Reading Contracts
Recall: Mathematical Models

- Each **variable** in the program has a **type**
  - Examples: `int`, `double`, ...
- Each program type has a **mathematical type** that **models** it: you should think of any variable of that program type as having a value from its mathematical model’s mathematical space/domain
  - Examples (respectively): `integer`, `real`, ...
# Mathematical Models

<table>
<thead>
<tr>
<th>Program type</th>
<th>Mathematical type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>char</td>
<td>character</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>(-2147483648 through 2147483647)</td>
</tr>
<tr>
<td>double</td>
<td>real</td>
</tr>
<tr>
<td></td>
<td>(about $\pm 10^{\pm 308}$, 15 significant digits)</td>
</tr>
<tr>
<td>String</td>
<td>string of character</td>
</tr>
</tbody>
</table>
# More Mathematical Models

<table>
<thead>
<tr>
<th>Program type</th>
<th>Mathematical type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaturalNumber</td>
<td>integer (non-negative)</td>
</tr>
<tr>
<td>Queue&lt;T&gt;</td>
<td>string of $T$</td>
</tr>
<tr>
<td>Stack&lt;T&gt;</td>
<td>string of $T$</td>
</tr>
<tr>
<td>Sequence&lt;T&gt;</td>
<td>string of $T$</td>
</tr>
<tr>
<td>Set&lt;T&gt;</td>
<td>finite set of $T$</td>
</tr>
<tr>
<td>Map&lt;K,V&gt;</td>
<td>finite set of $(K,V)$ (with function property)</td>
</tr>
</tbody>
</table>
Recall: Method Contracts

• Each method has:
  – A *precondition* (*requires clause*) that characterizes the responsibility of the program that *calls* (*uses*) that method (client code)
  – A *postcondition* (*ensures clause*) that characterizes the responsibility of the program that *implements* that method (implementation code in the method body)
Recall: Meaning of a Contract

• If its precondition is true when a method is called, then the method will terminate — return to the calling program — and the postcondition will be true when it does return.

• If its precondition is not true when a method is called, then the method may do anything (including not terminate).
Example

/**
 * ...
 * @replaces s2
 * @requires
 *   |s1| >= 1
 * @ensures
 *   |s2| = |s1| - 1  and
 *   for all i, j: integer, a, b: string of integer
 *   where (s1 = a * <i> * <j> * b)
 *   (there exists c, d: string of integer
 *   (|c| = |a|  and  s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                           Sequence<Integer> s2) {...}
Example

```java
/**
 * ... 
 * @replaces s2
 * @requires
 * |s1| >= 1
 * @ensures
 * |s2| = |s1| - 1  and 
 * for all i, j: integer, a, b: string of integer
 * where (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a|  and  s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
        Sequence<Integer> s2) {...}
```
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 *     |s1| >= 1
 * @ensures
 *     |s2| = |s1| - 1 and
 *     for all i, j: integer, a, b: string of integer
 *     where (s1 = a * <i> * <j> * b)
 *     (there exists c, d: string of integer
 *      (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                          Sequence<Integer> s2) {...}
```

Stated in English, what does this method do?
How Do You Approach This?

• Recommendations:
  – Know the meanings of all the individual symbols and terms appearing in the contract
  – Consider specific values for the parameters
    • Start with “smallest” legal input values as examples
    • Continue with more examples
    • “See the pattern”
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 *     |s1| >= 1
 * @ensures
 *     |s2| = |s1| - 1 and
 *     for all i, j: integer, a, b: string of integer
 *     where (s1 = a * <i> * <j> * b)
 *     (there exists c, d: string of integer
 *      (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */
public static void smooth(Sequence<Integer> s1,
                     Sequence<Integer> s2) {...}
```

What do these Javadoc tags mean?
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 *     |s1| >= 1
 * @ensures
 *     |s2| = |s1| - 1 and
 *     for all i, j: integer, a, b: string of integer
 *     where (s1 = a * <i> * <j> * b)
 *     (there exists c, d: string of integer
 *      (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
             Sequence<Integer> s2) {...}
```

What does this mean?
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 *   |s1|  >= 1
 * @ensures
 *   |s2|  = |s1|  - 1  and
 *   for all i, j: integer, a, b: string of integer
 *     where (s1 = a * <i> * <j> * b)
 *   (there exists c, d: string of integer
 *     (|c|  = |a|  and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                         Sequence<Integer> s2) {...}
```

What does this mean?
public static void smooth(Sequence<Integer> s1, Sequence<Integer> s2) {...}
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 * |s1| >= 1
 * @ensures
 * |s2| = |s1| - 1  and
 * for all i, j: integer, a, b: string of integer
 * where (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a|  and  s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                             Sequence<Integer> s2) {...}
```

What does this mean?
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 * |s1| >= 1
 * @ensures
 * |s2| = |s1| - 1  and
 * for all i, j: integer, a, b: string of integer
 * where (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 *  (|c| = |a|  and  s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1, Sequence<Integer> s2) {...}
```
Example

```java
/**
  * ...  
  * @replaces s2
  * @requires
  *  |s1| >= 1
  * @ensures
  *  |s2| = |s1| - 1  and
  *    for all i, j: integer, a, b: string of integer
  *    where (s1 = a * <i> * <j> * b)
  *    (there exists c, d: string of integer
  *    (|c| = |a|  and  s2 = c * <(i+j)/2> * d))
  */

public static void smooth(Sequence<Integer> s1,
                      Sequence<Integer> s2) {...}
```

What does this mean?
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 * |s1| >= 1
 * @ensures
 * |s2| = |s1| - 1 and
 * for all i, j: integer, a, b: string of integer
 * where (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 *  (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                         Sequence<Integer> s2) {...}
```

What does all this mean?
/**
 * ...
 * @replaces s2
 * @requires
 * |s1| >= 1
 * @ensures
 * |s2| = |s1| - 1 and
 * for all i, j: integer, a, b: string of integer
 * when (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                         Sequence<Integer> s2) {...}
Example

```java
/**
 * ... is true for every possible combination of values of i, j, a, and b...
 * @replaces s2
 * @requires
 * |s1| >= 1
 * @ensures
 * |s2| = |s1| - 1 and
 * for all i, j: integer, a, b: string of integer
 * where (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1, Sequence<Integer> s2) {...}
```
Example

```java
/**
 * @replaces s2
 * @requires |s1| >= 1
 * @ensures |s2| = |s1| - 1 and
 * for all i, j: integer, a, b: string of integer
 * where (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
             Sequence<Integer> s2) {...}
```

... at least, for every combination of values of i, j, a, and b satisfying the where clause.
/**
 * ...
 * @replaces s2
 * @requires
 *     |s1| >= 1
 * @ensures
 *     |s2| = |s1| - 1  and
 *     for all i, j: integer, a, b: string of integer
 *     where (s1 = a * <i> * <j> * b)
 *     (there exists c, d: string of integer
 *      (|c| = |a|  and  s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                          Sequence<Integer> s2) {...}
Universal Quantification

• *Universal quantification* is used when you want to say something is true for every combination of values that satisfy a certain property:

\[
\text{for all quantified-variables where (restriction-assertion)} \\
\text{(main-assertion)}
\]
Universal Quantification

- **Universal quantification** is used when you want to say something is true for every combination of values that satisfy a certain property:

  \[
  \text{for all quantified-variables where (restriction-assertion) (main-assertion)}
  \]
Example Use of \textit{for all}

- “Every non-empty string has positive length.”
Example Use of \textit{for all}

• “Every non-empty string has positive length.”

\begin{verbatim}
for all \texttt{s}: \texttt{string of } T
    where (s /= < >)
          (|s| > 0)
\end{verbatim}
Existential Quantification

• *Existential quantification* is used when you want to say something is true for some combination of values:

  \[
  \textbf{there exists } \text{quantified-variables} \quad \text{(assertion)}
  \]
Existential Quantification

• *Existential quantification* is used when you want to say *something* is true for some combination of values:

\[
\text{there exists } \text{quantified-variables} \\
\text{(assertion)}
\]
Example Use of \textit{there exists}

- “Some positive integer is odd.”
Example Use of \(\textbf{there exists}\)

- “Some positive integer is odd.”

\[
\text{there exists } n, k: \text{integer} \\
(n > 0 \text{ and } n = 2 \times k + 1)
\]
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 *   |s1| >= 1
 * @ensures
 *   |s2| = |s1| - 1 and
 *     for all i, j: integer, a, b: string of integer
 *     where (s1 = a * <i> * <j> * b)
 *     (there exists c, d: string of integer
 *      (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                          Sequence<Integer> s2) {...}
```

Back to the main example... What are some “smallest” possible input values for the method’s parameters $s_1$ and $s_2$?
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 * |s1| >= 1
 * @ensures
 * |s2| = |s1| - 1 and
 * for all i, j: integer, a, b: string of integer
 * where (s1 = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                          Sequence<Integer> s2) {...}
```

Note that the incoming value of `s2` does not matter because it is a replaces-mode parameter.
### A Smallest Value (for \texttt{s1})

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| smooth(s1, s2);                           | \texttt{s1 = \langle7\rangle}  
|                                            | \texttt{s2 = \langle1, 2, 3\rangle} |

\texttt{s1 =}  
\texttt{s2 =}
A Smallest Value (for $s_1$)

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| smooth($s_1$, $s_2$) | $s_1 = \langle 7 \rangle$
                 | $s_2 = \langle 1, 2, 3 \rangle$ |
|               | $s_1 =$              |
|               | $s_2 =$              |

How do you predict the values of $s_1$ and $s_2$?
/**
 * ...
 * @replaces s2
 * @requires
 * |<7>| >= 1
 * @ensures
 * |s2| = |<7>| - 1 and
 * for all i, j: integer, a, b: string of integer
 * where (<7> = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1, 
    Sequence<Integer> s2) {...}
Example

```java
/**
 * ...
 * @replaces s2
 * @requires
 * |<7>| >= 1
 * @ensures
 * |s2| = |<7>| - 1 and
 * for all i, j: integer, a, b: string of integer
 * where (<7> = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 * (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1, Sequence<Integer> s2) {...}
```

To determine whether the value of \textit{s1} is a “legal” input, substitute \textit{s1} = \textit{<7>} in the requires clause.
To predict the value of $s_2$, substitute $s_1 = <7>$ in the ensures clause.

```
/**
 * ...
 * @replaces s2
 * @requires
 *  |<7>| >= 1
 * @ensures
 *  |s2| = |<7>| - 1   and
 *  for all i, j: integer, a, b: string of integer
 *('; where (<7> = a * <i> * <j> * b)
 * (there exists c, d: string of integer
 *  (|c| = |a|   and   s2 = c * <$>((i+j)/2$> * d))
 */

public static void smooth(Sequence<Integer> s1,
                          Sequence<Integer> s2) {...}
```
We see that, in general, $|s_2| = |s_1| - 1$ so, in this case, $|s_2| = 0$.

```java
/**
 * ...
 * @replaces s2
 * @requires
 *     |s1| >= 1
 * @ensures
 *     |s2| = |<7>| - 1 and
 *     for all i, j: integer, a, b: string of integer
 *     where (<7> = a * <i> * <j> * b)
 *     (there exists c, d: string of integer
 *     (|c| = |a| and s2 = c * <(i+j)/2> * d))
 */

public static void smooth(Sequence<Integer> s1,
                          Sequence<Integer> s2) {...}
```
Vacuously True

• To make the ensures clause true, the first line implies we must have \( s_2 = \langle \rangle \)

• But there are no values of \( i, j, a, \) and \( b \) that make the where-clause true in:

\[
\text{for all } i, j: \text{integer}, \ a, \ b: \text{string of integer} \\
\text{where } (\langle7\rangle = a \times \langle i\rangle \times \langle j\rangle \times b)
\]

(…)

• This universally quantified expression is defined to be \textit{vacuously true}
A Next Smallest Value (for \texttt{s1})

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| \texttt{smooth(s1, s2);} | \texttt{s1 = \langle 7, 23 \rangle}  \\
|                       | \texttt{s2 = \langle 1, 2, 3 \rangle}  |
|                       | \texttt{s1 = \langle 7, 23 \rangle}  \\
|                       | \texttt{s2 = = } |
A Next Smallest Value (for $s_1$)

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = &lt;7, 23&gt;$</td>
<td>$s_1 = &lt;7, 23&gt;$</td>
</tr>
<tr>
<td>smooth($s_1$, $s_2$)</td>
<td>$s_2 = &lt;1, 2, 3&gt;$</td>
</tr>
</tbody>
</table>

How do you predict the value of $s_2$?
# Values of Quantified Variables

<table>
<thead>
<tr>
<th>s1</th>
<th>&lt;7, 23&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>&lt; &gt;</td>
</tr>
<tr>
<td>i</td>
<td>7</td>
</tr>
<tr>
<td>j</td>
<td>23</td>
</tr>
<tr>
<td>b</td>
<td>&lt; &gt;</td>
</tr>
<tr>
<td>c</td>
<td>&lt; &gt;</td>
</tr>
<tr>
<td>d</td>
<td>&lt; &gt;</td>
</tr>
<tr>
<td>s2</td>
<td>&lt;15&gt;</td>
</tr>
</tbody>
</table>
Values of Quantified Variables

<table>
<thead>
<tr>
<th>$s1$</th>
<th>$&lt;7, 23&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$i$</td>
<td>7</td>
</tr>
<tr>
<td>$j$</td>
<td>23</td>
</tr>
<tr>
<td>$b$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$c$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$d$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$s2$</td>
<td>$&lt;15&gt;$</td>
</tr>
</tbody>
</table>

We see that, when $s1 = <7, 23>$ we must have $s2 = <15>$.
## A Next Smallest Value (for $s_1$)

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = &lt;7, 23, 2&gt;$</td>
<td>$s_2 = &lt;1, 2, 3&gt;$</td>
</tr>
<tr>
<td>smooth($s_1, s_2$);</td>
<td></td>
</tr>
<tr>
<td>$s_1 = &lt;7, 23, 2&gt;$</td>
<td>$s_2 = =$</td>
</tr>
</tbody>
</table>
A Next Smallest Value (for $s_1$)

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = &lt;7, 23, 2&gt;$</td>
<td>$s_1 = &lt;7, 23, 2&gt;$</td>
</tr>
<tr>
<td>$s_2 = &lt;1, 2, 3&gt;$</td>
<td>$s_2 = &lt;1, 2, 3&gt;$</td>
</tr>
</tbody>
</table>

How do you predict the value of $s_2$?

smooth($s_1$, $s_2$)
## Values of Quantified Variables

<table>
<thead>
<tr>
<th>$s1$</th>
<th>$&lt;7, 23, 2&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$i$</td>
<td>7</td>
</tr>
<tr>
<td>$j$</td>
<td>23</td>
</tr>
<tr>
<td>$b$</td>
<td>$&lt;2&gt;$</td>
</tr>
<tr>
<td>$c$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$d$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$s2$</td>
<td>$&lt;15, ?&gt;$</td>
</tr>
</tbody>
</table>
## Values of Quantified Variables

<table>
<thead>
<tr>
<th>$s1$</th>
<th>$&lt;7, 23, 2&gt;$</th>
<th>$&lt;7&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$&lt; &gt;$</td>
<td>$&lt;7&gt;$</td>
</tr>
<tr>
<td>$i$</td>
<td>$7$</td>
<td>$23$</td>
</tr>
<tr>
<td>$j$</td>
<td>$23$</td>
<td>$2$</td>
</tr>
<tr>
<td>$b$</td>
<td>$&lt;2&gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>$c$</td>
<td>$&lt; &gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>$d$</td>
<td>$&lt; &gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>$s2$</td>
<td>$&lt;15, ?&gt;$</td>
<td>$&lt; ?, 12&gt;$</td>
</tr>
</tbody>
</table>
Values of Quantified Variables

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$&lt;7, 23, 2&gt;$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$&lt; &gt;$</td>
<td></td>
</tr>
<tr>
<td>$i$</td>
<td>$7$</td>
<td></td>
</tr>
<tr>
<td>$j$</td>
<td>$23$</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$&lt;2&gt;$</td>
<td>$&lt;$</td>
</tr>
<tr>
<td>$c$</td>
<td>$&lt; &gt;$</td>
<td>$&gt; ?&gt;$</td>
</tr>
<tr>
<td>$d$</td>
<td>$&lt; ? &gt;$</td>
<td>$&lt; &gt;$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$&lt;15, ?&gt;$</td>
<td>$&lt; ?, 12 &gt;$</td>
</tr>
</tbody>
</table>

We see that, when $s_1 = <7, 23, 2>$ we must have $s_2 = <15, 12>$
A Next Smallest Value (for \(s_1\))

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| smooth(s1, s2); | \(s_1 = \langle 7, 23, 2, 6 \rangle\)  
|               | \(s_2 = \langle 1, 2, 3 \rangle\)  |

\[\]

\(s_1 = \langle 7, 23, 2, 6 \rangle\)  
\(s_2 = \)
A Next Smallest Value (for $s_1$)

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth($s_1$, $s_2$)</td>
<td>$s_1 = &lt;7, 23, 2, 6&gt;$</td>
</tr>
<tr>
<td></td>
<td>$s_2 = &lt;1, 2, 3&gt;$</td>
</tr>
</tbody>
</table>

How do you predict the value of $s_2$?
# Values of Quantified Variables

<table>
<thead>
<tr>
<th>$s1$</th>
<th>$&lt;7, 23, 2, 6&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$i$</td>
<td>$7$</td>
</tr>
<tr>
<td>$j$</td>
<td>$23$</td>
</tr>
<tr>
<td>$b$</td>
<td>$&lt;2, 6&gt;$</td>
</tr>
<tr>
<td>$c$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$d$</td>
<td>$&lt;?, ?&gt;$</td>
</tr>
<tr>
<td>$s2$</td>
<td>$&lt;15, ?, ?&gt;$</td>
</tr>
</tbody>
</table>
# Values of Quantified Variables

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>(&lt;7, 23, 2, 6&gt;)</th>
<th>(&lt;7&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>(&lt; &gt;)</td>
<td>(&lt;7&gt;)</td>
</tr>
<tr>
<td>$i$</td>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>$j$</td>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>$b$</td>
<td>(&lt;2, 6&gt;)</td>
<td>(&lt;6&gt;)</td>
</tr>
<tr>
<td>$c$</td>
<td>(&lt; &gt;)</td>
<td>(&lt;?&gt;)</td>
</tr>
<tr>
<td>$d$</td>
<td>(&lt;?, ?&gt;)</td>
<td>(&lt;?&gt;)</td>
</tr>
<tr>
<td>$s_2$</td>
<td>(&lt;15, ?, ?&gt;)</td>
<td>(&lt;?, 12, ?&gt;)</td>
</tr>
</tbody>
</table>
## Values of Quantified Variables

<table>
<thead>
<tr>
<th></th>
<th>&lt;7, 23, 2, 6&gt;</th>
<th>&lt;7&gt;</th>
<th>&lt;7, 23&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>&lt; &gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>&lt;2, 6&gt;</td>
<td>&lt;6&gt;</td>
<td>&lt; &gt;</td>
</tr>
<tr>
<td>c</td>
<td>&lt; &gt;</td>
<td>&lt;??&gt;</td>
<td>&lt;?, ?&gt;</td>
</tr>
<tr>
<td>d</td>
<td>&lt;?, ?&gt;</td>
<td>&lt;??&gt;</td>
<td>&lt; &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>&lt;15, ?, ?&gt;</th>
<th>&lt;?, 12, ?&gt;</th>
<th>&lt;?, ?, 4&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Values of Quantified Variables

<table>
<thead>
<tr>
<th>$s1$</th>
<th>$&lt;7, 23, 2, 6&gt;$</th>
<th>$s2$</th>
<th>$&lt;15, ?, ?, 4&gt;$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$&lt;&gt;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i$</td>
<td>$7$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$j$</td>
<td>$23$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$&lt;2, 6&gt;$</td>
<td>$&lt;6&gt;$</td>
<td>$&lt;&gt;$</td>
</tr>
<tr>
<td>$c$</td>
<td>$&lt;&gt;$</td>
<td>$&lt;?&gt;$</td>
<td>$&lt;?, ?&gt;$</td>
</tr>
<tr>
<td>$d$</td>
<td>$&lt;?, ?&gt;$</td>
<td>$&lt;?&gt;$</td>
<td>$&lt;&gt;$</td>
</tr>
</tbody>
</table>

We see that, when $s1 = <7, 23, 2, 6>$ we must have $s2 = <15, 12, 4>$.
Conclusion

• So, stated in English, what does smooth do?
• Even when you can “see the pattern”, it’s surprisingly difficult to phrase it in a simple way...