (i) A book not in the LIMS database cannot be checked out, (ii) A book not in the LIMS database cannot be checked in, (iii) A book not in the LIMS database cannot be removed, (iv) A book cannot be checked out if it is already registered as checked out, and, (v) A book cannot be checked in if it is already registered as checked in. From the state specifications, it is seen that an update of state information occurs only on two conditions: (i) The book is in the LIMS database and this is reflected by the function SearchB which searches for the book in the booklist, and, (ii) If the book is present then the function Copy checks to verify if the book is present (indicated by an integer number 1) in the library, meaning it is not checked out. Using this information, the proof for the first property is as below<sup>3</sup>: The first property can be fomrally stated as **Statement:**  $Checkout(k, Removebook(k, st)) = Removebook(k, st)$

The proof of this property is derived from Figure 8, especially the axiom for Checkout (k, Addbook(b, st)), where

$$\begin{aligned}
& Checkout(k, Removebook(b, st)) ; (k = Bookid(b)) \wedge b \in bl \\
& \Rightarrow Checkout(k, st') = st' ; \sim \exists b \in bl : bookid(b) = k \\
& \Leftrightarrow Removebook(k, st) = st'
\end{aligned}$$

The function  $SearchB(k, BookL(Addbook(b, st))) = Null$  within the if condition returns a 'False' and hence the state is not updated. Given the book id, SearchB recursively searches for a book from the booklist. If the book is present in the booklist it returns the bookinfo for the particular bookid, otherwise it returns Null. This implies that checkout does not call CreateST to update the checkout information for the particular bookid. ■

## III. Conclusions

In this paper, the algebraic specification of a LIMS has been derived. The specifications of the basic constructs, and the verification of the basic assumptions about the correct LIMS operations has been presented. The specification has been used as a basis for the implementation of an object-oriented LIMS.

<sup>3</sup> The proofs for other properties can be similarly derived

Although the example involved is relatively small and deterministic, the LIMS illustrates the simplicity and appropriateness of using formal techniques in the specification of information management systems. The use of such techniques enhance the design and simplify the maintenance of information management systems and serve to improve the quality of modern information management systems.

### **References**

1. John V. Guttag, James J. Horning, Jeannette M. Wing, "The Larch Family of Specification Languages", *IEEE Software*, pp 24-36, September 1985.
2. John Rushby, "Formal Methods and their Role in the Certification of Critical Systems", *Technical Report CSL-95-1, Computer Science Laboratory, SRI International, March 1995*.
3. Ralph Stair, "Principles of Information Systems", *Boyd & Fraser Publishing Company, 1992*.
4. Bernard Sufrin, "Formal Specification of a Display-Oriented Text Editor", *Science of Computer Programming 1, North Holland Publishing Company, 1982*.
5. Carrol Morgan, Bernard Sufrin, "Specification of the Unix Filing System", *IEEE Transactions on Software Engineering, Vol. SE-10, No. 2, March 1990*.