5.2 Computing with XSLT

- XSLT is a declarative rule-based language
  - for a special purpose: XML transformation
  - Can we use XSLT for procedural computing?
  - What is the exact computational power of XSLT?
- We've seen some programming-like features:
  - iteration over source nodes (xsl:for-each)
  - conditional evaluation (xsl:if and xsl:choose)

Visibility of Variable Bindings

- The binding is visible in following siblings of xsl:variable, and in their descendants:

Solution (1/2)

- Pass the column-count to a named template which generates an appropriate number of \( \text{'l}' \)s:

Solution 2/2: Recursive \textit{gen-cols}

- Alan Turing 1936/37
- formal model of algorithms
- primitive but powerful enough to simulate any computation expressible in any algorithmic model (Church/Turing thesis)

Computational power of XSLT

- XSLT seems quite powerful, but how powerful is it?
  - Implementations may provide extension mechanisms, e.g., to call arbitrary Java methods
  - Are there limits to XSLT processing that we can do without extensions?
- We'll see that any algorithmic computation can be simulated with XSLT
  - shown indirectly, through simulating Turing machines by XSLT

Turing machine

- A finite set of states
- unlimited tape of cells for symbols, examined by a tape head
The “transition” template

- **Parameters:**
  - state: the current state
  - left: contents of the tape up to the tape head
  - right: contents of the tape starting at the cell pointed by the tape head

- Transition simulates a single transition step; calls itself with updated parameters

Simulating a single transition (1/2)

```xml
<xsl:template name="transition">
  <!-- parameters and trace output omitted -->
  <xsl:choose>
    <xsl:when test="@state='YES'">
      <xsl:call-template name="transition">
        <xsl:with-param name="newstate" select="$newstate"/>
        <xsl:with-param name="newright" select="$newright"/>
        <xsl:with-param name="newleft" select="concat($left, 2)"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="transition">
        <xsl:with-param name="newstate" select="$newstate"/>
        <xsl:with-param name="newright" select="concat($right, 2)"/>
        <xsl:with-param name="newleft" select="concat($left, 2)"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Simulating a single transition (2/2)

```xml
<!-- Then call "transition" with new params: -->
<xsl:call-template name="transition">
  <xsl:with-param name="state" select="$newstate"/>
  <xsl:with-param name="left" select="$newleft"/>
  <xsl:with-param name="right" select="$newright"/>
</xsl:call-template>
</xsl:when>
```

Sample trace of the simulation

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:template name="transition">
  <M state="#[mark]a#"/>
  <M state="#[move_a]b#"/>
  <M state="#[test_a]a#"/>
  <M state="#[mark]b#"/>
  <M state="#[move_b]b#"/>
  <M state="#[test_b]###"/>
  <M state="#[mark]b#"/>
</xsl:template>
```
What does this mean?

- XSLT has **full algorithmic power**
  - (It is "Turing-complete")
  - Is this intentional?
    » Inconvenient as a general-purpose programming language!
  - Impossible to recognise non-terminating transformations automatically
    (« the "halting problem" has no algorithmic solution)
    » Malicious hacker could cause "denial-of-service" through non-terminating style sheets