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 1's:

Computational power of XSLT

- XSLT seems quite powerful, but how powerful is it?
  - Implementations 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 plain XSLT
  - shown indirectly, through simulating Turing machines by XSLT

Solution 2/2: Recursive 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)
- Turing machine
  - A finite set of states
  - unlimited tape of cells for symbols, examined by a tape head
Illustration of a Turing Machine

SDPL 2005 Notes 5.2: Computing with XSLT

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

Overall structure of the simulation

<xsl:template name="transition">
  <!-- params and trace output omitted -->
  <xsl:choose>
    <xsl:when test="$state='YES'">
      <ACCEPT />
    </xsl:when>
    <xsl:otherwise>
      <REJECT />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

Updating the representation of the tape

For each right-move $\sigma(q, a) = (q', b, right)$, concatenate ‘b’ at the end of $\$left$ and drop the first character of $\$right$

Left-moves $\sigma(q, a) = (q, b, left)$ in a similar manner:
- drop the last character of $\$left$, and concatenate it in front of $\$right$ whose first character has been replaced by ‘b’

Example: a TM for palindromes over alphabet \{a, b\}; (# used for denoting empty cells)

Simulating a single transition (1/2)

<!-- $\sigma$ (mark, a) = (move_a, #, right): -->
<xsl:when test="$\$state='mark' and starts-with($\$right, 'a')">
  <!-- First update the parameters: -->
  <xsl:variable name="newstate" select="move_a"/>
  <xsl:variable name="newleft" select="concat($\$left, '#')"/>
  <xsl:variable name="newright" select="substring($\$right, 2)"/>
</xsl:when>

Simulating a single transition (2/2)

<!-- 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

Saxon dummy.xml tm-palindr.xsl input-tape=aba

state shown as tape-left[state]tape-right
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)
    - Could cause "denial-of-service" through non-terminating style sheets