Dependable Software Systems

Specification-based Testing

Material drawn from [Mancoridis, Vokolos]

Specifications

- Specification ≡ Requirements Specification

- Requirements Specification:
  - Precise and detailed description of the system's functionality and constraints.
  - Intended to communicate what is required to system developers and serve as the basis of a contract.
  - Written for technical audience.

Specification-based Testing

- Specification-based testing uses the specification of the program as the point of reference for test data selection and adequacy.

What specification ?

Specifications

- A specification can be any one (or more) of the following:
  - A written document
  - A collection of user scenarios (use-cases)
  - A set of models
  - A formal, mathematical description
  - A prototype

Specifications as Written Document

Command: lookup

Syntax: lookup <pattern> <file>

Function: The lookup command is used to locate one or more instances of a given pattern in a text file. All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern , the entire pattern must be enclosed in quotes (“). To include a quotation mark in the pattern , two quotes in a row (“”) must be used.

Specifications as Written Document

Examples:

```
lookup john myfile
```

displays lines in the file myfile which contain john

```
lookup “john smith” myfile
```

displays lines in the file myfile which contain john smith

```
lookup “john”” smith” myfile
```

displays lines in the file myfile which contain john” smith

Specification as Written Document

- Typically written in natural language.
- Readable by all, but...
  - **Lack of clarity**. Precision is difficult without making the document difficult to read.
  - **Requirements confusion**. Functional and non-functional requirements tend to be mixed-up.
  - **Requirements amalgamation**. Several different requirements may be expressed together.

Modeling

- A model is an abstraction of the system being studied.
- Different models present the system from different perspectives
- Various “semi-formal” and “formal” notations are available for modeling
  - Diagrams, tables, structured language, etc.
    - Mostly visual, capture structure and some semantics
    - Facilitate communication with different audiences
  - Finite State Machines (FSMs), grammars, set theory, ...
    - Precise semantics, reasoning possible
    - Detailed models

Modeling Techniques

- Modeling Information & Behavior
  - E-R (Information modeling)
  - SADT (Structured Analysis)
  - UML (Object Oriented Analysis)
  - Z, Larch, VDM, Pre/Post Conditions (Formal Methods)
- Modeling System Qualities
  - Timed Petri Nets (Performance)
  - Task Models (Usability)
  - Probabilistic MTTF (Reliability)

Specification-based Testing

- The multitude of specification flavors, coupled with different degrees of conduciveness for analysis have made life difficult for people looking to develop encompassing technology for specification-based testing.

- There has been attention from the research community, however small, fragmented steps have been accomplished in producing techniques for the practitioner.

- In general, when testing industrial software systems, testers may need to rely on more than one approach to test to the specification.

Formal Specification

Specification in the Software Process

- Specification and design are intermingled.
- Architectural design is essential to structure a specification.
- Formal specifications are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

Formal Specification on Trial

- Formal techniques are not widely used in industrial software development.
- Given the relevance of mathematics in other engineering disciplines, why is this the case?

Why Aren't Formal Methods Used?

- Inherent management conservatism.
- Many software engineers lack the training in discrete math necessary for formal specification.
- System customers may be unwilling to fund specification activities.
- Some classes of software (particularly interactive systems and concurrent systems) are difficult to specify using current techniques.

Why Aren't Formal Methods Used?

- There is widespread ignorance of the applicability of formal specifications.
- There is little tool support available for formal notations.
- Some computer scientists who are familiar with formal methods lack knowledge of the real-world problems to which these may be applied and therefore oversell the technique.

Advantages of Formal Specification

- It provides insights into the software requirements and the design.
- Formal specifications may be analyzed mathematically for consistency.
- It may be possible to prove that the implementation satisfies the specification.

Advantages of Formal Specification

- Formal specifications may be used to guide the tester of the component in identifying appropriate test cases.
- Formal specifications may be processed using software tools. It may be possible to animate the specification to provide a software prototype.

Seven Myths of Formal Methods

- Perfect software results from formal methods
  - Nonsense - the formal specification is a model of the real-world and may incorporate misunderstandings, errors and omissions.
- Formal methods means program proving
  - Formally specifying a system is valuable without formal program verification as it forces a detailed analysis early in the development process.
- Formal methods can only be justified for safety-critical systems
  - Industrial experience suggests that the development costs for all classes of system are reduced by using formal specification.

Seven Myths of Formal Methods

- Formal methods are for mathematicians
  - Nonsense - only simple math is needed.
- Formal methods increase development costs
  - Not proven. However, formal methods definitely push development costs towards the front-end of the life cycle.
- Clients cannot understand formal specifications
  - They can if they are paraphrased in natural language.
- Formal methods have only been used for trivial systems
  - There are now many published examples of experience with formal methods for non-trivial software systems.

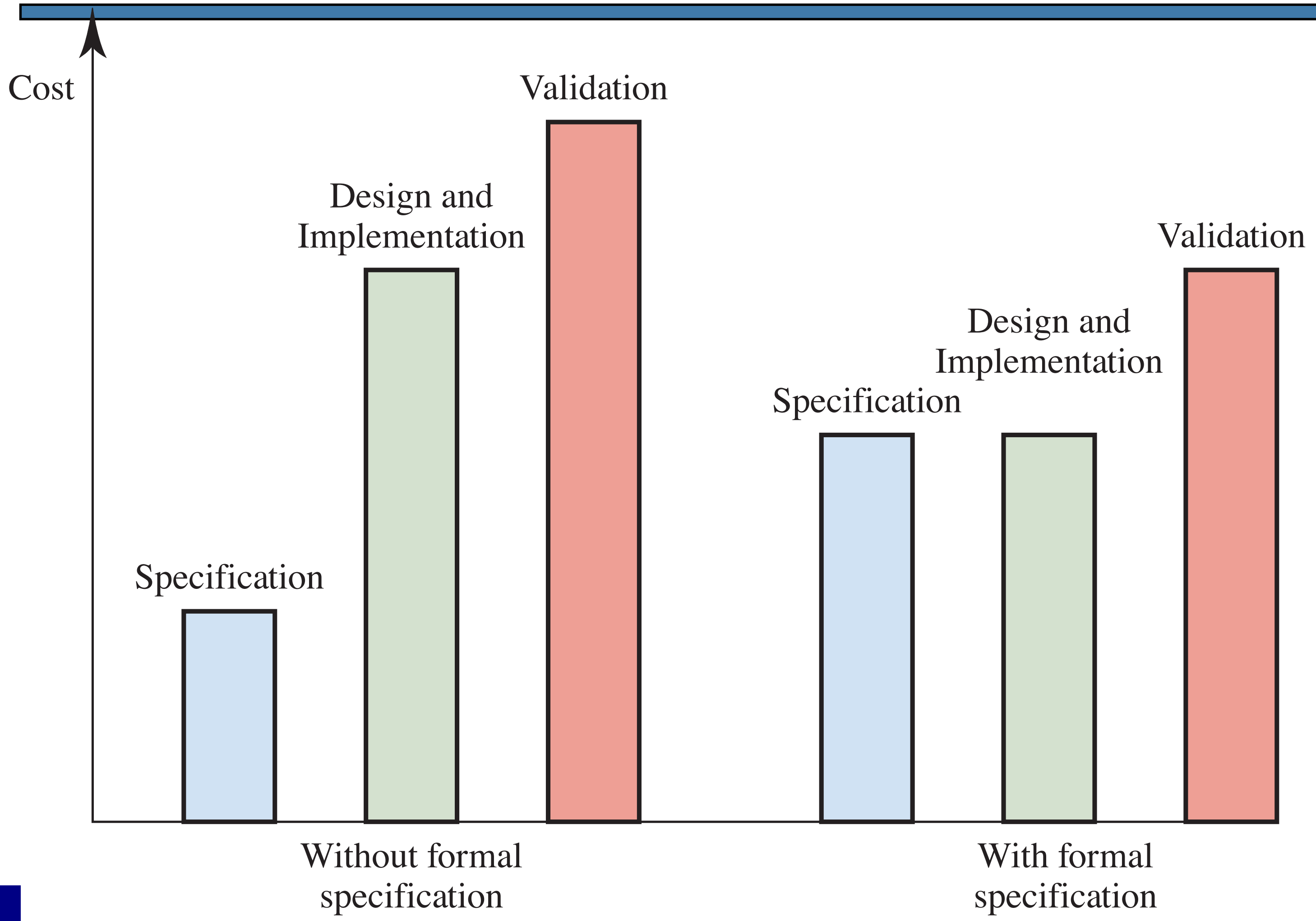
The Verdict!

- The reasons put forward for not using formal specifications and methods are weak.
- However, there are good reasons why these methods are not used:
  - The move to interactive systems. Formal specification techniques cannot cope effectively with graphical user interface specification.
  - Successful software engineering. Investing in other software engineering techniques may be more cost-effective.

Use of Formal Methods

- These methods are unlikely to be widely used in the foreseeable future. Nor are they likely to be cost-effective for most classes of system.
- They will become the normal approach to the development of safety critical systems and standards.
- This changes the expenditure profile through the software process.

Development Costs with Formal Specification



Specifying Functional Abstractions

- The simplest specification is function specification. There is no need to be concerned with global state.
- The formal specification is expressed as input and output predicates (pre and post conditions).
- Predicates are logical expressions which are always either true or false .
- Predicate operators include the usual logical operators and quantifiers such as for-all and exists.

Specification with Pre- and Post-Conditions

- Set out the pre-conditions
  - A statement about the function parameters stating what is invariably true before the function is executed
- Set out the post-conditions
  - A statement about the function parameters stating what is invariably true after the function has executed
- The difference between the pre and post conditions is due to the application of the function to its parameters. Together the pre and post conditions are a function specification.

Specification Development

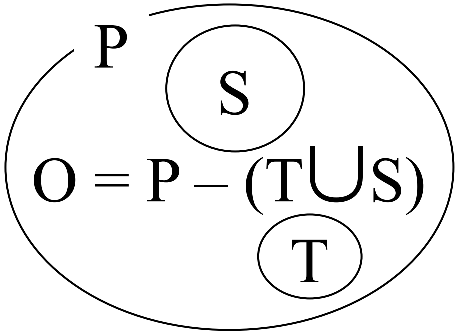
- Establish the bounds of the input parameters. Specify this as a predicate.
- Specify a predicate defining the condition which must hold on the result of the function if it computes correctly.
- Establish what changes are made to the input parameters by the function and specify these as a predicate.
- Combine the predicates into pre and post conditions.

The Specification of a Search

function Search (X: **array 1 .. N of** INTEGER; Key: INTEGER)
return INTEGER ;

Pre: $\exists\ 1 \leq i \leq N \bullet X[i] = Key$
Post: $X'\ [Search\ (X, Key)] = Key \wedge X = X' \wedge Key' = Key$

Set Theory Review



$O = P - (T \cup S)$

teacher(Mary)

student(John)

$x \in T \Leftrightarrow \text{teaches}(x)$

$x \in S \Leftrightarrow \text{student}(x)$

$x \in O \Leftrightarrow \neg\text{teaches}(x) \wedge \neg\text{student}(x)$

$\text{teacher}(x) \Rightarrow \neg\text{student}(x)$

$\text{student}(x) \Rightarrow \neg\text{teacher}(x)$

$\text{teacher}(x) \Leftrightarrow \neg\text{student}(x) \wedge \neg\text{other}(x)$

$\text{Mary} \in T \Leftrightarrow \text{teaches}(\text{Mary})$

$\text{teaches}(\text{Mary}) \Rightarrow \neg\text{student}(\text{Mary})$

$\quad \Leftrightarrow \neg\text{student}(\text{Mary}) \wedge \neg\text{other}(\text{Mary})$

$\exists x \in P \bullet \text{teaches}(x)$

$\forall x \in S \bullet \text{student}(x)$

$\neg\exists x \in S \bullet \text{teaches}(x) \sim> \forall x \in S \bullet \neg \text{teaches}(x)$

$$\begin{aligned} &\forall x \in S \bullet \exists y \in T \bullet \text{student}(x) \\ &\qquad \wedge\ \text{teaches}(y) \\ &\qquad \wedge\ \text{supervises}(y,x) \end{aligned}$$

Implication

$a \Rightarrow b \equiv \neg a \lor b$

a \ b	0	1
0	1	1
1	0	1

a = possibility of precipitation (p.o.p) > 50%

b = take an umbrella

So ⇒ informally means if the p.o.p is greater than 50% I will take an umbrella, but if the p.o.p is less than or equal to 50%, I may or may not take an umbrella

$a \Rightarrow b$

If (p.o.p > 50%) then (take an umbrella)

$a = 0 \land b = 0$

If (p.o.p ≤50%) then ¬ (take an umbrella)
= 1

$a = 0 \land b = 1$

If (p.o.p ≤ 50%) then (take an umbrella)
= 1

$a = 1 \land b = 0$

If (p.o.p > 50%) then ¬(take an umbrella)
= 0

$a = 1 \land b = 1$

If (p.o.p > 50%) then (take an umbrella)
= 1

Pre/Post Condition Examples

proc Reverse(a:array 0..9 of int, size:int)

pre: ?

post: ?

proc D_Sort(A:array 0..9 of int, size:int)

pre: ?

post: ?

Pre/Post Condition Examples

```
proc Reverse(a:array 0..9 of int, size:int)
    pre: $1 \leq size \leq 10$
    post: $size = size'$
        $\wedge\ \forall 0 \leq i < size \bullet A[i] = A'[(size-1)-i]$

proc D_Sort(A:array 0..9 of int, size:int)
    pre: $(1 \leq size \leq 10)$
        $\wedge\ (\forall 0 \leq i < size \bullet \neg \exists 0 \leq j < size \bullet (i \neq j) \wedge (A[\,i\,] = A[\,j\,]))$

    post: $\forall 0 \leq i < (size - 1) \bullet A'[\,i\,] \leq A'[\,i+1\,]$
        $\wedge\ size' = size$
        $\wedge\ \forall 0 \leq i < size \bullet \exists 0 \leq j < size \bullet A[\,i\,] = A'[\,j\,]$
```

A' is a permutation of A

Pre/Post Condition Examples(2)

proc Avg(A:array 0..9 of int, size:int, res:real)

pre: ?

post: ?

proc Init(A:array 0..9 of int, size:int)

pre: ?

post: ?

Pre/Post Condition Examples(2)

proc Avg(A:array 0..9 of int, size:int, res:real)

pre: $0 < size \le 10$

post: (A’ = A)

$\wedge$ (size’ = size)

$$\wedge\ res' = \frac{\sum_{i=0}^{size-1} A[i]}{size}$$

proc Init(A:array 0..9 of int, size:int)

pre: $0 \le size \le 10$

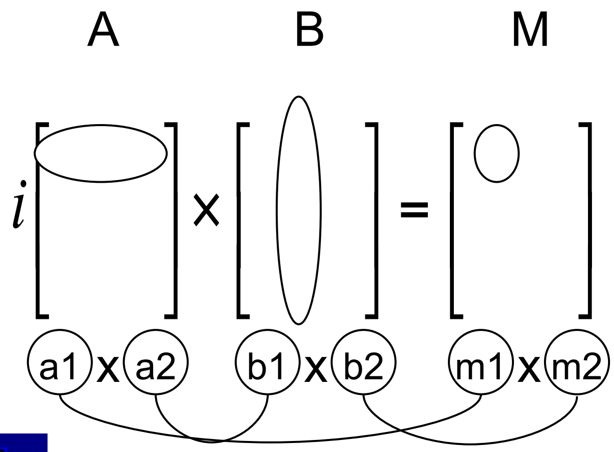
post: $(\forall\ 0 \le i < size \bullet A'[\,i\,] = 0) \wedge (size' = size)$

Pre/Post Condition Examples(3)

```
proc m_mult(A:array 0..9,0..9 of int; a1, a2: int;
            B:array 0..9,0..9 of int; b1, b2: int;
            M:array 0..9,0..9 of int; m1, m2:int)
```

pre: ?

post:?



$$\begin{bmatrix}1 & 3\\ 2 & 4\end{bmatrix}\times\begin{bmatrix}1 & 3\\ 2 & 4\end{bmatrix}=\begin{bmatrix}1\cdot1+3\cdot2 & 1\cdot3+3\cdot4\\ 2\cdot1+4\cdot2 & 2\cdot3+4\cdot4\end{bmatrix}$$

$$=\begin{bmatrix}7 & 15\\ 10 & 22\end{bmatrix}$$

$$=\sum_{k=0}^{1}A_{1k}\cdot B_{k1}$$

$$M_{11}=A_{1,0}\cdot B_{0,1}+A_{11}\cdot B_{11}$$

Pre/Post Condition Examples(3)

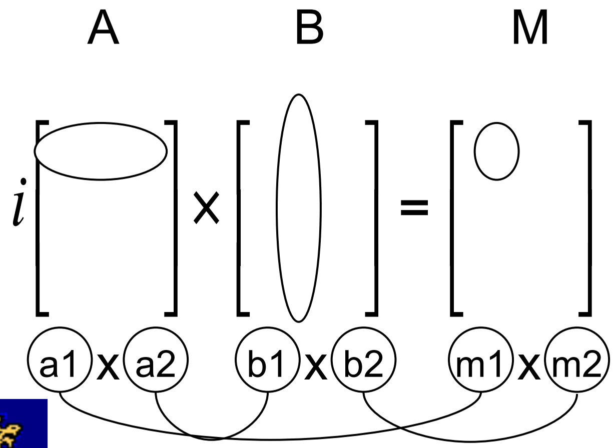
proc m_mult(A:array 0..9,0..9 of int; a1, a2: int;
B:array 0..9,0..9 of int; b1, b2: int;
M:array 0..9,0..9 of int; m1, m2: int)

pre: $(0 \leq a1,a2,b1,b2,m1,m2 \leq 9) \wedge (a2 = b1)$

post: $(A' = A) \wedge (B' = B) \wedge (m1' = a1) \wedge (m2' = b2)$

$\wedge \ \forall 0 \leq i < m1', 0 \leq j < m2' \bullet m[i,j]' = \sum_{k=0}^{a2-1} A[i,k] \cdot B[k,j]$

$\wedge \ (a1' = a1) \wedge (a2' = a2) \wedge (b1' = b1) \wedge (b2' = b2)$



$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 3 \cdot 2 & 1 \cdot 3 + 3 \cdot 4 \\ 2 \cdot 1 + 4 \cdot 2 & 2 \cdot 3 + 4 \cdot 4 \end{bmatrix}$$

$$= \begin{bmatrix} 7 & 15 \\ 10 & 22 \end{bmatrix}$$

$$= \sum_{k=0}^{1} A_{1k} \cdot B_{k1}$$

$$M_{11} = A_{1,0} \cdot B_{0,1} + A_{11} \cdot B_{11}$$

Pre/Post Condition Examples(4)

Number of **e**’s in A

```
proc count(A:array 0..9 of int; size, e, count:int)
    pre:  ?
    post: ?
```

Pre/Post Condition Examples(4)

Number of **e**’s in A

proc count(A:array 0..9 of int; size, e, count:int)

pre: $0 \le size \le 10$

post: $size' = size$

$\wedge\ e' = e$

$\wedge\ A' = A$

$\wedge\ count' = |\{\forall\ 0 \le i < size \cdot A[\ i\ ] = e\}|$

Pre/Post Conditions with Java

```
import java.util.*;

class Stack
    private Vector v;
    void push(Object obj) {
      pre: ?
      post: ?
     }
    Object pop() {
      pre: ?
      post: ?
  }
```

```
  Object top() {
    pre:   ?
    post:  ?
  }

  stack() {
    pre:   ?
    post: ?
  }
}
```

Pre/Post Conditions with Java

```
import java.util.*;

class Stack
   private Vector v;

   void push(Object obj) {
       pre:        (obj ≠ null) ∧ (v ≠ null)
       post:       (v'.elementAt(v.size()) = obj)
                   ∧ (∀ 0 ≤ i < v.size() •
                        v'.elementAt(i) = v.elementAt(i))
                   ∧  v'.size() = v.size()+1
   }

   Object pop() {
       pre:        v ≠ null
       post:       (v.size() > 0) ⇒
                    (pop' = v.elementAt(v.size()-1))
                ∧ (v.size() < 0) ⇒ (pop' = null)
                ∧ (∀0 ≤ i < v.size()-1 •
                       v'.elementAt(i)= v.elementAt(i))
                ∧ (v'.size() = v.size()-1)
}

  Object top() {
      pre:    v ≠ null
      post:       (v.size() ≤ 0) ⇒ (top' = null)
                ∧ (v.size() > 0) ⇒
                     (top' = v.elementAt(v.size()-1)
                ∧ ∀0 ≤  i < v.size() •
                     v'.elementAt(i) =
                            v.elementAt(i)
                ∧ v'.size() = v.size()
  }

  stack() {
      pre:    true
      post:   v ≠ null   ∧ v.size() = 0
  }
}
```

Stack Implementation in Java

```
import java.util.*;

class stack {
    private Vector v;

    void push(Object obj) {
        v.addElement(Obj);
        }

    Object pop() {
        if (v.size() > 0) {
                Object res = v.elementAt(v.size
                    ()-1);
                v.removeElementAt(v.size()-1);
                return res;
        } else return null;
    }
```

```
    Object top() {
        if (v.size > 0)
            return v.elementAt(v.size()-1);
        else return null);
        }

        Stack() {
            v = new Vector();
            v.removeAllElements();
            }
}
```

Pre/Post Conditions for C functions

strlen()

In order to be able to work with C strings, We can introduce function:

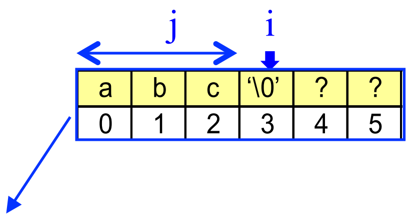
- **isNullTerminated(s)**: returns True if the string s is null terminated and False otherwise.

// The strlen() function returns the number of bytes in s, not including the terminating null character.

int strlen (const char *s)

pre: **isNullTerminated(s)**

post: $(s' = s) \wedge$



a	b	c	'\0'	?	?
0	1	2	3	4	5

$$(\exists\ 0 \leq i \cdot (s[\ i\ ] = \text{'\textbackslash 0'} \wedge (\forall\ 0 \leq j < i \cdot (s[\ j\ ] \neq \text{'\textbackslash 0'}\ )) \Rightarrow \forall\ 0 \leq k < i \cdot (s[\ k\ ] = s'[\ k\ ]))$$

$\wedge$

$$(\exists\ 0 \leq i \cdot (s[\ i\ ] = \text{'\textbackslash 0'} \wedge (\forall\ 0 \leq j < i \cdot (s[\ j\ ] \neq \text{'\textbackslash 0'})) \Rightarrow strlen(s) = i\ ))$$

Pre/Post Conditions for C functions

strcmp()

The strcmp() function compares two strings byte-by-byte, according to the ordering of your machine's character set. The function returns an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2 respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared. Bytes following a null byte are not compared.

int strcmp(const char *s1, const char *s2)

pre: ?

post: ?

Pre/Post Conditions for C functions

strcmp()

int strcmp(const char *s1, const char *s2)

pre: $isNullTerminated(s1) \land isNullTerminated(s2)$

post: $(s1' = s1) \land (\exists\ 0 \le i \cdot (s1[\ i\ ] = `\backslash 0' \land (\forall\ 0 \le j < i \cdot (s1[\ j\ ] \ne `\backslash 0'))$

$\Rightarrow \forall\ 0 \le k < i \cdot (s1[\ k\ ] = s1'[\ k\ ])) \land$

$(s2' = s2) \land (\exists\ 0 \le i \cdot (s2[\ i\ ] = `\backslash 0' \land (\forall\ 0 \le j < i \cdot (s2[\ j\ ] \ne `\backslash 0'))$

$\Rightarrow \forall\ 0 \le k < i \cdot (s2[\ k\ ] = s2'[\ k\ ])) \land$

$(strlen(s1) = strlen(s2)) \Rightarrow$

$(\forall\ 0 \le i \le strlen(s1) \bullet s1[\ i\ ] = s2[\ i\ ] \Rightarrow strcmp(s1,s2)=0) \land$

$(\exists\ 0 \le i \le min(strlen(s1),strlen(s2)) \bullet$

$((s1[\ i\ ] > s2[\ i\ ] \land \forall\ 0 \le j < i \bullet s1[\ j\ ] = s2[\ j\ ]) \Rightarrow$

$strcmp(s1,s2) > 0) \land$

$((s1[\ i\ ] < s2[\ i\ ] \land \forall\ 0 \le j < i \bullet s1[\ j\ ] = s2[\ j\ ]) \Rightarrow$

$strcmp(s1,s2) < 0))$

$)$

Pre/Post Conditions for C functions

strchr() solution

The strchr() function returns a pointer to the first occurrence of c (converted to a char) in string s, or a null pointer if c does not occur in the string.

char *strchr(const char *s, int c)

pre: ?

post: ?

Pre/Post Conditions for C functions

strchr() solution

char *strchr (const char *s, int c)

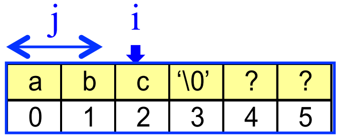
pre: isNullTerminated(s) $\wedge$ $0 \leq c \leq 255$

post:

$(s' = s) \wedge$

$(c' = c) \wedge$

$(\mathrm{strcmp}(s',s)=0) \wedge$



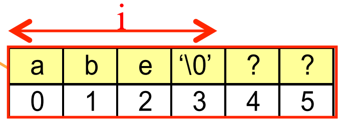
a	b	c	'\0'	?	?
0	1	2	3	4	5

$(\exists\ 0 \leq i < \mathrm{strlen}(s) \bullet s[\,i\,] = c \wedge (\forall\ 0 \leq\ j < i \bullet s[\,j\,] \neq c) \Rightarrow$

strchr

$(s,c) = s+i\ ) \wedge$

$(\forall\ \ 0 \leq i < \mathrm{strlen}(s) \bullet s[\,i\,] \neq c \Rightarrow \mathrm{strchr}(s,c) = \mathrm{null}\ )$



a	b	e	'\0'	?	?
0	1	2	3	4	5