5.8.1 Generated Tests ......................................................... 17
5.9 Testing the Copy File Operation ........................................ 18
  5.9.1 Generated Tests ...................................................... 18
5.10 Testing the ChdirDown Operation ...................................... 19
  5.10.1 Generated Tests ...................................................... 19
5.11 Testing the ChdirUp Operation .......................................... 20
  5.11.1 Generated Tests ...................................................... 20
5.12 Testing the MoveFile Operation ........................................ 21
5.13 Testing the List Operation ............................................. 21
  5.13.1 Generated Tests ...................................................... 21

6 Algebraic MiStix Specifications ........................................ 22
Contents

1 Introduction 2

2 Original Informal Mix Specifications 2

3 Revised Informal Mix Specifications 3

4 Z Mix Specifications 5
   4.1 State Description ........................................ 5
   4.2 The Initial State ......................................... 5
   4.3 The GetNode Question ................................... 6
   4.4 The Hetary Question .................................... 6
   4.5 The Get Directory Question .............................. 7
   4.6 The Where Question ..................................... 8
   4.7 The Getfile Question .................................... 8
   4.8 The Hetary Question .................................... 8
   4.9 The File Qry Questions .................................. 9
   4.10 The File Qry Questions ................................. 11
   4.11 The List Qry ........................................... 11
   4.12 The Logfile Qry ......................................... 11

5 Zard Category-Partition Based Test Specifications 11
   5.1 Testing the GetNode Question ......................... 12
      5.1.1 Generated Tests .................................. 12
   5.2 The Hetary Question .................................. 13
      5.2.1 Generated Tests .................................. 13
   5.3 Testing the File Qry ................................... 14
      5.3.1 Generated Tests .................................. 14
   5.4 Testing the Up Qry ...................................... 15
      5.4.1 Generated Tests .................................. 15
   5.5 Testing the Root Qry ................................... 15
      5.5.1 Generated Tests .................................. 16
   5.6 Testing the Move Qry ................................... 16
      5.6.1 Generated Tests .................................. 16
   5.7 Testing the Getfile Qry ................................ 16
      5.7.1 Generated Tests .................................. 17
   5.8 Testing the Hetary Qry ................................ 17
8. ListFiles (NewDT, NewCW) = Empty
9. ListFiles (CreateFile (DT, c1, s), c2) = if (Where (c1) = Where (c2))
   then if Member (ListDirs (DT, c1), s)
       then Add (ListFiles (DT, c2), s)
       else ListFiles (DT, c2)
   else ListFiles (DT, c2)
10. ListFiles (DeleteFile (DT, c1, s), c2) = if (Where (c1) = Where (c2))
    then Remove (ListFiles (DT, c2), s)
    else ListFiles (DT, c2)
11. ListFiles (CopyFile (DT, c1, s1, s2), c2) = if (Where (c1) = Where (c2))
    then Add (ListFiles (DT, c2), s2)
    else ListFiles (DT, c2)
12. ListFiles (CopyFileDown (DT, c1, s1, s2), c2) = if (Where (c2) = Where (Down (c1, s2)))
    then Add (ListFiles (DT, c2), s1)
    else ListFiles (DT, c2)
13. ListFiles (CopyFileUp (DT, c1, s), c2) = if (Where (c2) = Where (Up (c1)))
    then Add (ListFiles (DT, c2), s)
    else ListFiles (DT, c2)
14. ListFiles (MoveFile (DT, c1, s1, s2), c2) = if (Where (c1) = Where (c2))
    then Add (Remove (ListFiles (DT, c2), s1), s2)
    else ListFiles (Remove (DT, c2), s1)
15. ListFiles (MoveFileDown (DT, c1, s1, s2), c2) = if (Where (c2) = Where (Down (c1, s2)))
    then Add (ListFiles (DT, c2), s1)
    else ListFiles (DT, c2)
16. ListFiles (MoveFileUp (DT, c1, s), c2) = if (Where (c2) = Where (Up (c1)))
    then Add (ListFiles (DT, c2), s)
    else ListFiles (DT, c2)

References

    Shriver publication

4 LEqual (Append (SL1, s1), Append (SL2, s2)) = (s1 = s2 \land LEqual (SL1, SL2))
5 DelTail (Null) = Null
6 DelTail (Append (SL1, s1)) = SL1
7. Tail (Null) = '/' -- Odd indeed!
8. Tail (Append (SL1, s1)) = s1

The MiSTIX specification is below. It uses various objects and operations from the previous specification. Notably, Where is defined to return a list of strings (SIMIL) and the List functions return sets of strings (the first is the set of directories, and the second is the set of files).

DTREE_SPEC: trait

CWD : StrList

introduces
NewDTree : \rightarrow DTREE
NewCWD : \rightarrow CWD
CreateDir : DTREE, CWD, String \rightarrow DTREE
DeleteDir : DTREE, CWD, String \rightarrow DTREE
Down : DTREE, CWD, String \rightarrow CWD
Up : CWD \rightarrow CWD
Root : CWD \rightarrow CWD
Where : CWD \rightarrow StrList
CreateFile : DTREE, CWD, String \rightarrow DTREE
DeleteFile : DTREE, CWD, String \rightarrow DTREE
CopyFile : DTREE, CWD, String, String \rightarrow DTREE
CopyFileDown : DTREE, CWD, String, String \rightarrow DTREE
CopyFileUp : DTREE, CWD, String \rightarrow DTREE
MoveFile : DTREE, CWD, String, String \rightarrow DTREE
MoveFileDown : DTREE, CWD, String, String \rightarrow DTREE
MoveFileUp : DTREE, CWD, String \rightarrow DTREE
ListDirs : DTREE, CWD \rightarrow StrSet
ListFiles : DTREE, CWD \rightarrow StrSet

contains NewDTree, NewCWD, CreateDir, DeleteDir, Down, Up, Root, Where,
CreateFile, DeleteFile, CopyFile, CopyFileDown, CopyFileUp, MoveFile,
MoveFileDown, MoveFileUp, ListDirs, and ListFiles

so that for all [DT : DTREE; c1, c2 : CWD; s, s1, s2 : String]
1 Where (NewCWD) = Null
2 Where (Down (DT, c, s)) = if (Member (ListDirs (DT, c), s))
   then Append (Where (c, s))
   else Where (c)
3 Where (Up (c)) = if (Where (c) = Null)
   then Null
   else DelTail (Where (c))
4 Where (Root (c)) = Null
5 ListDirs (NewDTree, NewCWD) = Empty
6 ListDirs (CreateDir (DT, c1, s), c2) = if (Where (c1) = Where (c2))
   then if (Member (ListFiles (DT, c1), s))
         then Add (ListDirs (DT, c2), s)
         else ListDirs (DT, c2)
   else ListDirs (DT, c2)
7. ListDirs (DeleteDir (DT, c1, s), c2) = if (Where (c1) = Where (c2))
   then Remove (ListDirs (DT, c2), s)
   else ListDirs (DT, c2)
(([], [], a), a)
(([], [a], a), a)

3 The of dirs
((b), [], a) -- cwd = a impossible, so do this instead,

4 The of cd
((b), [a], a),

Note that files and directories are not forced to be listed simultaneously.

6 Algebraic MiStix Specifications

What start with a specification for a string set and a string list. These will be imported into the eventual
specification for the MiStix file system.

**STRSET_SPEC: tmit**

**introduces**

Empty : StrSet
Add : StrSet, String → StrSet
Member : StrSet, String → Boolean
Remove : StrSet, String → StrSet

**axioms** Empty, Add, Member, and Remove
so that for all [SS: StrSet; s1, s2: String]
1 Member (Empty, s1) = FALSE
2 Member (Add (SS, s2), s1) = if s1 = s2
   then TRUE
   else Member (SS, s1)
3 Remove (Empty, s1) = Empty
4 Remove (Add (SS, s1), s2) = if s1 = s2
   then Remove (SS, s2)
   else Add (Remove (SS, s2), s1)

**STRLIST_SPEC: tmit**

**introduces**

Null : → StrList
Append : StrList, String → StrList
LEqual : StrList, StrList → Boolean
Tail : StrList → String
DelTail : StrList → StrList

**axioms** Null, Append, DelTail, Tail, and LEqual
so that for all [SL1, SL2: StrList; s1, s2: String]
1 LEqual (Null, Null) = TRUE
2 LEqual (SL1, SL2) = LEqual (SL2, SL1)
3 LEqual (Append (SL1, s1), Null) = FALSE
(b, {<c>}, {<e>,<a>}, <a>)

3 Exception 2
(b, {<b>}, {<e>,<a>}, <e>)

4 Exception 3
(b, {<a,b>}, {<e>,<a>,<b>}, <a>)
(b, {<a,b>,<b>}, {<e>,<a>}, <a>)

5 The of files
No test case possible that satisfies precondition

6 The of dirs
No test case possible that satisfies precondition
The of cad
7 No test case possible that satisfies precondition

5.12 Testing the MoveFile Operation

Sure the preconditions for MoveFile, MoveFile Down, and MoveFile Up are identical to that for CopyFile, CopyFile Down, and CopyFile Up, respectively the test inputs should be identical. Test inputs differ in that the source file should no longer exist here.

5.13 Testing the List Operation

Initial Bit:
List

Inputs: None

Environment Variables:

<table>
<thead>
<tr>
<th>files</th>
<th>Path Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirs</td>
<td>Path Name</td>
</tr>
<tr>
<td>cad</td>
<td>Path Name</td>
</tr>
</tbody>
</table>

Categories: The same preconditions, although the presence or absence of files and/or directories in cad clearly seems to be a category.

5.13.1 Generated Tests

Tests are triples of (files, dirs, cad).

1. Base Test Case - All Categories Normal
   (null, null, null)

2. The of files
(b, c, \{<a, b>\}, \{<>\}, <a>)

4 Precondition 3
(b, c, \{<a, b>\}, \{<>\}, <a>, <a, c>, <a, c, b>, <a>)
(b, c, \{<a, b>\}, \{<>\}, <a, <a, c, b>, <a>, <a, c>, <a>)

5 The of files
No test case possible that satisfies precondition

6 The of dirs
No test case possible that satisfies precondition

7 (b, c, \{<b>\}, \{<>\}, <c>, <>)

5.11 Testing the Copy FileUp Operation

Initial lit:
- Copfile

Inputs
- n?: Name

Environment lits:
- fil es: PFull Name
- dir s: PFull Name
- cud: Full Name

Givens:
- Precondition 1
  * Satisfied File if? exists
  * Unsatisfied File if? does not exist
- Precondition 2
  * Satisfied - parent directory exists
  * Unsatisfied - parent directory does not exist
- Precondition 3
  * Satisfied File if? does not exist in parent
  * Unsatisfied Directory exists in parent
  * Unsatisfied File exists in parent

5.11.1 Generated Tests

Tests are families of (n?, fil es, dir s, cud).

1 The Test Gie - All Givens Null
(b, \{<a, b>\}, \{<>\}, <a>)

2 Recursion 1
(b, c, {<a,b>}, {<>},<a>,<a,c>, <a>)
(b, c, {<a,b>,<a,c>}, {<>},<a>, <a>)

4 The of files

No test case possible that satisfies precondition

5 The of dirs

(b, c, {<b>}, {<>}, <>)

The of cud

6 (b, c, {<b>}, {<a>,<>}, <>)

5.10 Testing the CopyFileDown Operation

Initial Init:
CopyFileDown

Inputs
n?: Name
d?: Name

Input Variables:
files : Full Name
dirs : Full Name
cud : Full Name

Covers:

Procedure 1
* Satisfied File if exists
* Unsatisfied File if dirs exist

Procedure 2
* Satisfied Directory d exists
* Unsatisfied Directory d does not exist

Procedure 3
* Satisfied File if does not exist in d'
* Unsatisfied Directory exists in d'
* Unsatisfied File exists in d'

5.10.1 Granted Tests

These are tuples of (n?, d?, files, dirs, cud).

1 Testing All Covers

(b, c, {<a,b>}, {<>},<a>,<a,c>, <a>)

2 Procedure 1

(b, c, {<b>}, {<>},<a>,<a,c>, <a>)

3 Procedure 2
1 The Test Case - All Conditions Satisfied

\((b, \{<a,b>\}, <a>)\)

2 Condition 1

\((b, \{<a,c>\}, <a>)\).

3 The of files

No test case possible that satisfies precondition

4 The of out

\((b, \{<b>\}, <>),\)

5.9 Testing the CopyFile Operation

Initial Init:

Copyfile

Inputs

\(\text{old?}: \text{Name}\)

\(\text{new?}: \text{Name}\)

Variables:

\(\text{files} : \text{PFull Name}\)

\(\text{dirs} : \text{PFull Name}\)

\(\text{cut} : \text{Full Name}\)

Conditions:

Condition 1

\* Satisfied file old exists

\* Unsatisfied file old does not exist

Condition 2

\* Satisfied file new does not exist

\* Unsatisfied directory and new exists

\* Unsatisfied file and new exists

5.9.1 Created Tests

Tests are tuples of \((\text{old?}, \text{new?}, \text{files}, \text{dirs}, \text{cut})\)

1 The Test Case - All Conditions Satisfied

\((b, c, \{<a,b>\}, \{<>\}, \{<a>\}, <a>)\)

2 Condition 1

\((b, c, \{<b>\}, \{<>\}, \{<a>\}, <a>)\)

3 Condition 2
Exercise 1

1. Satisfied file does not exist
2. Unsatisfied directory exists
3. Unsatisfied file exists

5.7.1 Generated Tests

Tests are tuples of (n?, f1e1, d1r1, c1u1).

1. niece file - All Categories
   
   \((c, \{<a,b>\}, \{<>\}, <a>)\)

2. Exercise 1
   
   \((c, \{<a,b>\}, \{<>\}, <a>, <a,c>), <a>)\)
   \((b, \{<a,b>\}, \{<>\}, <a>)\).

3. Type of files
   
   \((b, \{}\), \{<>\}, <a>)\)

4. Type of dirs
   
   \((a, \{<b>\}, \{<>\}, <>\} -- cwd = <a> impossible, so do this instead,\)

5. Type of cmd
   
   \((c, \{<b>\}, \{<>\}, <a>), <>\)

5.8 Testing the Del ete B le O peration

Initial bit:

\(\text{Del ete ble}\)

Inputs

\(n?: \text{Name}\)

Input Variables:

\(\text{file es : } \text{Ful l Name}\)
\(\text{cmd : } \text{Ful l Name}\)

Categories:

\(\text{Exercise 1}\)

\(* \text{Satisfied file exists}\)
\(* \text{Unsatisfied file does not exist}\)

5.8.1 Generated Tests

Tests are tuples of (n?, file es, cmd).

Exercise 1

1. Satisfied file does not exist
2. Unsatisfied directory exists
3. Unsatisfied file exists
5.5.1 Generated Tests

Tests are examples of {}.

1. \texttt{Check Gge - All Glegries Nil}

() \texttt{Check Gge - All Glegries Nil}

Note that this add state of affairs is due to \texttt{Root} being a constant function. Since there are no input parameters, there are no conditions and partitions to drive tests from.

5.6 Testing the \texttt{Where} Operation

Initial lit:

\texttt{Where}

Inputs: Nil

Environment Values:

\texttt{c\_uw : Full Name}

Conditions:

\texttt{Nil based on conditions}

5.6.1 Generated Tests

Tests are examples of \texttt{(c\_uw)}.

1. \texttt{Check Gge - All Glegries Nil}

\texttt{Check Gge - All Glegries Nil}

(<\texttt{a}>)

2. \texttt{The of \texttt{c\_uw}}

\texttt{The of \texttt{c\_uw}}

(<\texttt{a}>),

5.7 Testing the \texttt{CreateFile} Operation

Initial lit:

\texttt{CreateFile}

Inputs

\texttt{n?: Name}

Environment Values:

\texttt{files : P Full Name}

\texttt{dirs : P Full Name}

\texttt{c\_uw : Full Name}

Conditions:
5.4 Testing the Up Operation

Input bit: 
\[ \uparrow \]

Inputs

\emph{N/A}

Input Variables:

- \textit{dirs} : Full Name
- \textit{cwd} : Full Name

Categories:

\textbf{Precondition 1}

- Satisfied - parent directory exists
- Unsatisfied - parent directory does not exist

5.4.1 Generated Tests

Tests are pairs of (\textit{dirs}, \textit{cwd})

\begin{enumerate}
  \item \textbf{Pre Test Case - All Categories Satisfied}
        \[ (\{\langle \rangle, \langle \text{a} \rangle \}, \langle \text{a} \rangle) \]
  \item \textbf{Precondition 1}
        \[ (\{\langle \rangle, \langle \text{a} \rangle \}, \langle \rangle) \]
  \item \textbf{Test for dirs}
        No test case possible that satisfies precondition
  \item \textbf{Test for cwd}
        No test case possible that satisfies precondition
\end{enumerate}

5.5 Testing the Root Operation

Input bit: 
\[ \texttt{Rt} \]

Inputs

\emph{N/A}

Input Variables:

\emph{N/A}

Categories:

\emph{N/A based on preconditions}
(b, \{<a,b,c>\}, \{<>\,<a>,<a,b>\}, \,<a>)
(b, \{<a,c>\}, \{<>\,<a>,<a,b>,<a,b,c>\}, \,<a>)

4 The of dirs

No test case possible that satisfies precondition

5 The of files

(b, \{\}, \{<>\,<a>,<a,b>\}, \,<a>)

6 The of vol

(a, \{<>\}, \{<>\,<a>\}, <>)

5.3 Testing the Down Operation

Input list:
- Dir

Inputs:
- n?: Name

Input Constraints:
- dirs : P'Full' Name
- cwd : Full Name

Categories:
- Prediction 1
  - Satisfied - subdirectory exists
  - Unsatisfied - subdirectory does not exist

5.3.1 Granted Tests

Tests are triples of (n?, dirs, cwd).

1 Be 'Dir Gie - All Categories Null

(b, \{<>\,<a>,<a,b>\}, \,<a>)

2 Prediction 1

(b, \{<>\,<a>\}, \,<a>)

3 The of dirs

No test case possible that satisfies precondition

4 The of vol

(a, \{<>\,<a>\}, <>)
2 **Condition 1**

\[(b, \{<a,c>\}, \{<\>,<a>,<a,b>,<a>\}).\]

\[(b, \{<a,b>\}, \{<\>,<a>\}, <a>).\]

3 **The of files**

\[(b, \{\}, \{<\>,<a>\}, <a>\)]

4 **The of dirs**

\[(a, \{\{b\}\}, \{<\>,<>\}) -- there is no directory <a>, so do this instead,\]

5 **The of cd**

\[(b, \{<a,c>\}, \{<\>,<a>\}, <\>),\]

5.2 **The DeleteDr Operation**

**Initial let:**

```
DeleteDr
```

**Inputs**

\[n?: Name\]

**Explicit Variables:**

\[files : P Full Name\]
\[dirs : P Full Name\]
\[cwd : Full Name\]

**Glosses:**

**Condition 1**

- Satisfied Directory exists
- Unsatisfied Directory does not exist

**Condition 2**

- Satisfied Directory is empty
- Unsatisfied Directory contains a file
- Unsatisfied Directory contains a subdirectory

5.2.1 **Generated Tests**

Tests are tuples of \((n?, dirs, cwd)\).

1 **Test 1**

\[(b, \{<a,c>\}, \{<\>,<a>,<a,b>,<a>\})\]

2 **Condition 1**

\[(b, \{<a,c>\}, \{<\>,<a>\}, <a>).\]

3 **Condition 2**
1. Only methods in all categories apply to all testable units. In each question, the precondition(s) gives rise to one or more additional categories. When categories such that one partition is considered "invalid," and other partitions are other than invalid. Winintak, by default, a test section that holds all categories at the invalid partition and one at a time computes categories over all possible partitions. In such cases, setting a category to invalid will make a partition in another category impossible. In this case, we make a feasible choice. The resulting number of test cases is 1 plus the total number of partitions times the total number of categories. In general, of course, the test engineer is free to define additional result categories as appropriate.

The categories and associated partitions that apply to all questions are as follows. (Note: Partitions values are not complete for given tests...)

Categories:

- The of di r s
  - Notat: \{\langle a \rangle \}, \{\langle \rangle, \langle a \rangle \}, \{\langle b \rangle, \langle a \rangle \}, \{\langle b \rangle, \langle a \rangle \}, \{\langle a, b \rangle \}, \{\langle a, c \rangle \}, \{\langle a \rangle \}, \{\langle a \rangle \}, \{\langle a \rangle \}, \{\langle a \rangle \}, \{\langle b \rangle \}, \{\langle a \rangle \}, \{\langle a \rangle \}, ..
  - Eity: \{\langle \rangle \}

- The of fil e s
  - Notat: \{\langle a \rangle \}, \{\langle b \rangle \}, \{\langle a, b \rangle \}, \{\langle a, c \rangle \}, \{\langle a \rangle , \{\langle b \rangle \}}
  - Eity: \{\}

- The of c u d
  - Notat: \{\langle a \rangle \}
  - Eity: \{\}

5.1 Testing the GeateDr Operation

Input list:

GeateDr

Inputs

n? Name

Immediate Values:

fil e s : P Full Name
di r s : P Full Name
c u d : Full Name

Categories:

- Precondition 1
  - Satisfied Directory does not exist
  - Satisfied Directory exists
  - Satisfied File exists

5.1.1 GeateDr Tests

Tests are for some of \{n?, fil e s, di r s, c u d\}.

1. Be 'Et Gie - All Categories Notat

(b, \langle a, c \rangle\}, \{\langle a \rangle, \{\langle a \rangle\}, \langle a \rangle\}
4.10 The Move File Operations

The move file operations are specified just as the copy file operations are. Indeed we explicitly set the semantics of move to be that of a copy followed by a delete.

\[
\text{Move File} \equiv \text{Copy File Delete File}\{n/?}\{d?\}
\]

\[
\text{Move File Down} \equiv \text{Copy File Down Delete File}
\]

\[
\text{Move File Up} \equiv \text{Copy File Up Delete File}
\]

Since the preconditions of move file operations are identical with those of the corresponding copy file operations, we omit the robust operation specification here.

4.11 The List Operation

The operation List displays all of the files and subdirectories in the current working directory. Subdirectories are to be distinguished with special marks; we ignore that aspect here.

```
\[
\text{List}
\]
\[
\exists \text{File System}
\]
\[
\text{Files}, \text{Dirs} : P \text{ Full Name}
\]
\[
\text{Files}! = \{ f : \text{files} \mid \text{cwd } \subset f \}
\]
\[
\text{Dirs}! = \{ d : \text{dirs} \mid \text{cwd } \subset d \}
\]
```

Would also specify List so as to only include the simple names in the current directory.

```
\[
\text{List}
\]
\[
\exists \text{File System}
\]
\[
\text{Files}, \text{Dirs} : P \text{ Name}
\]
\[
\text{Files}! = \{ f : \text{files} \mid \text{cwd } \preceq \text{front f } \text{ last f} \}
\]
\[
\text{Dirs}! = \{ d : \text{dirs} \mid \text{cwd } \preceq \text{front d } \text{ last d} \}
\]
```

Finally List could easily be split so as to specify files and directories separately. Leave this as an exercise to the reader.

4.12 The LogOff Operation

The logoff operation is technically not part of the abstract data type, and so we do not specify it explicitly.

5 Z and Category-Partition Based Test Specifications

We consider standard categories derived from the specification as a whole. While these categories are for the entire test specification and therefore to the categories as necessary. Certain of these categories appear to arise from design decisions, e.g., the use of "." to represent the parent specification (i.e., in this specification "." has been deliberately suppressed we do not include such a category here, even though such a category is present in our paper.
Copy a file to a subdirectory \( w::gt \):

\[
\begin{align*}
\text{Copy File Down} & \quad \Delta \text{File System} \\
n?, d?: \text{Name} \\
cwd \cap \{ n? \} \in \text{files} \\
cwd \cap \{ d? \} \in \text{dirs} \\
cwd \cap \{ d?, n? \} \notin \text{files} \cup \text{dirs} \\
\text{files} = \text{files} \cup \text{cwd} \cap \{ d? \ \text{n?} \}
\end{align*}
\]

Handle the case where the new file cannot be created in the subdirectory \( w::gt \):

\[
\begin{align*}
\text{Directory Or File Already Exists In Subdirectory} & \quad \exists \text{File System} \\
n?, d?: \text{Name} \\
\text{report!}: \text{Message} \\
cwd \cap \{ d?, n? \} \in \text{files} \cup \text{dirs} \\
\text{report!} = \text{Directory or file already exists in subdirectory}
\end{align*}
\]

The robust specification is:

\[
\begin{align*}
\text{RCopy File Down} \models & \text{Copy File Down} \land \text{Ok} \\
\lor & \text{File Does Not Exist} \\
\lor & \text{Directory Or File Already Exists In Subdirectory}
\end{align*}
\]

Copy a file to the parent directory \( w::gt \):

\[
\begin{align*}
\text{Copy File Up} & \quad \Delta \text{File System} \\
n?: \text{Name} \\
cwd \cap \{ n? \} \in \text{files} \\
cwd \neq \emptyset \\
f \text{ront c}wd \cap \{ n? \} \in \text{files} \cup \text{dirs} \\
\text{files} = \text{files} \cup \text{front c}wd \cap \{ n? \}
\end{align*}
\]

Handle the case where the new file cannot be created in the parent directory \( w::gt \):

\[
\begin{align*}
\text{Directory Or File Already Exists In Parent} & \quad \exists \text{File System} \\
n?: \text{Name} \\
\text{report!}: \text{Message} \\
f \text{ront c}wd \cap \{ n? \} \in \text{files} \cup \text{dirs} \\
\text{report!} = \text{Directory or file already exists in parent}
\end{align*}
\]

The robust specification is:

\[
\begin{align*}
\text{RCopy File Up} \models & \text{Copy File Up} \land \text{Ok} \\
\lor & \text{File Does Not Exist} \\
\lor & \text{Parent Of Root Does Not Exist} \\
\lor & \text{Directory Or File Already Exists In Parent}
\end{align*}
\]
\textbf{Delete File}

\textbf{Filename}
\textit{n? : Name}

\texttt{cwd (\textbackslash n?) \in \textit{files}}

\texttt{files \textbackslash \{ \texttt{cwd (\textbackslash n?) \} }}

In the case that a file does not exist, \texttt{File Does Not Exist}:

\textbf{File Does Not Exist}

\textbf{Filename}
\textit{n? : Name}

\texttt{report! : Message}

\texttt{cwd (\textbackslash n?) \notin \textit{files}}

\texttt{report! = \textit{File does not exist}}

To robust delete file command \texttt{Delete File}, is:

\texttt{Delete File \equiv \textit{Delete File} \land \textit{Ok} \lor \textit{File Does Not Exist}}

\subsection*{4.9 The File Copy Operations}

To change directory functions are:

\begin{itemize}
  \item \texttt{Copy File} - copy a file to another file in \textit{the same} directory
  \item \texttt{Copy File Down} - copy a file to a subdirectory
  \item \texttt{Copy File Up} - copy a file to the parent of the current working directory if it exists
\end{itemize}

There are two inputs to \texttt{Copy File}: \textit{old?}, which is the name of the file to be copied, and \textit{new?}, which is the name of the newfile. There are two inputs to \texttt{Copy File Down}, \textit{n?}, which is the name of the file to be copied, and \textit{d?}, which is the name of the subdirectory into which the file is to be copied. There is one input to \texttt{Copy File Up}, \textit{n?}, which is the name of the file to be copied. (Note: in this specification, file names are considered, but file contents are ignored.)

\texttt{Copy File} locally \texttt{wgt}:

\textbf{Copy File}

\textbf{Filename}
\textit{new? , old? : Name}

\texttt{cwd (\textbackslash old?) \in \textit{files}}

\texttt{cwd (\textbackslash new?) \notin \textit{files \cup dirs}}

\texttt{files = \textit{files} \cup \texttt{cwd (\textbackslash new?)}}

To robust specification is:

\texttt{Copy File \equiv \textit{Copy File} \land \textit{Ok}}

\texttt{\lor \textit{File Does Not Exist}[old?/n?]}

\texttt{\lor \textit{Directory Or File Already Exists}[new?/n?]}

9
4.6 The Where Operation

The operation Where prints the complete name of the current working directory.

\[
\text{Where} \\
\text{FileSystem} \\
\text{FullName}! : \text{FullName} \\
\text{FullName}! = \text{cwd}
\]

Since Where is total, RWhere is

\[
R\text{Where} \equiv \text{Where} \land \text{Ok}
\]

4.7 The CreateFile Operation

The operation CreateFile creates a new file with the (simple) name \( n \) in the current working directory.

\[
\text{CreateFile} \\
\text{FileSystem} \\
\text{Name} \mapsto \text{Status} \mapsto \text{Ok} \\
\text{cwd} \mapsto \text{files} \\
\text{files} = \text{files} \cup \{ \text{cwd} \mapsto \text{Name}\}
\]

Since create file \( \text{Ok} \), RCreateFile is:

\[
R\text{CreateFile} \equiv \text{CreateFile} \land \text{Ok} \\
\lor \text{Directory Or File Already Exists}
\]

4.8 The DeleteFile Operation

The operation DeleteFile deletes a file from the current working directory.
In the case that a subdirectory is not empty we define

\[ DirectoryNotEmpty \]

\[ FileSystem \]
\[ n?: Name \]
\[ report!: Message \]

\[ \exists f : files \cup dirs \cdot c u d \ n? \subseteq f \]
\[ report! = \text{Directory not empty} \]

The robust delete directory command \texttt{RDeleteDir}, is

\[ RDeleteDir \subseteq \text{Delete Dir} \wedge Ok \]
\[ \lor \text{Directory Does Not Exist} \]
\[ \lor \text{Directory Not Empty} \]

4.5 The Change Directory Operations

The change directory operations are:

- \texttt{Down} - change to a subdirectory in the current working directory
- \texttt{Up} - change to the parent of the current working directory if it exists.
- \texttt{Root} - change to the root directory

For changing to a child directory we specify

\[ Down \]

\[ FileSystem \]
\[ n?: Name \]
\[ c u d \ n? \subseteq c u d \]
\[ \text{cwd'} = \text{cwd} \cap \n? \]

The robust change directory to child command \texttt{RDown}, is

\[ RDown \subseteq \text{Down} \wedge Ok \]
\[ \lor \text{Directory Does Not Exist} \]

For changing to the parent directory we specify

\[ Up \]

\[ FileSystem \]
\[ \text{cwd} \neq \emptyset \]
\[ \text{cwd'} = \text{front} \text{ cwd} \]

\texttt{Up} has the precondition that \texttt{cwd} is not root or prior to the operation

\[ ParentOfRootDoesNotExist \]

\[ FileSystem \]
\[ report!: Message \]
\[ \text{cwd} = \emptyset \]
\[ report! = \text{Parent of root does not exist} \]
4.3 The CreateDir Operation

The operation \texttt{CreateDir} creates a new directory with the (single) name \texttt{n} in the current working directory.

\begin{verbatim}
CreateDir
\end{verbatim}

\begin{verbatim}
FileSystem
n?: Name

\texttt{cwd} \texttt{\langle n\rangle} \in \texttt{dirs} \cup \texttt{files}

\texttt{dirs} = \texttt{dirs} \cup \{ \texttt{cwd} \texttt{\langle n\rangle} \}
\end{verbatim}

In the case that a directory or file named \texttt{n} already exists, we define

\begin{verbatim}
DirectoryOrFileAlreadyExists
\end{verbatim}

\begin{verbatim}
FileSystem
n?: Name
report!: Message

\texttt{cwd} \texttt{\langle n\rangle} \in \texttt{dirs} \cup \texttt{files}

report! = \texttt{Directory or File already exists}
\end{verbatim}

We follow the notation of Miller [2] in defining the \texttt{Ok} report:

\begin{verbatim}
Ok
\end{verbatim}

\begin{verbatim}
report!: Message

report! = \texttt{Ok}
\end{verbatim}

Finally, we define the rest create directory command, \texttt{RCreateDir}, as:

\begin{verbatim}
RCreateDir \equiv \texttt{CreateDir} \land \texttt{Ok}
\end{verbatim}

\begin{verbatim}
\lor \texttt{DirectoryOrFileAlreadyExists}
\end{verbatim}

4.4 The DeleteDir Operation

The operation \texttt{DeleteDir} deletes an empty subdirectory in the current working directory.

\begin{verbatim}
DeleteDir
\end{verbatim}

\begin{verbatim}
FileSystem
n?: Name

\texttt{cwd} \texttt{\langle n\rangle} \in \texttt{dirs}

\neg \exists f : \texttt{files} \cup \texttt{dirs} \bullet \texttt{cwd}(n) \subset f

\texttt{dirs} = \texttt{dirs} \setminus \{ \texttt{cwd} \texttt{\langle n\rangle} \}
\end{verbatim}

In the case that a subdirectory does not exist, we define

\begin{verbatim}
DirectoryDoesNotExist
\end{verbatim}

\begin{verbatim}
FileSystem
n?: Name
report!: Message

\texttt{cwd} \texttt{\langle n\rangle} \notin \texttt{dirs}

report! = \texttt{Directory does not exist}
\end{verbatim}
4 Z M Sti x Specifications

When the specification with a description of the base types model. There are two basic types of objects in
the system files and directories. The type Name corresponds to a single file or directory name (for example, the
MStix file "foo"):

[Name]

Sequences of Name are full file or directory names (for example, the MStix file "foo/bar"):

Full Name ::= seq Name

The representation shown here has the last elements at the tail of the sequence, and so, for example, the representation
of "foo/bar" is the sequence (foo, bar). We use the Z sequence manipulation functions front, which yields a
subsequence up to the last element and last, which yields the element at the end of the sequence. Full file names are
unique in the system as are full directory names. Further, we restrict the system so that a file and a directory may
not share the same name.

4.1 State Description

The state of the file system is as follows:

\begin{verbatim}
FileSystem
  files : P Full Name
  dirs : P Full Name
  cwd : Full Name

\forall f : files \cup dirs \implies f /= < \implies \text{ front } f \in dirs

dirs \cap files = \emptyset

\text{ cwd } \in dirs
\end{verbatim}

There are three components: files, dirs, and cwd. The first component, files, is the set of files that currently exist
in the system. The second component, dirs, is the set of directories that currently exist in the system. The first
invariant states that all intermediate directories must exist for a file or directory to exist. The second invariant states
that file and directory names are distinct. The last component, cwd, does not record any permanent feature of the
file system but is instead used to mark a (user's) current directory in the system. The third invariant states that
cwd must correspond to an existing directory.

We proceed with operations that change or destroy the state. We specify the desired operation, and then
proceed to define the operation formally by defining behavior for cases in which the preconditions of an operation is not
satisfied.

4.2 The Initial State

An illustrative initial state for the file system is one in which there is only the root directory (i.e., the empty sequence):

\begin{verbatim}
InitFileSystem
  FileSystem
    files = \{ \}
    dirs = \{ ( ) \}
    cwd = ( )
\end{verbatim}

It is clear that the new state produced by InitFileSystem satisfies the invariants of FileSystem.
- **DeleteDir DirName**
  If DirName is an empty directory of the current directory it is removed, otherwise an appropriate error message is printed

- **Down DirName**
  If DirName exists as a subdirectory of the current, the current directory is set to DirName

- **Up**
  The current directory is set to the parent of the current directory

- **Root**
  The current directory is set to the root.

- **Where**
  Prints the 'extended' path of the current directory. The extended path includes the name of every directory between the current directory and the root, inclusive.

- **CreateFile FileName**
  If FileName is not already a file or directory in the current directory, it is created in the current directory.

- **DeleteFile FileName**
  If FileName is a file in the current directory it is removed, otherwise an appropriate error message is printed.

- **CopyFile OldFileName NewFileName**
  Copies any of the file OldFileName to NewFileName (since MS-DOS uses file contents, this is equivalent to creating a new file called NewFileName). CopyFile

- **CopyFile Down OldFileName NewFileName**
  Copies OldFileName into the directory NewFileName

- **CopyFile Up OldFileName**
  The file is copied to the parent of the current directory

- **MoveFile OldFileName NewFileName**
  Moves the name of the file OldFileName to NewFileName

- **MoveFile Down OldFileName NewFileName**
  Moves OldFileName into the directory NewFileName

- **MoveFile Up OldFileName**
  The file is moved to the parent of the current directory

- **ListDirs**
  Prints the names of all subdirectories in the current directory

- **ListFiles**
  Prints the names of all files in the current directory

- **Logoff**
  Safeguard

**MS-DOS handles the following exceptional conditions in a user-friendly manner:**

- **Directory or file already exists on a command, me, or copy**
- **Directory contains files on a rename**
- **Directory does not exist, can change, or delete**
- **File does not exist, can copy, me, or delete**
- `ChangeDir [DirName]`
  If `DirName` is given and exists as a subdirectory of the current directory, the current directory is set to `DirName`.
  If `DirName` is not given, the current directory is set to the root. If the `DirName` is ";", the current directory is set to the parent of the current directory.

- `Where`
  Prints the "extended" name of the current directory. The extended name includes the name of every directory between the current directory and the root, inclusive.

- `CreateFile [File]`
  The file `File` is created in the current directory.

- `DeleteFile [File]`
  If `File` is in the current directory, it is removed, otherwise an appropriate error message is printed.

- `Move [OldFile] [NewDir]`
  Moves the file `OldFile` to the directory `NewDir`. If the `NewDir` is ";", the file is moved to the parent of the current directory.

- `Copy [OldFile] [NewDir]`
  Creates a copy of the file `OldFile` into the directory `NewDir`. If the `NewDir` is ";", the file is copied to the parent of the current directory.

- `List`
  Prints the names of all files and subdirectories in the current directory. Subdirectories should be distinguished from files by a trailing slash ("/").

- `Logoff`
  Signs execution.

`MIX` must handle the following exceptional conditions in a user-friendly manner:

- Directory or file already exists on a `create`, `move`, or `copy`
- Directory contains files on a `rename`
- Directory does not exist on a `rename`, or `delete`
- File does not exist on a `copy`, `move`, or `delete`.

3 Revised Informal MIX Specifications

`MIX` is specified informally below with a description of the syntax and semantics of each command. Each of the commands takes one or more arguments, which are directory or file names. Directory and file names are strings of characters, "+", ",", ",", and ".".

- `InitFileSystem`
  Sets a valid initial (empty) state for the file system.

- `CreateDir [DirName]`
  If `DirName` is not already a file or directory in the current directory, creates a new directory called `DirName` as a subdirectory of the current directory else an appropriate error message is printed.
1 Introduction

This paper presents several different specifications of a single file system based on the MSIX file system. This project was started from class assignent initially used in undergraduate computer science courses at Boston University and more recently in the software engineering courses at Georgia Institute of Technology. Sections 2 and 3 present two slightly different informal specifications for MSIX. The initial specification was used in classes through Fall 1982. We now also specify and test specifications of several inconsistencies and inaccuracies with the specifications, as well as features that were difficult to express with formal approaches. The most important problems were that the initial specification does not completely describe file names, allow a file and a directory of the same name in the same subdirectory, and some of the original commands (ChangeDir, CopyFile, and MoveFile) actually create several commands. Rectify these problems in the second informal specification and the formal specifications.

Section 4 gives a multi-based specification for MSIX in Z, and section 5 gives formal test specifications based on the Z speck and the Geppi verification method. Results from test cases derived from these specifications are in our common paper [1]. Finally section 6 gives an algebraic specification for MSIX.

MSIX is a hierarchical system consisting of directories and files. Each directory can contain an arbitrary number of files and subdirectories, where a subdirectory is also a directory. MSIX ignores the contents of files; it only concerned with the file's name. The only operation MSIX keeps track of is a current directory. Initially there is an empty directory called the root, which is the current directory.

There are a total of eighteen operations defined in (the revised specification for) MSIX:

- Get operation to create a valid (empty) initial state for the file system
- The operations to create and delete directories
- The operations to create and delete files
- The operations to copy files
- The operations to move files
- The operations to change the current directory
- Get operation to print the full path of the current directory
- The operations to list files and directories
- Get operation to log off

2 Original Informal MSIX Specifications

MSIX is specified informally below with a description of the syntax and semantics of each command. Initially there is one empty directory called the root, which is the current directory.

- CreateDir DirName
  If DirName is not already in the current directory creates a new directory called DirName as a subdirectory of the current directory else an appropriate error message is printed.

- DeleteDir DirName
  If DirName is an empty subdirectory of the current directory it is removed, otherwise an appropriate error message is printed.
Functional and Test Specifications for the MiStix File System

Paul Ammann
Jeff Ott
{pamann,ott}@gmu.edu

Department of Information and Software Systems
George Mason University, Fairfax, VA 22030
January 19, 1993

Abstract

MiStix is a simple file system based on the Unix file system. MiStix is used in classroom exercises in graduate software engineering classes at George Mason University. In this document, we supply several different specifications for MiStix. First we give an informal specification. Next, since the informal specification turns out to be difficult to formalize directly, we supply a refined informal specification that matches subsequent formal specifications. We give a mathematical specification for MiStix in Z, followed by functional test specifications based on the Category-Partition method. Finally we give an algebraic specification for MiStix.