

Flattening and unflattening XML markup

A Zen garden of “raising” methods

David J. Birnbaum, Elisa Beshero-Bondar,
and C. M. Sperberg-McQueen

Balisage 2018

Flattening and raising

- We may use empty elements (*Trojan horse pairs* or [single] *milestones*) to work around overlap problems
- We may flatten container elements and raise empty ones to create or invert a hierarchy
- Fully inverting a hierarchy and raising scattered flattened elements may require different strategies

Flattening and raising

- We may use empty elements (*Trojan horse pairs* or [single] *milestones*) to work around overlap problems
 - We may flatten container elements and raise empty ones to create or invert a hierarchy
 - Fully inverting a hierarchy and raising scattered flattened elements may require different strategies
-
- **Example**
 - We want to tag the splitting of an element, but we can't wrap an end tag followed by a start tag in a container
 - This arises when we collate variant witnesses; we want to wrap each variant reading in the equivalent of TEI `<rdg>`

Flattening and raising

- We may use empty elements (*Trojan horse pairs* or [single] *milestones*) to work around overlap problems
- We may flatten container elements and raise empty ones to create or invert a hierarchy
- Fully inverting a hierarchy and raising scattered flattened elements may require different strategies



- **Example**
- We want to tag the splitting of an element, but we can't wrap an end tag followed by a start tag in a container
- This arises when we collate variant witnesses; we want to wrap each variant reading in the equivalent of TEI `<rdg>`

Flattening with intent to raise

Original content elements

```
<?xml version="1.0" encoding="UTF-8"?>


<cit>
    <quote>
      <lg>
        <l>Like one who, on a lonely road,</l>
        <l>Doth walk in fear and dread,</l>
        <l>And, having once turn'd round, walks on,</l>
        <l>And turns no more his head;</l>
        <l>Because he knows a frightful fiend</l>
        <l>Doth close behind him tread*.</l>
      </lg>
    </quote>
    <note> *
      <bibl>Coleridge's "Ancient Mariner."</bibl>
    </note>
  </cit>


```

Collating *Frankenstein*

Q. How do you get a text-collation tool to tell you about differences in (say) paragraphing or lineation?

Q. How do you ensure that XML collation output with variants that span line boundaries is well-formed?

Collating *Frankenstein*

Q. How do you get a text-collation tool to tell you about differences in (say) paragraphing or lineation?

Q. How do you ensure that XML collation output with variants that span line boundaries is well-formed?

Answers:

- Convert (some) start- and end-tags from markup into character data
- Flatten (some) start- and end-tags to empty *virtual elements*

Flattening with intent to raise

... converted to virtual elements (*)

```
<?xml version="1.0" encoding="UTF-8"?><p
    xmlns:th="http://www.blackmesatech.com/2017/nss/trojan-horse">
    <cit xml:id="fThomas_C10-cit_1" th:sID="d1e3"/>
    <quote xml:id="fThomas_C10-quote_1" th:sID="d1e5"/>
    <lg xml:id="fThomas_C10-lg_1" th:sID="d1e7"/>
    <l xml:id="fThomas_C10-l_1" th:sID="d1e9"/>
        Like one who, on a lonely road, <l th:eID="d1e9"/>
    <l xml:id="fThomas_C10-l_2" th:sID="d1e12"/>
        Doth walk in fear and dread, <l th:eID="d1e12"/>
    <l xml:id="fThomas_C10-l_3" th:sID="d1e15"/>
        And, having once turn'd round, walks on, <l th:eID="d1e15"/>
    <l xml:id="fThomas_C10-l_4" th:sID="d1e18"/>
        And turns no more his head; <l th:eID="d1e18"/>
    <l xml:id="fThomas_C10-l_5" th:sID="d1e21"/>
        Because he knows a frightful fiend <l th:eID="d1e21"/>
    <l xml:id="fThomas_C10-l_6" th:sID="d1e25"/>
        Doth close behind him tread*. <l th:eID="d1e25"/>
    <lg th:eID="d1e7"/>
    <quote th:eID="d1e5"/>
    <note xml:id="fThomas_C10-note_1" th:sID="d1e30"/> *
    <bibl xml:id="fThomas_C10-bibl_1">
        Coleridge's "Ancient Mariner."
    </bibl>
    <note th:eID="d1e30"/>
    <cit th:eID="d1e3"/>
</p>
```

Context:

preparing for
collation:

(we expect
witnesses to alter
line boundaries
and structure)

What we want: virtual overlapping hierarchies

Context: "hotspot" variant location markers inserted
in edition files following the collation process

```
<lg xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1">
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_11"> Like
    <seg xml:id="C10_app435-f1818_end"/>one
    <seg xml:id="C10_app437-f1818_start"/>who,
    <seg xml:id="C10_app437-f1818_end"/>on a
    <seg xml:id="C10_app439-f1818_start"/>lonely road,</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_12">
    Doth <seg xml:id="C10_app439-f1818_end"/>walk in fear and
    <seg xml:id="C10_app441-f1818_start"/>dread,</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_13">
    And, <seg xml:id="C10_app441-f1818_end"/>having once
    <seg xml:id="C10_app443-f1818_start"/>turn'd
    <seg xml:id="C10_app443-f1818_end"/>
    <seg xml:id="C10_app444-f1818_start"/>round, walks on,</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_14">
    And <seg xml:id="C10_app444-f1818_end"/>turns no more his
  <seg xml:id="C10_app446-f1818_start"/>head;</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_15">
    Because <seg xml:id="C10_app446-f1818_end"/>he knows a frightful
    <seg xml:id="C10_app448-f1818_start"/>fiend</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_16">
    Doth <seg xml:id="C10_app448-f1818_end"/>close behind him
    <seg xml:id="C10_app450-f1818_start"/>tread*.</l>
</lg>
```

What we want: virtual overlapping hierarchies

Context: "hotspot" variant location markers inserted
in edition files following the collation process

```
<lg xml:id="novell1_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1">
  <l xml:id="novell1_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_11"> Like
    <seg xml:id="C10_app435-f1818_end"/>one
    <seg xml:id="C10_app437-f1818_start"/>who,
    <seg xml:id="C10_app437-f1818_end"/>on a
    <seg xml:id="C10_app439-f1818_start"/>lonely road,</l>
    <l xml:id="novell1_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_12">
      Doth <seg xml:id="C10_app439-f1818_end"/>walk in fear and
      <seg xml:id="C10_app441-f1818_start"/>dread,</l>
    <l xml:id="novell1_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_13">
      And, <seg xml:id="C10_app441-f1818_end"/>having once
      <seg xml:id="C10_app443-f1818_start"/>turn'd
      <seg xml:id="C10_app443-f1818_end"/>
      <seg xml:id="C10_app444-f1818_start"/>round, walks on,</l>
    <l xml:id="novell1_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_14">
      And <seg xml:id="C10_app444-f1818_end"/>turns no more his
    <seg xml:id="C10_app446-f1818_start"/>head;</l>
    <l xml:id="novell1_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_15">
      Because <seg xml:id="C10_app446-f1818_end"/>he knows a frightful
      <seg xml:id="C10_app448-f1818_start"/>fiend</l>
    <l xml:id="novell1_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_16">
      Doth <seg xml:id="C10_app448-f1818_end"/>close behind him
      <seg xml:id="C10_app450-f1818_start"/>tread*.</l>
  </lg>
```

What we want: virtual overlapping hierarchies

Context: "hotspot" variant location markers inserted
in edition files following the collation process

```
<lg xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1">
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_11"> Like
    <seg xml:id="C10_app435-f1818_end"/>one
    <seg xml:id="C10_app437-f1818_start"/>who,
    <seg xml:id="C10_app437-f1818_end"/>on a
    <seg xml:id="C10_app439-f1818_start"/>lonely road,</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_12">
    Doth <seg xml:id="C10_app439-f1818_end"/>walk in fear and
    <seg xml:id="C10_app441-f1818_start"/>dread,</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_13">
    And, <seg xml:id="C10_app441-f1818_end"/>having once
    <seg xml:id="C10_app443-f1818_start"/>turn'd
    <seg xml:id="C10_app443-f1818_end"/>
    <seg xml:id="C10_app444-f1818_start"/>round, walks on,</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_14">
    And <seg xml:id="C10_app444-f1818_end"/>turns no more his
<seg xml:id="C10_app446-f1818_start"/>head;</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_15">
    Because <seg xml:id="C10_app446-f1818_end"/>he knows a frightful
    <seg xml:id="C10_app448-f1818_start"/>fiend</l>
  <l xml:id="novell_letter4_chapter4_div4_div4_p8_cit1_quote1_lg1_16">
    Doth <seg xml:id="C10_app448-f1818_end"/>close behind him
    <seg xml:id="C10_app450-f1818_start"/>tread*.</l>
</lg>
```

The problem in a nutshell

Given Trojan Horse* elements of the form

```
... <e th:sID="x"/> ... <e th:eID="x"/> ...
```

produce

```
... <e> ... </e> ...
```

* DeRose 2004 “Markup overlap: a review and a horse,”
Extreme Markup Languages 2004

XML processing and raising

XML processing may involve

1. Reading the serialization (string)
2. Parsing the input to identify events (SAX)
3. Building (and using) a tree (DOM)

Each of these stages can be accessed for raising

Seven* raising solutions: a zen garden walk**

- Thinking in strings
 - Regular expression matching
- Thinking in parser events
 - pulldom (Python)
- Thinking in nodes
 - Right-sibling traversal (XSLT/XQuery)
 - Inside-out recognition (XSLT/XQuery)
 - Outside-in recognition (XSLT/XQuery)
 - Accumulators (XSLT 3.0)
 - Tumbling Windows (XQuery 3.1)



Seven* raising solutions: a zen garden walk**

- Thinking in strings
 - Regular expression matching
- Thinking in parser events
 - pulldom (Python)
- Thinking in nodes
 - Right-sibling traversal (XSLT/XQuery)
 - Inside-out recognition (XSLT/XQuery)
 - Outside-in recognition (XSLT/XQuery)
 - Accumulators (XSLT 3.0)
 - Tumbling Windows (XQuery 3.1)

* Actually, there are more, but seven may be enough.



Seven* raising solutions: a zen garden walk**

- Thinking in strings
 - Regular expression matching
- Thinking in parser events
 - pulldom (Python)
- Thinking in nodes
 - Right-sibling traversal (XSLT/XQuery)
 - Inside-out recognition (XSLT/XQuery)
 - Outside-in recognition (XSLT/XQuery)
 - Accumulators (XSLT 3.0)
 - Tumbling Windows (XQuery 3.1)



* Actually, there are more, but seven may be enough.

** We only have time to visit five of these on our short walk today. For the others, read our paper / come talk to us afterwards!

For each solution . . .

- How does it work?
- How are pairs recognized?
- In what order are pairs raised to elements?



Thinking in strings

regular expression matching

Regular expression matching

- XML is not a *regular language*
- But matching specific tags as strings is not the same as parsing XML
- Guidelines
 - Dot-matches-all mode
 - Watch for the element end-delimiter (`>`)
 - Match non-greedily
 - Use `\s+` (not space) for whitespace
 - (Canonicalize the XML first)



Regular expression matching

Raise start markers

Match

```
(<[ ^>]+?)th:sID\s*=\\s*[ '\"']\w+?[ '\"'](.*?)/(>)
```

Replace

```
\1\2\3
```

Raise end markers

Match

```
(<)(\s+?)\s+[ ^>]*?th:eID=[ '\"']\w+[ '\"'][ ^>]*?/(>)
```

Replace

```
\1/\2\3
```

Thinking in parser events

Python pulldom

Python XML processing

Input: pulldom (SAX-like event processor)

XML output

- minidom (or lxml or xml.sax.saxutils.XMLGenerator())
 - Add nodes to the tree
 - Maintain context on a stack
- Only partially namespace-aware

< > or: String output </ >

- Construct tags (not elements) on element-start and element-end events
- Easy to create malformed output (esp. attribute value quoting)

A Python pulldom application

A Python pulldom application

SAX-like event interface, with ability to pull in DOM-like subtrees

A Python pulldom application

SAX-like event interface, with ability to pull in DOM-like subtrees

- When you see a start-marker

- start an element
- push it on a stack
- start collecting content for it

```
<lg th:sID="id1" />
```

A Python pulldom application

SAX-like event interface, with ability to pull in DOM-like subtrees

- When you see a start-marker

- start an element
 - push it on a stack
 - start collecting content for it

```
<lg th:sID="id1" />
```

- When you see an end-marker

- stop collecting content (close the element)
 - pop the stack

```
<lg th:eID="id1" />
```

A Python pulldom application

SAX-like event interface, with ability to pull in DOM-like subtrees

- When you see a start-marker

- start an element
 - push it on a stack
 - start collecting content for it

```
<lg th:sID="id1" />
```

- When you see an end-marker

- stop collecting content (close the element)
 - pop the stack

```
<lg th:eID="id1" />
```

- When you see anything else

- add it as last child of the element on top of the stack

pulldom source

```
for event, node in parseString(input.read()):
    if event == START_ELEMENT:
        # process pseudo-end-tags on END_ELEMENT event
        if not node.hasAttribute('th:eID'):
            open_elements.peek().appendChild(node)
            open_elements.push(node)
    elif event == END_ELEMENT:
        # don't declare now-unused th: namespace
        if node.hasAttribute('xmlns:th'):
            node.removeAttribute('xmlns:th')
        # can't remove @start until we're done with the node
        if node.hasAttribute('th:sID'):
            node.removeAttribute('th:sID')
        # pop only on container elements and Trojan end-tags
        else:
            open_elements.pop()
    elif event == CHARACTERS:
        # create and add text() node for character data
        t = d.createTextNode(node.data)
        open_elements.peek().appendChild(t)

result = open_elements.pop().toxml() # root is at top of stack
```

The core of the Python app

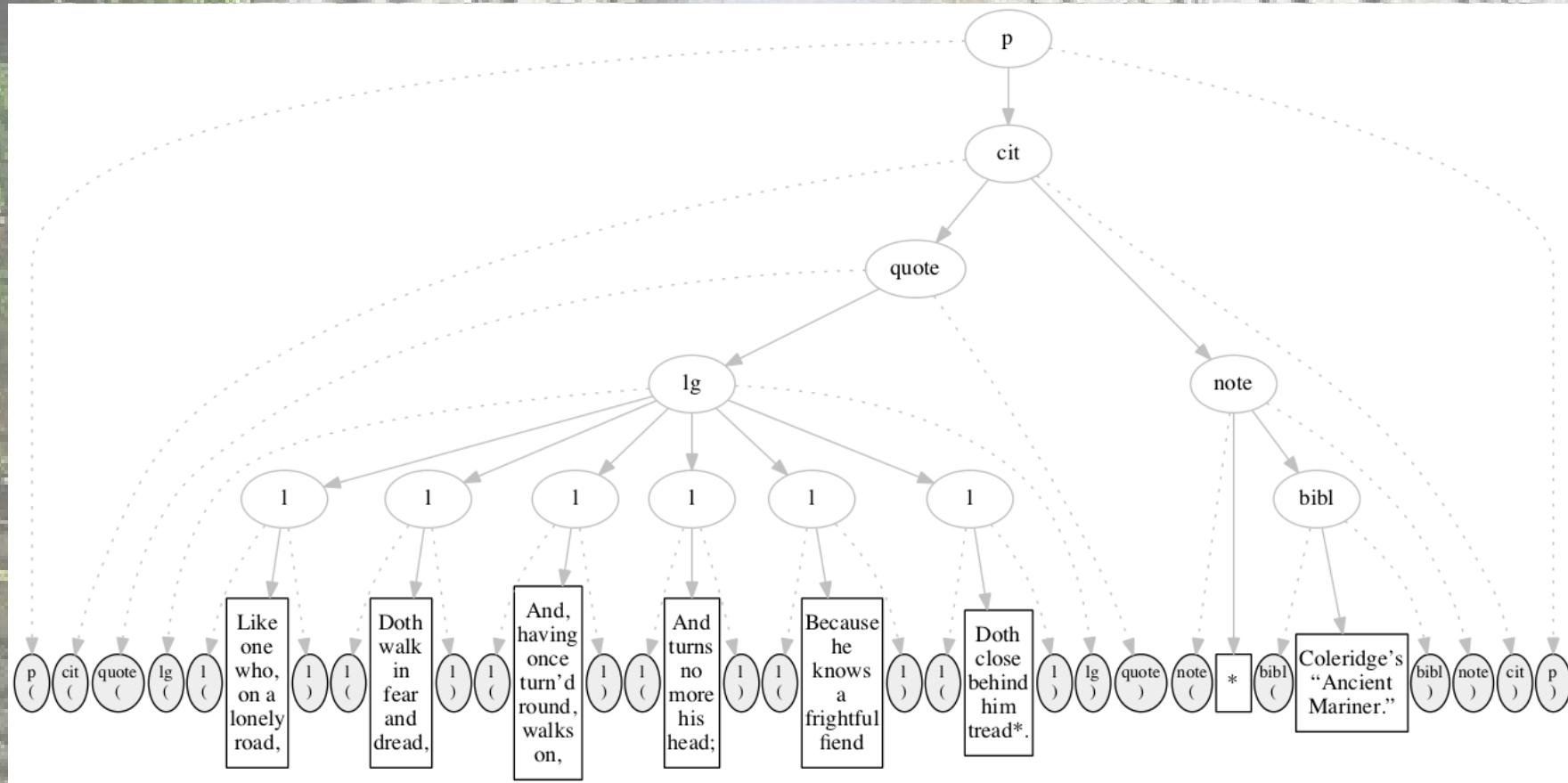
Raising with XSLT

- left-to-right
- inside-out
- outside-in
- accumulators

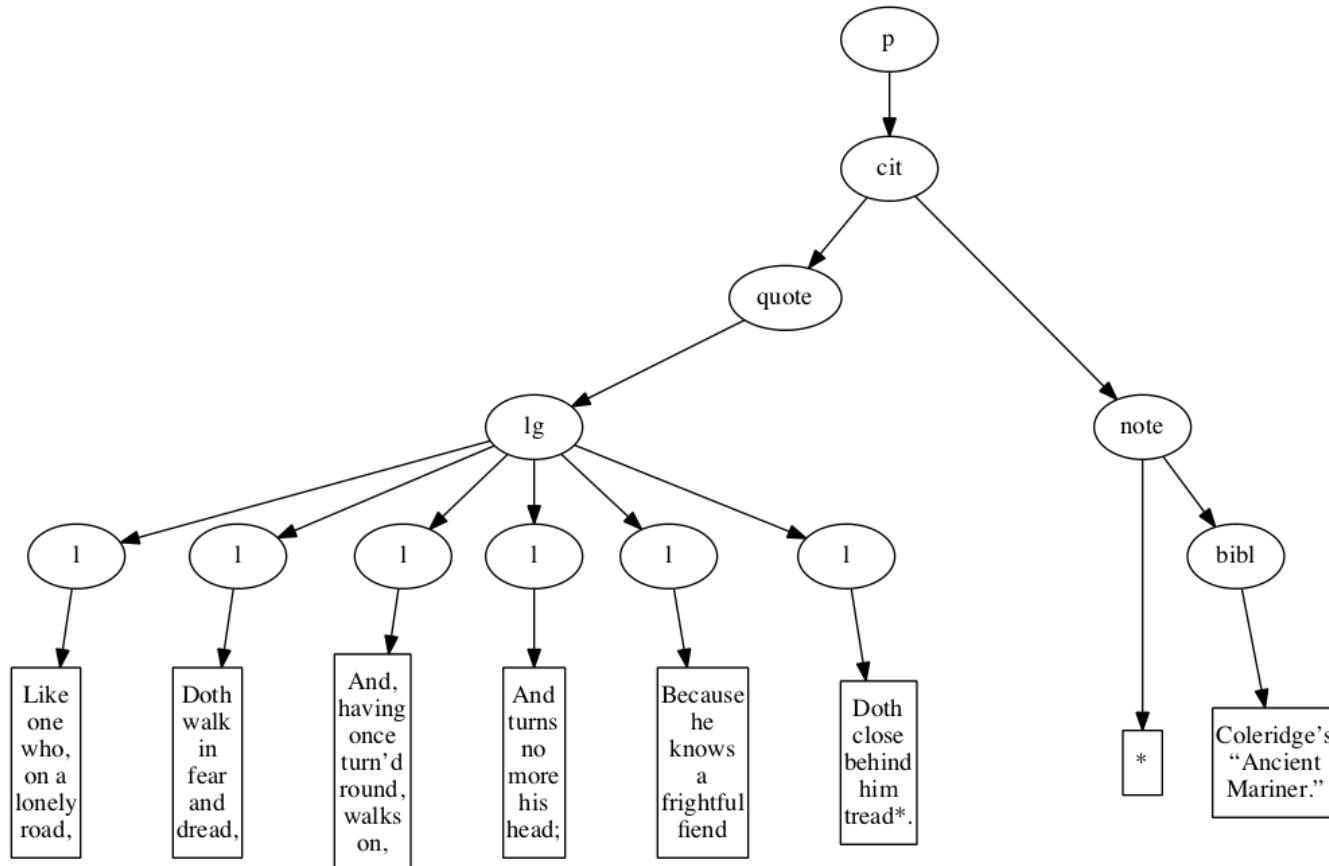


**From flattened to raised,
via nodes**

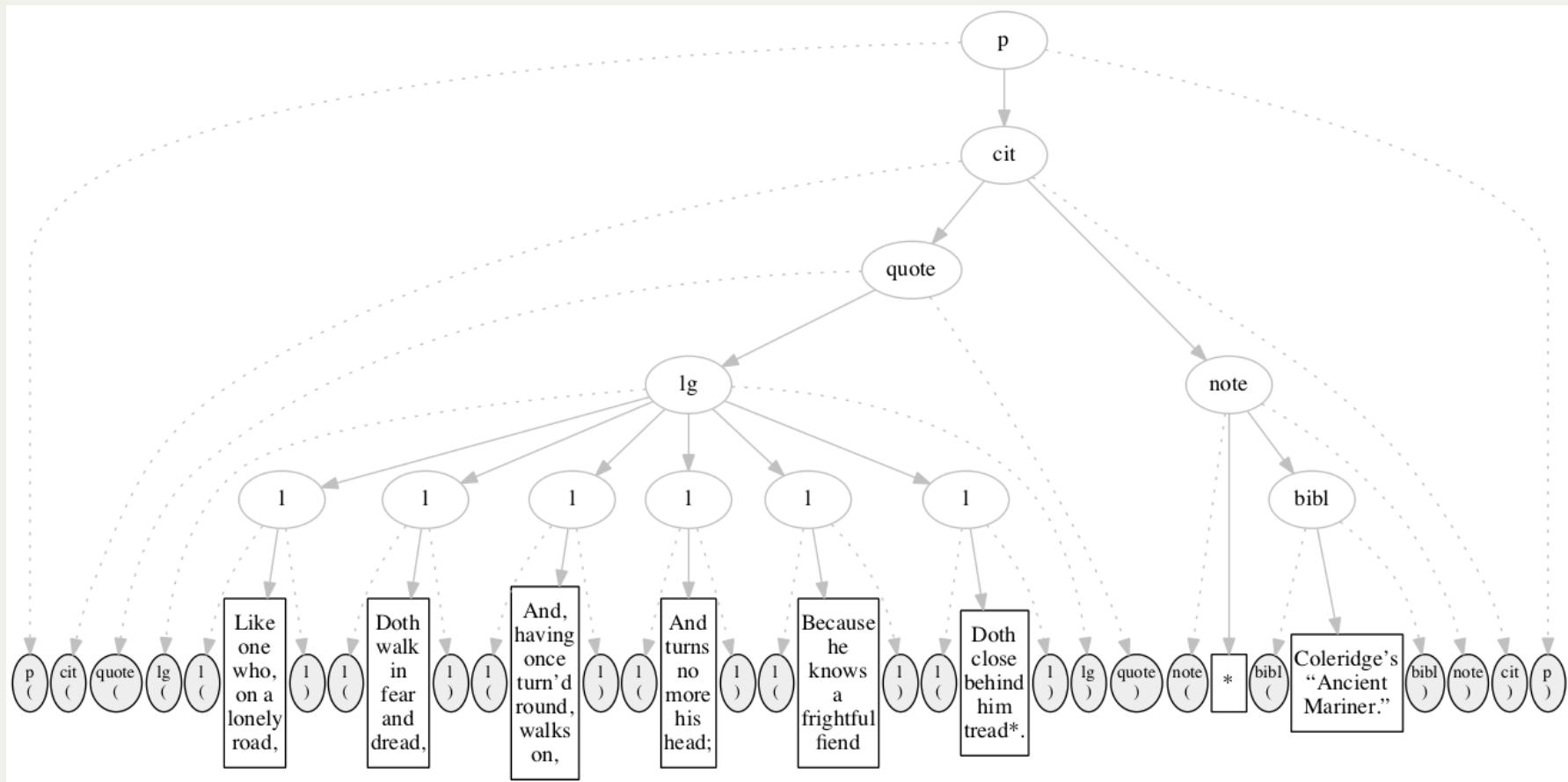
From flattened to raised, via nodes



From flattened to raised, via nodes

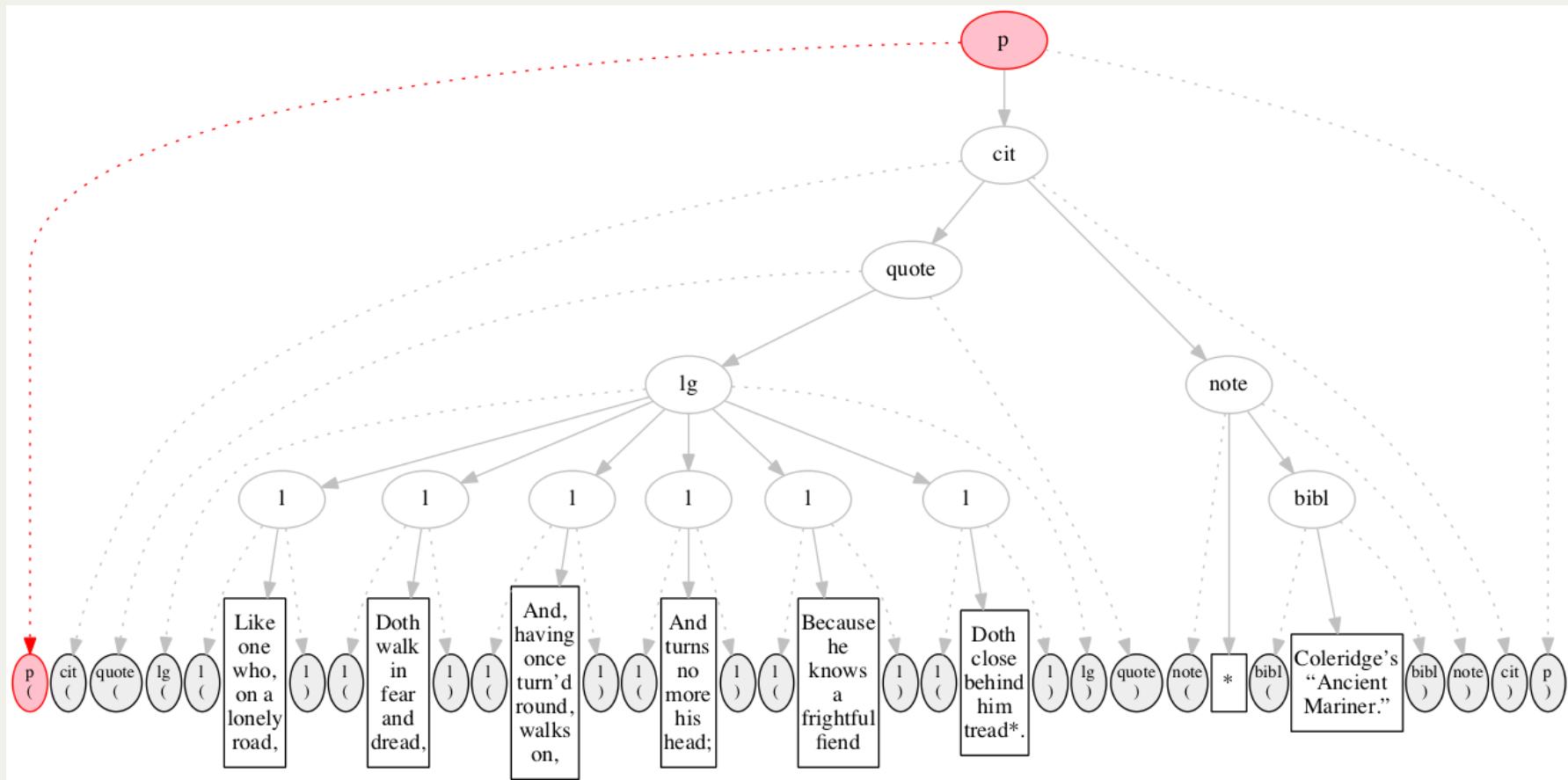


Left to right



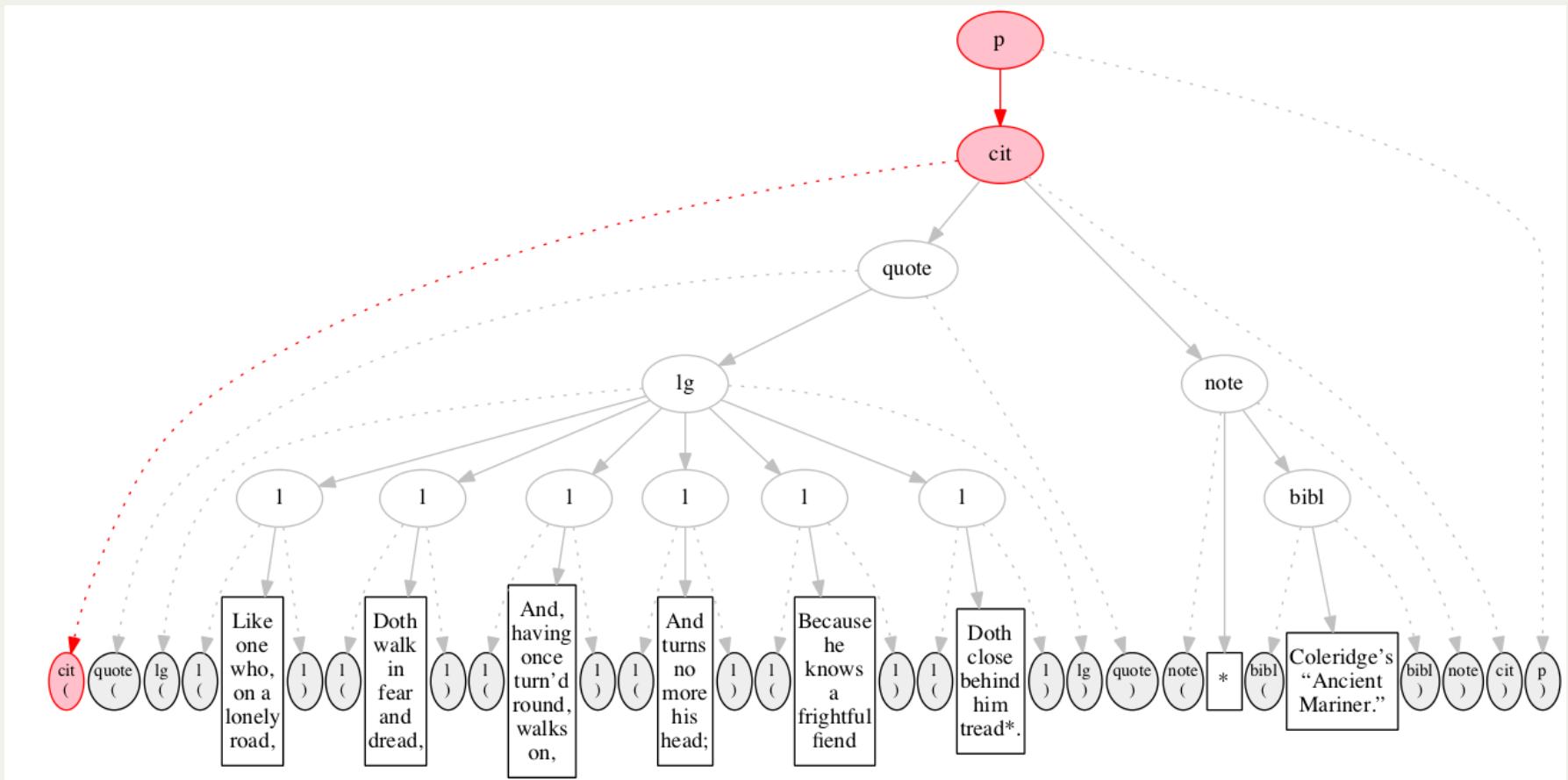
At the outset, we have a flat sequence of nodes.

Left to right



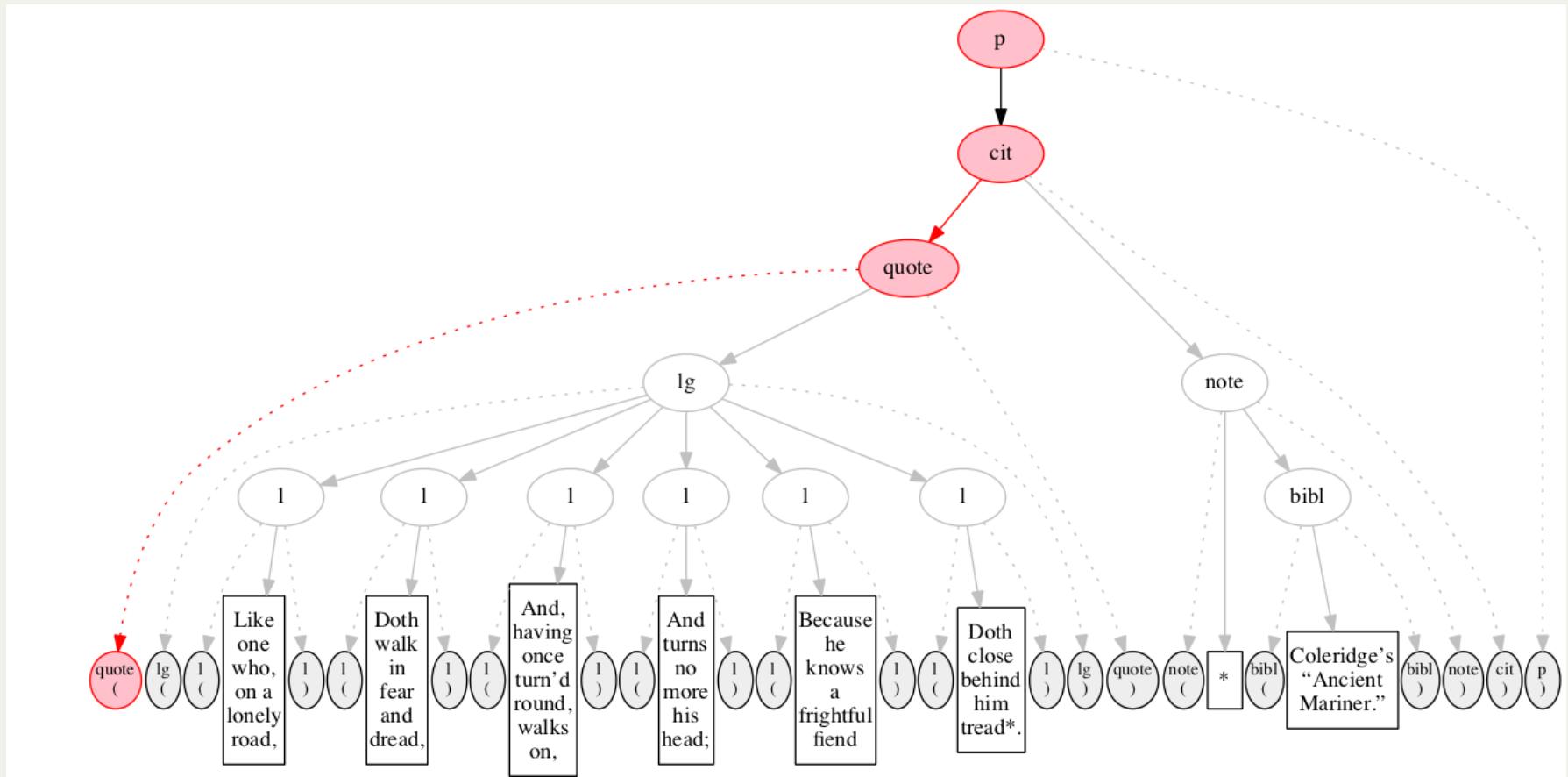
We first encounter the start-marker for the paragraph.

Left to right



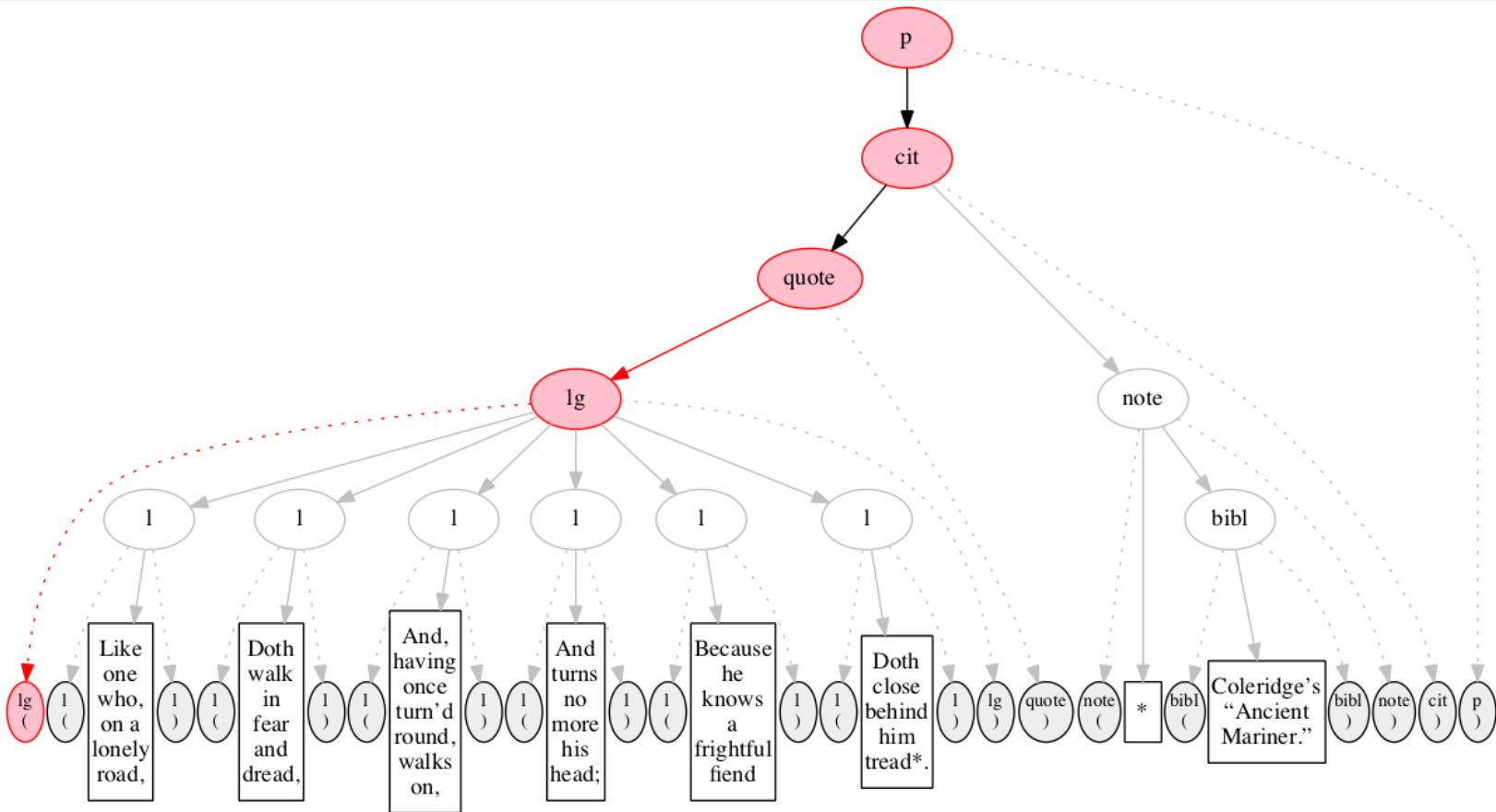
We then encounter a start-marker for a citation. The paragraph has been started but not finished.

Left to right



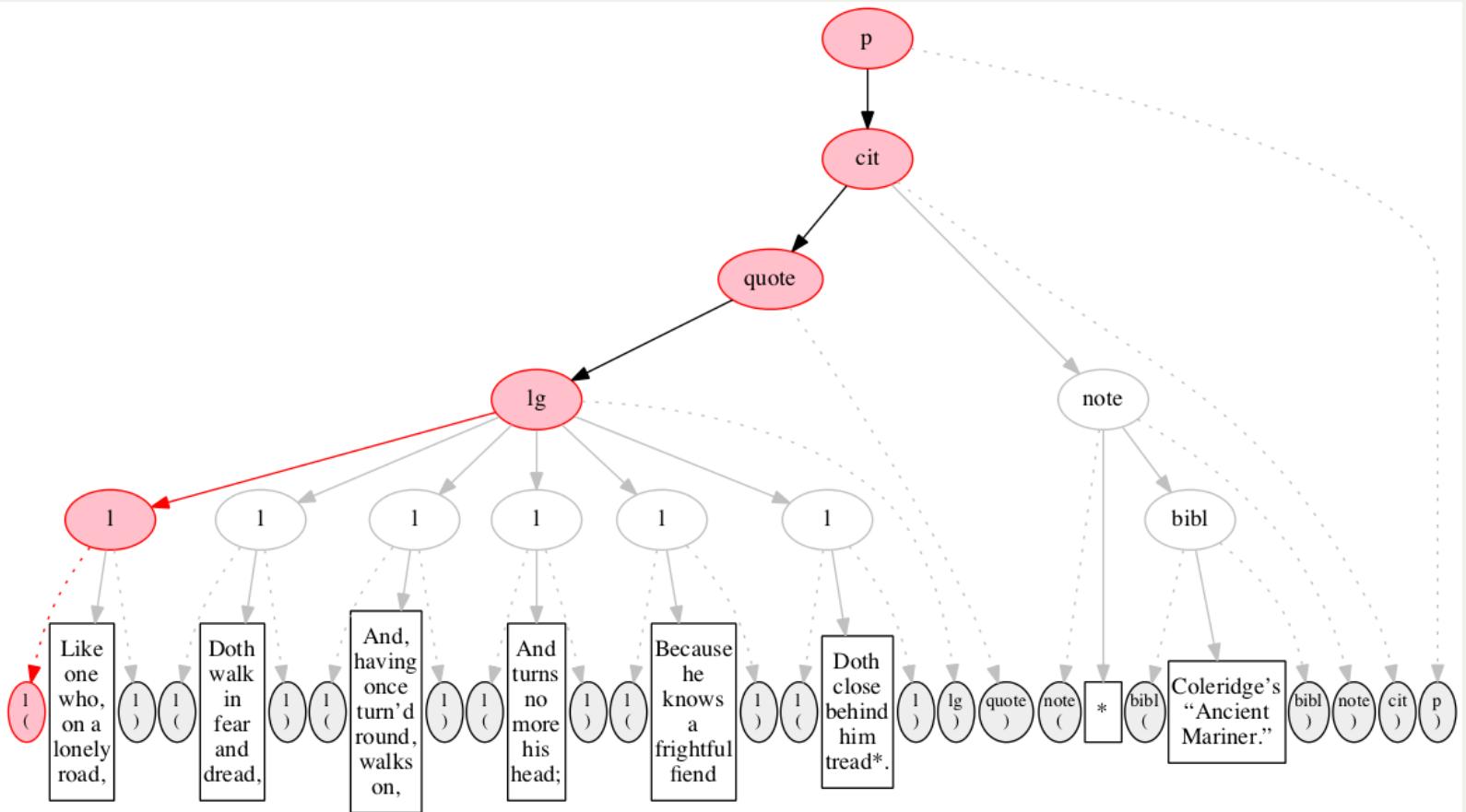
We then encounter a start-marker for a quotation.

Left to right



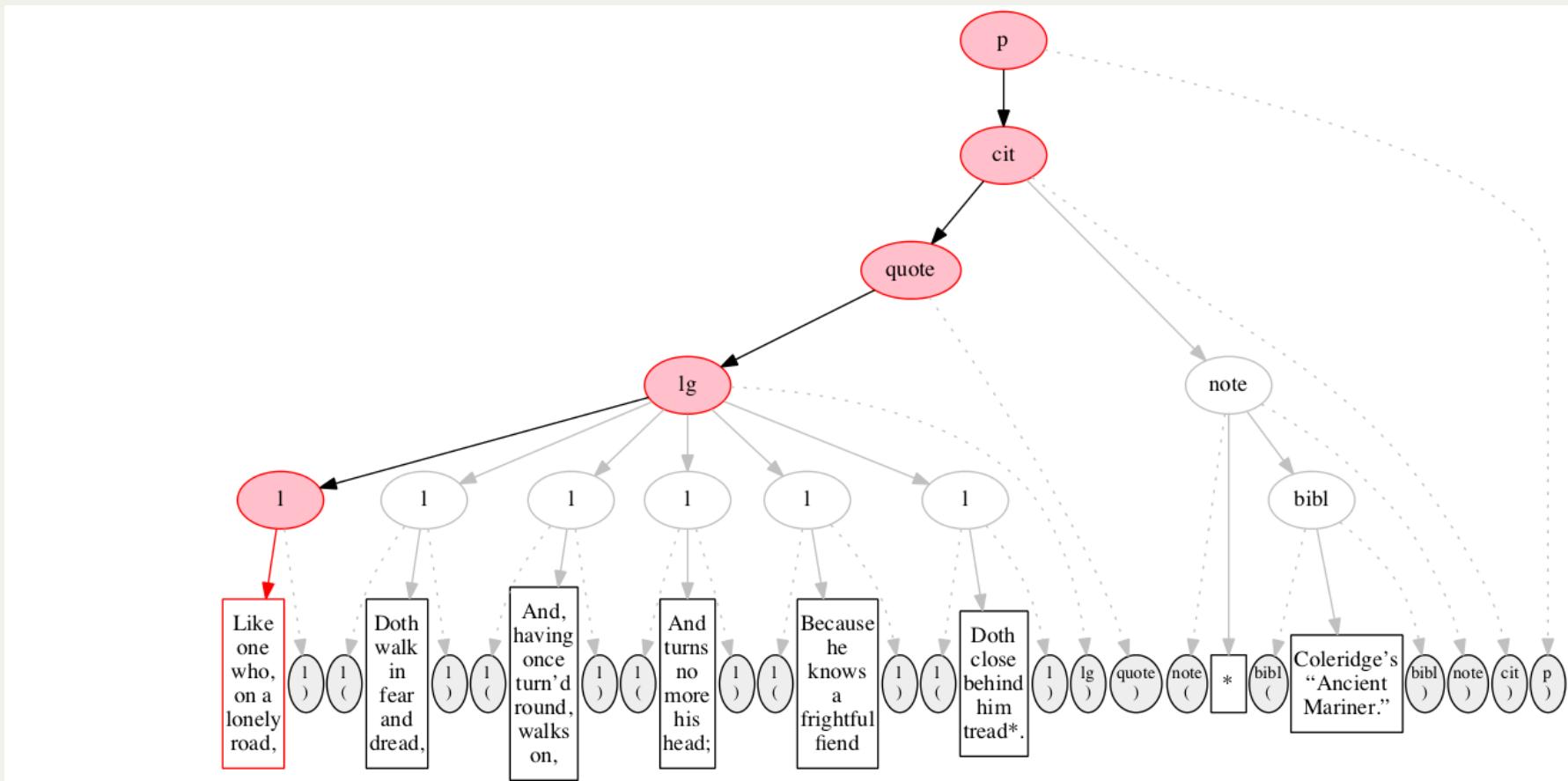
We then encounter a start-marker for a line group.

Left to right



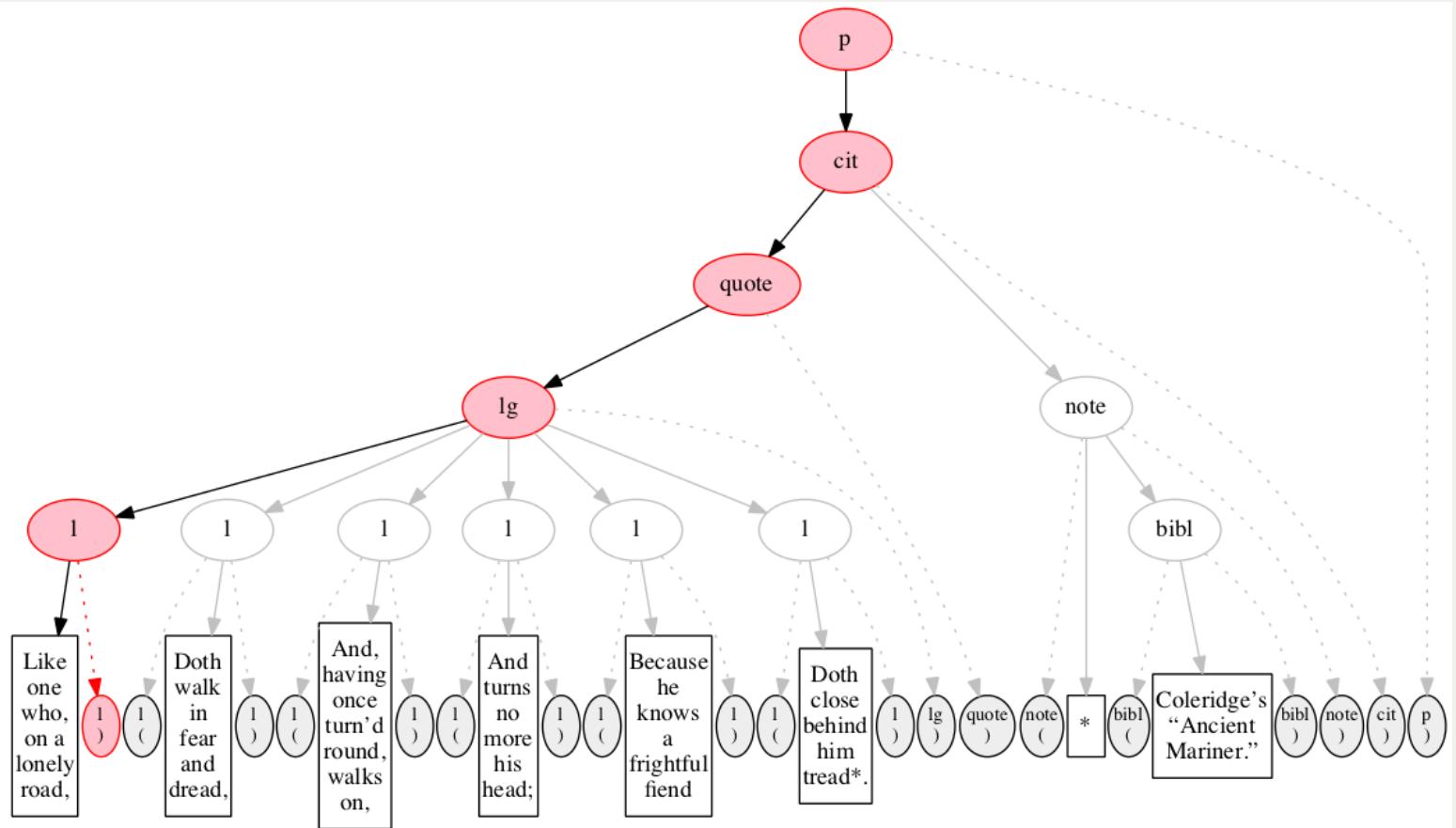
We then encounter a start-marker for the first line of verse.

Left to right



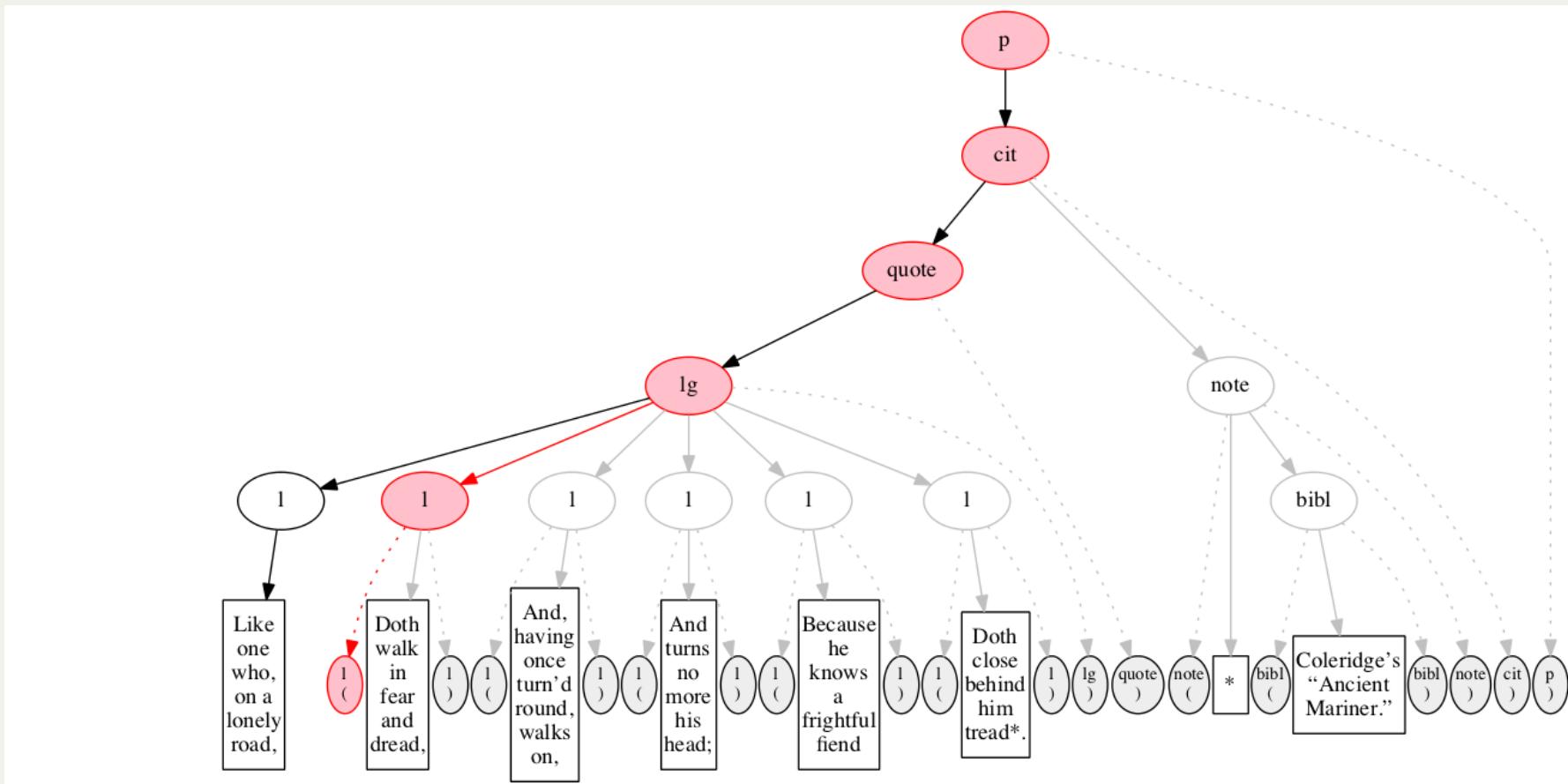
And finally some text data: the text of the first line of verse.

Left to right



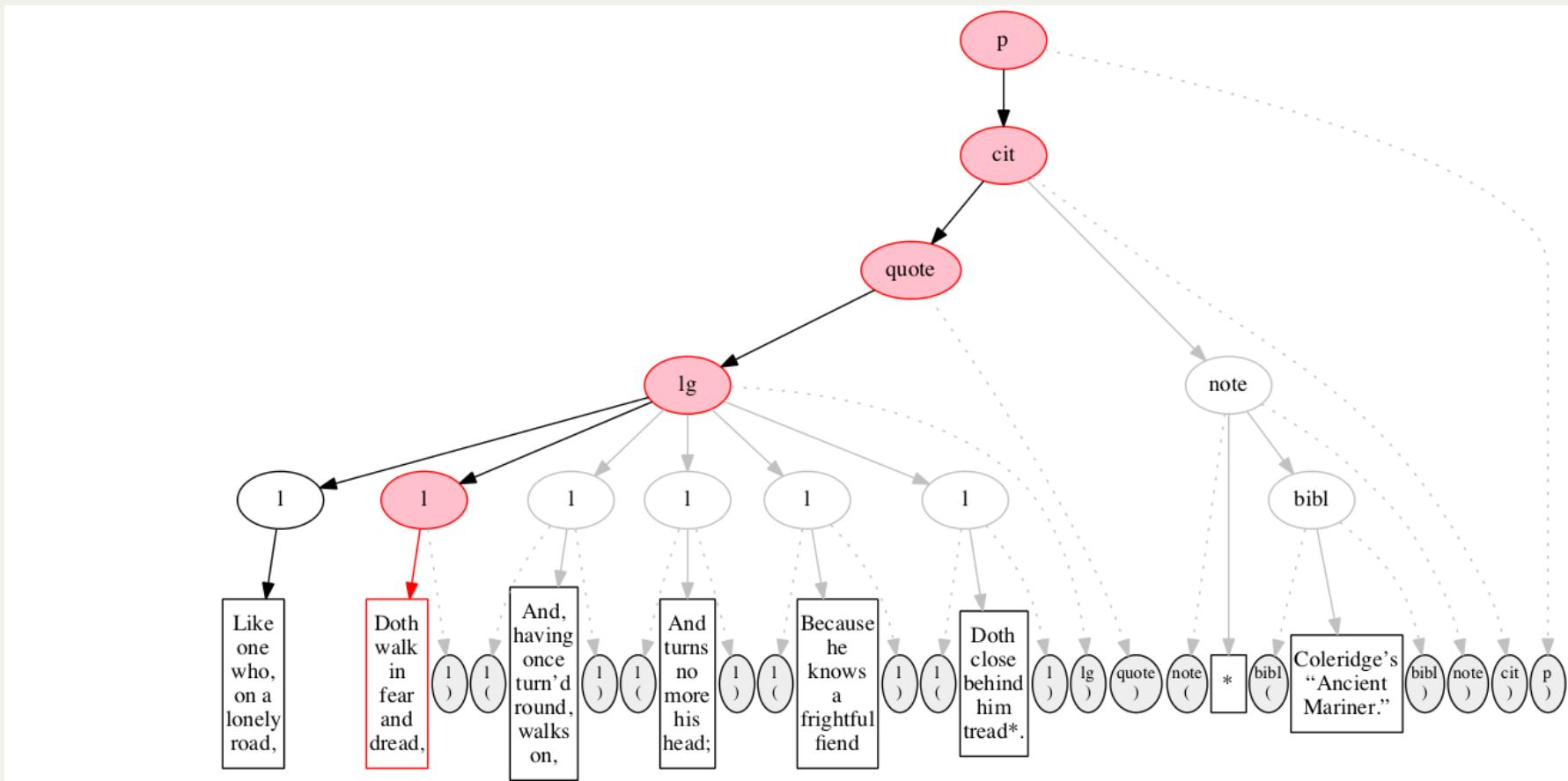
End-marker for the first line.

Left to right



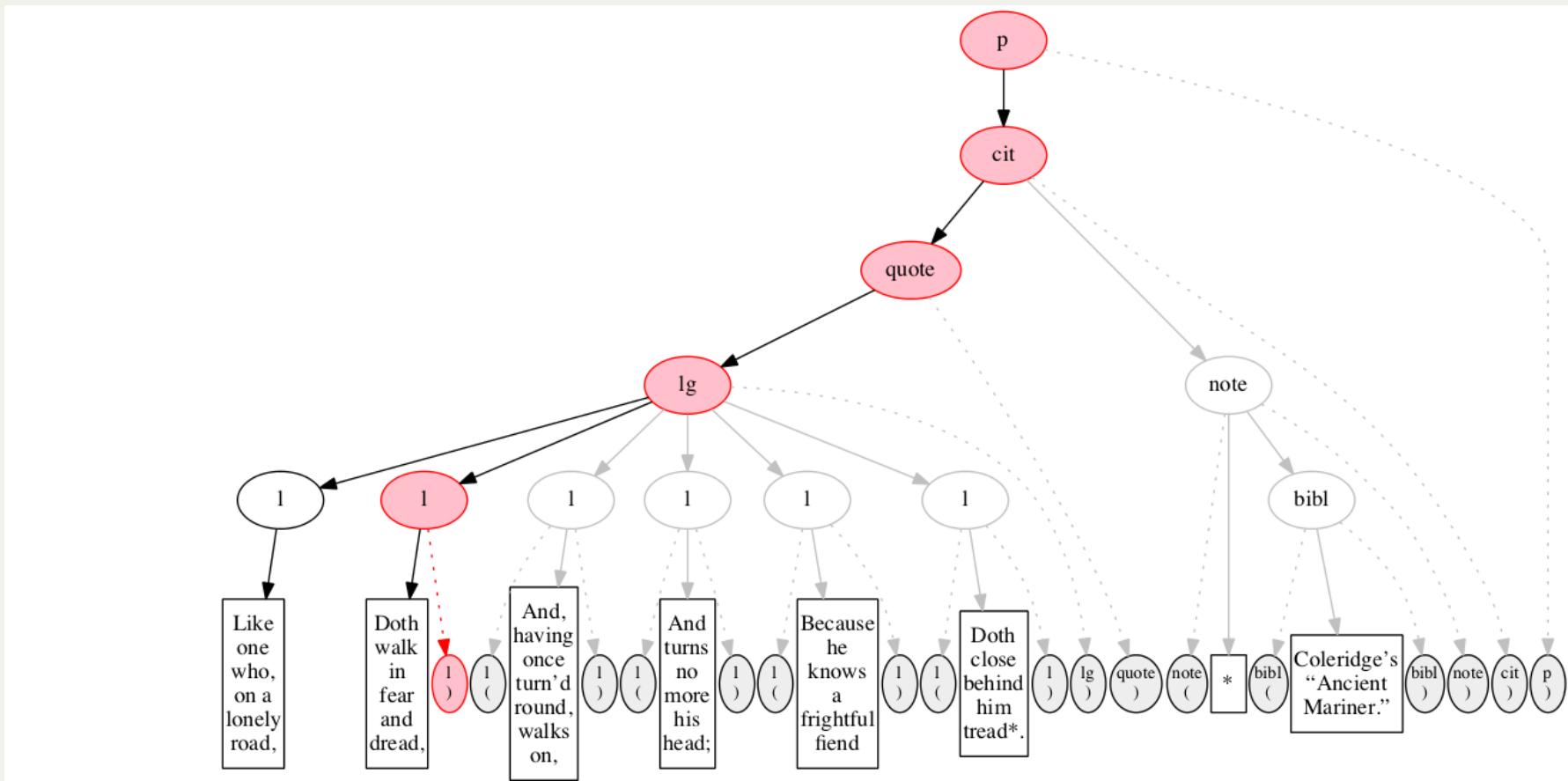
We then encounter a start-marker for the second line of verse. The first line is now finished (as signaled by the change in color).

Left to right



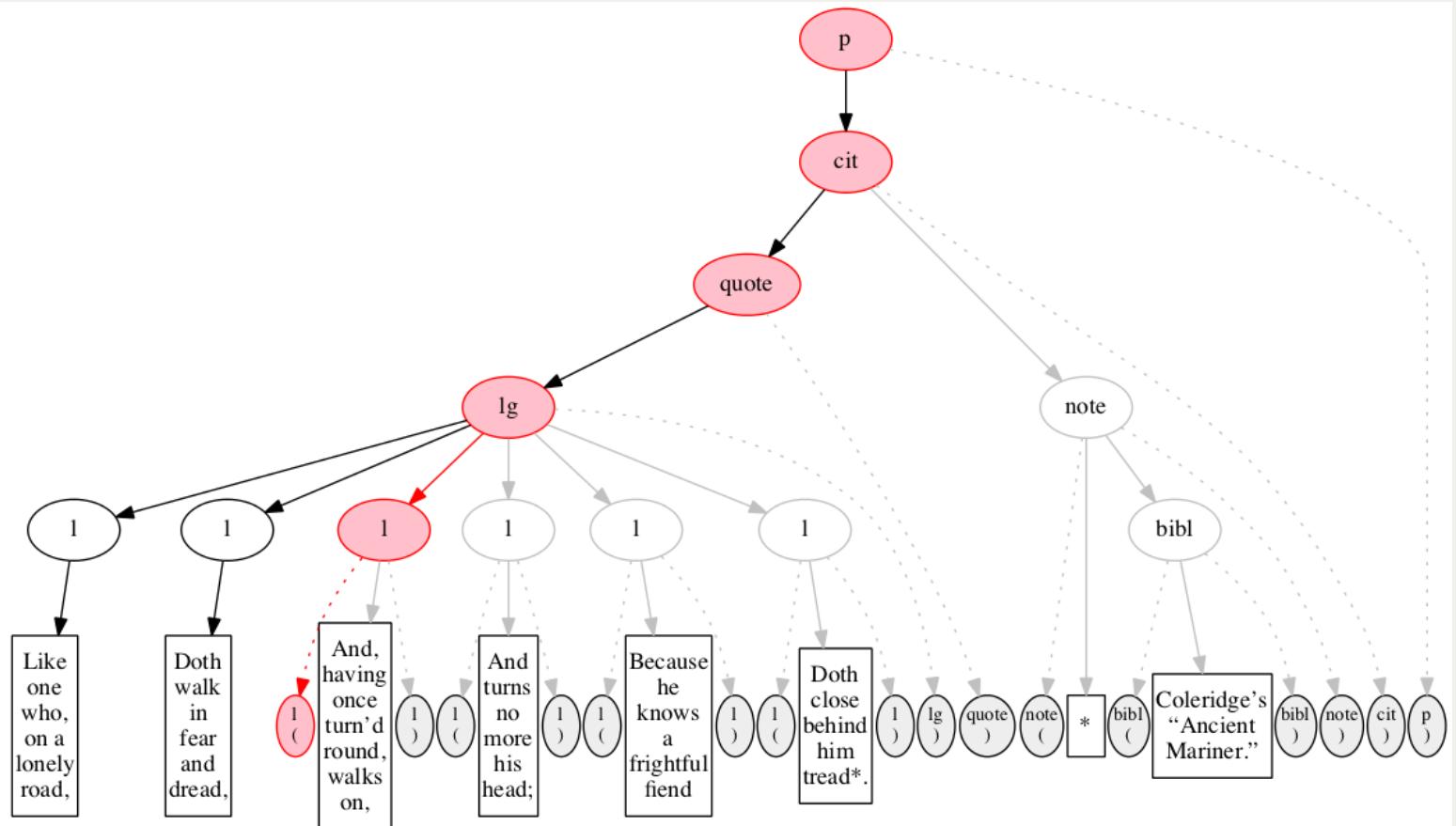
The text of the second line of verse.

Left to right



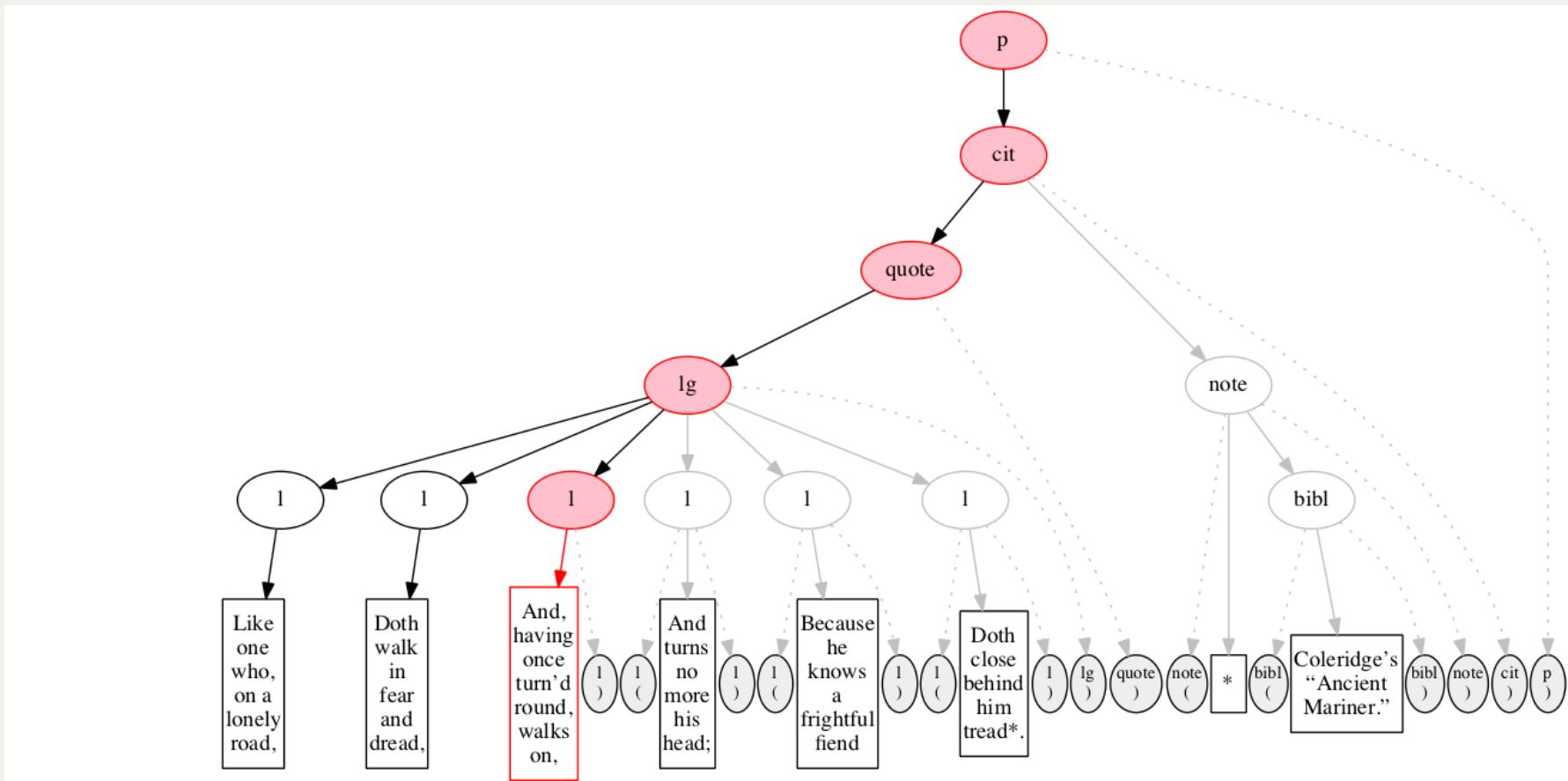
End-marker for the second line.

Left to right

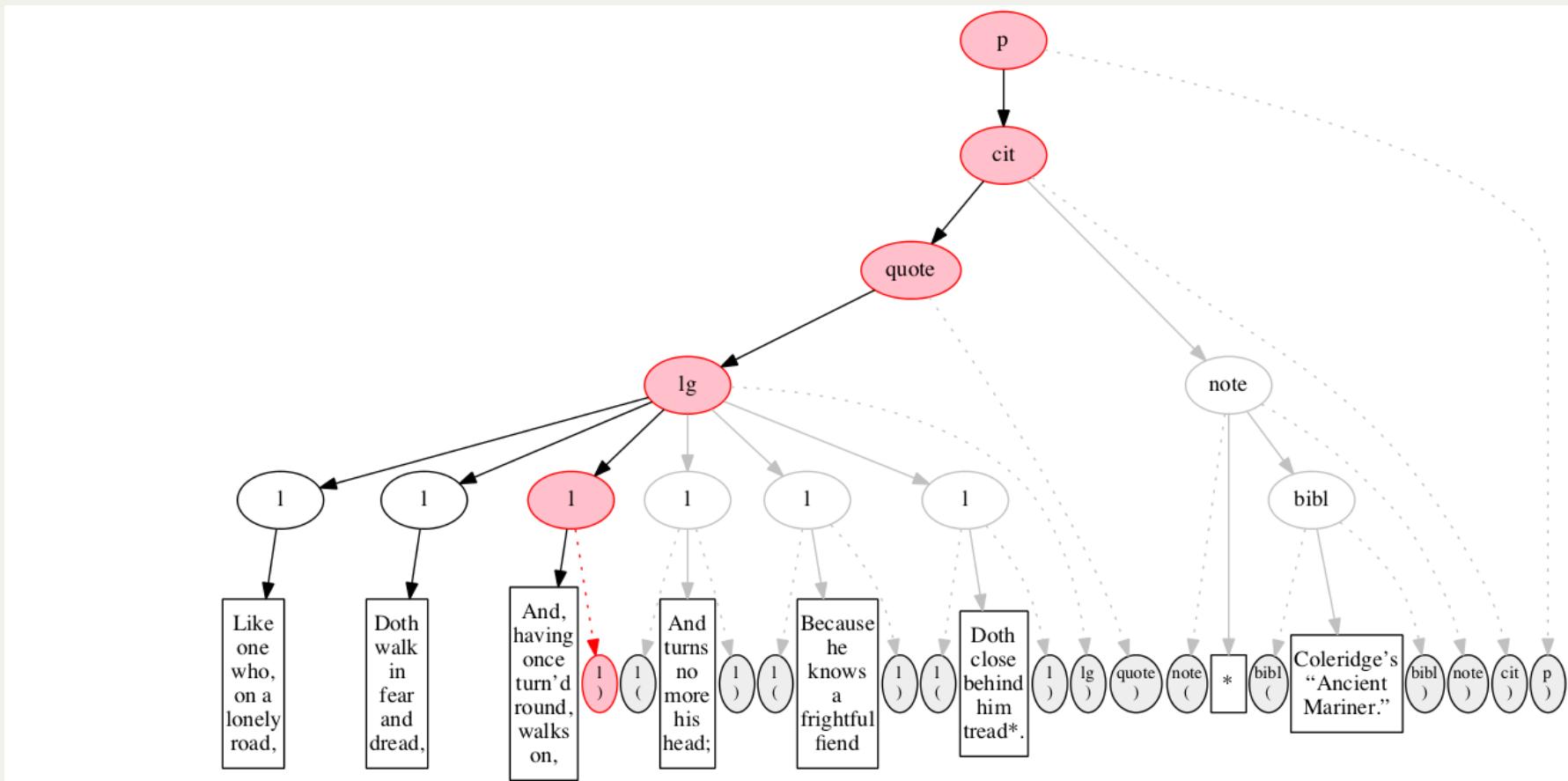


Start-marker for the third line of verse.

Left to right

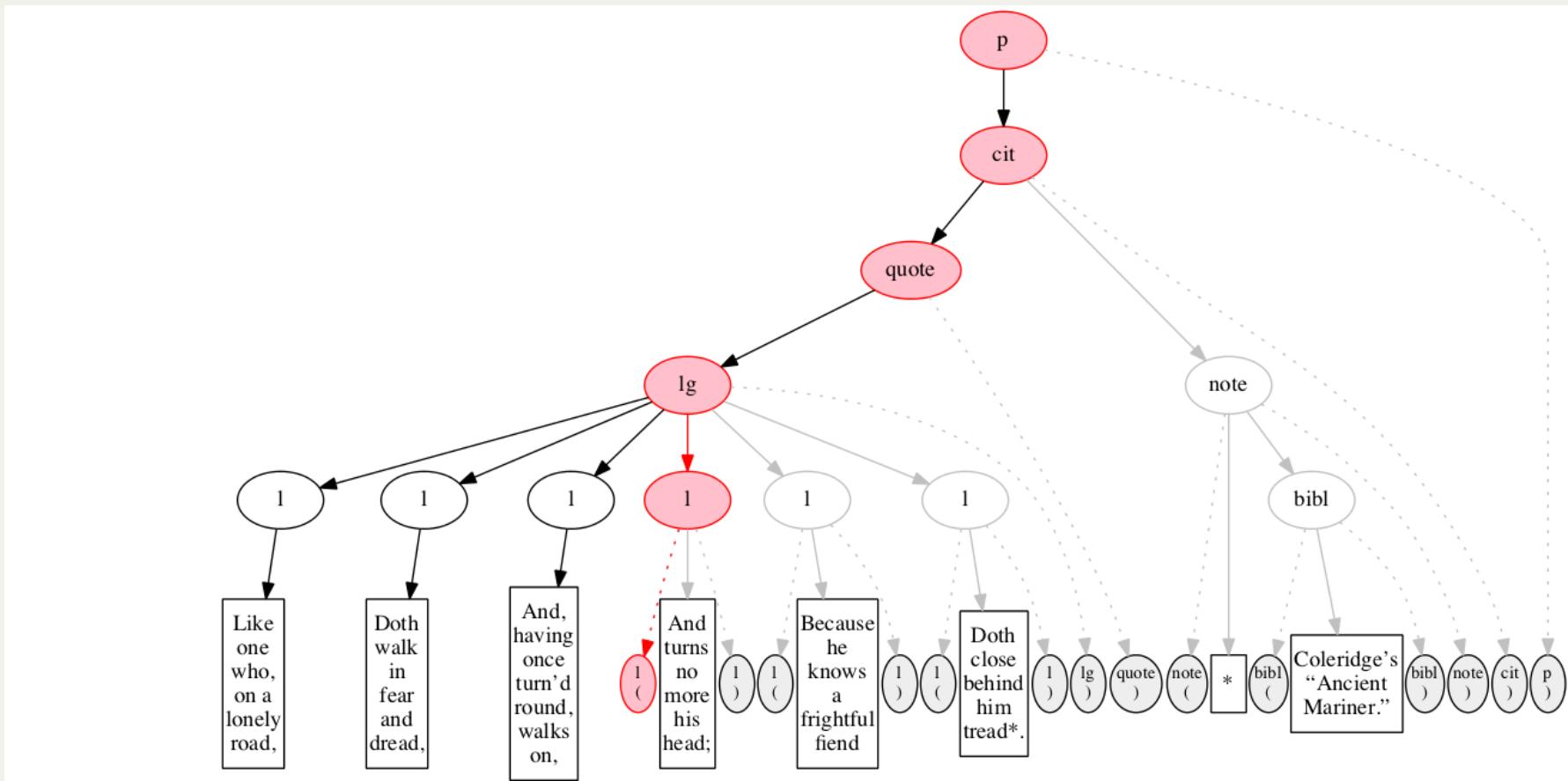


Left to right



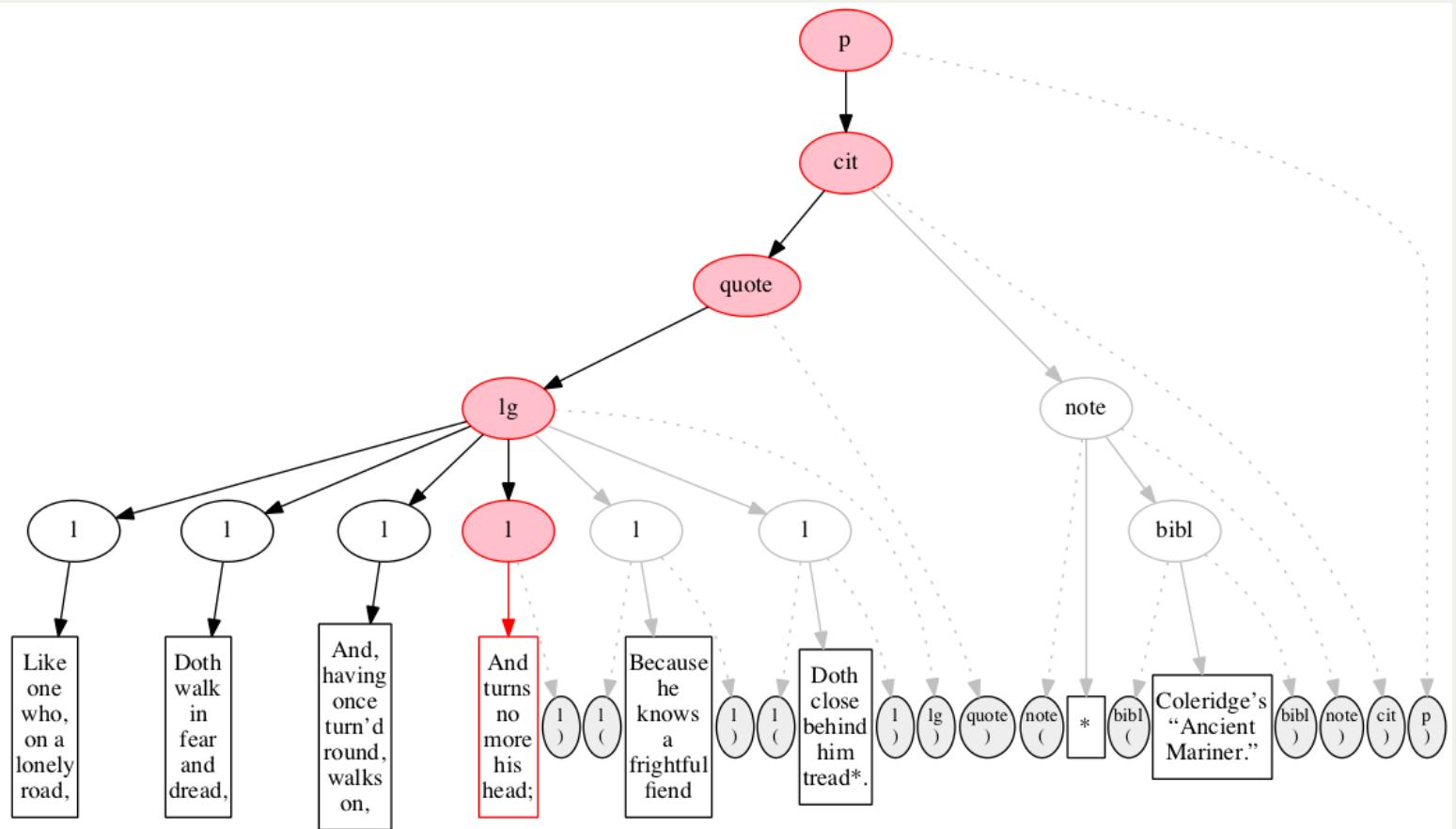
End-marker for the line.

Left to right



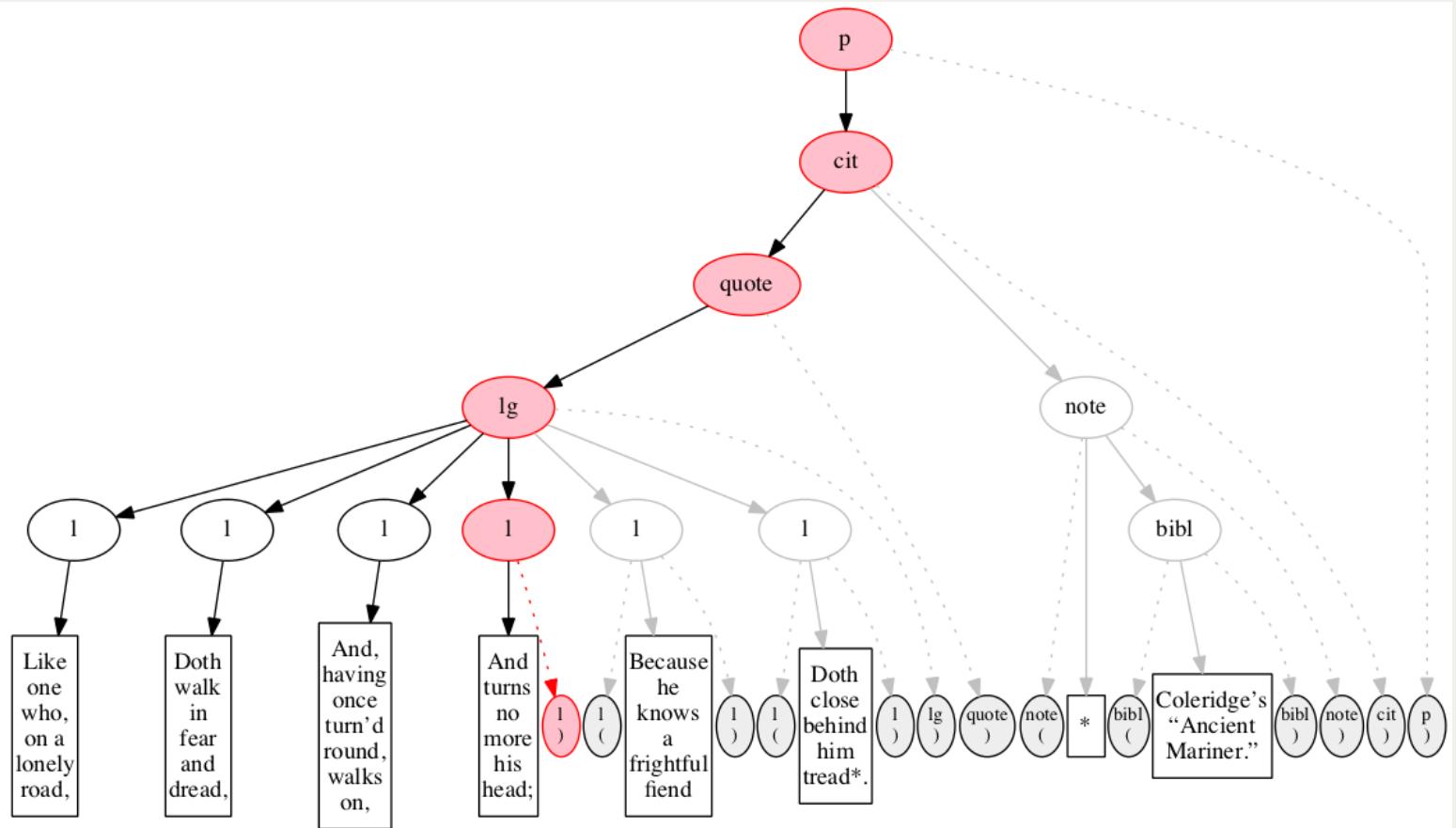
Start-marker for the fourth line of verse.

Left to right



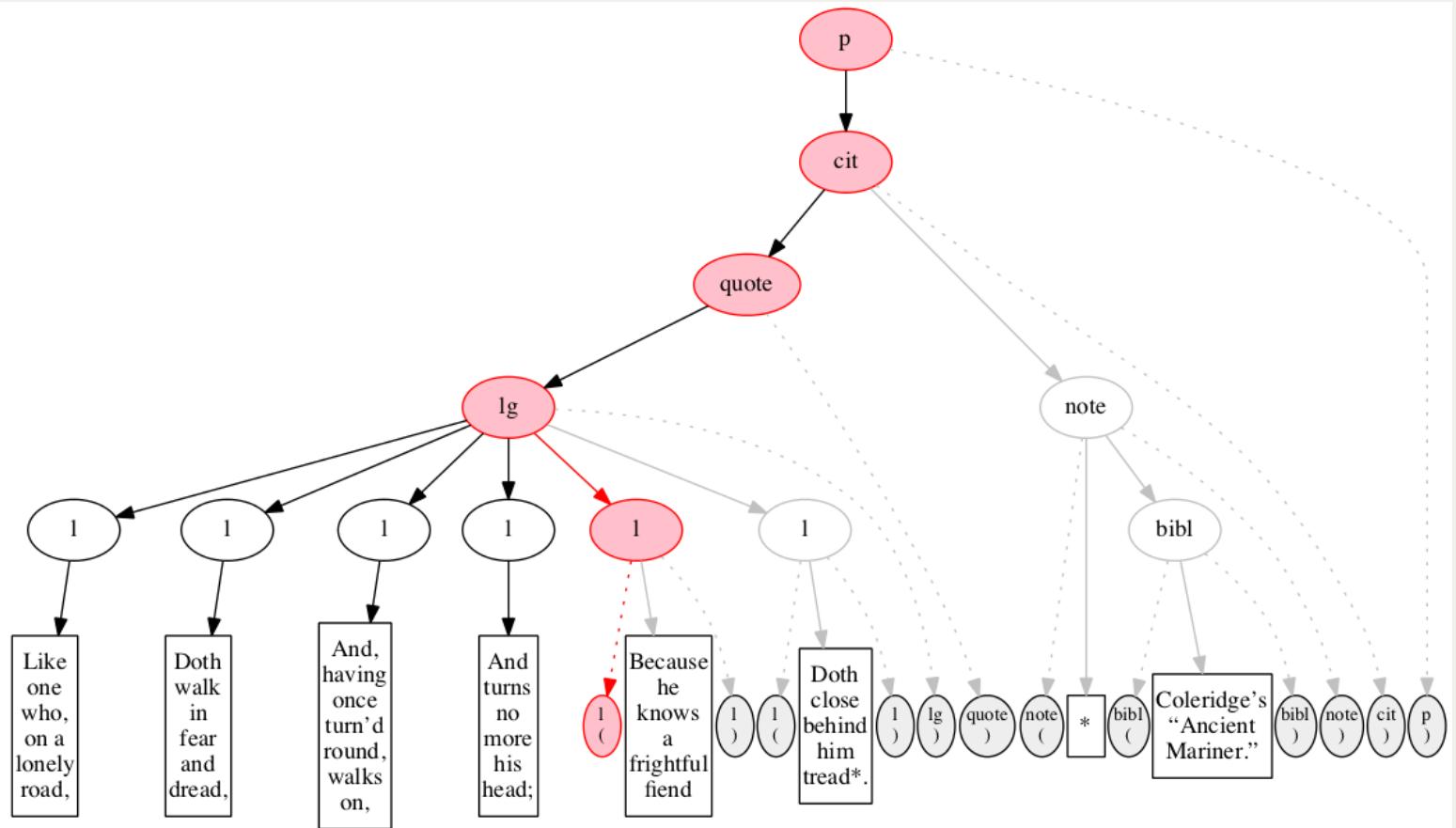
Text of the line.

Left to right



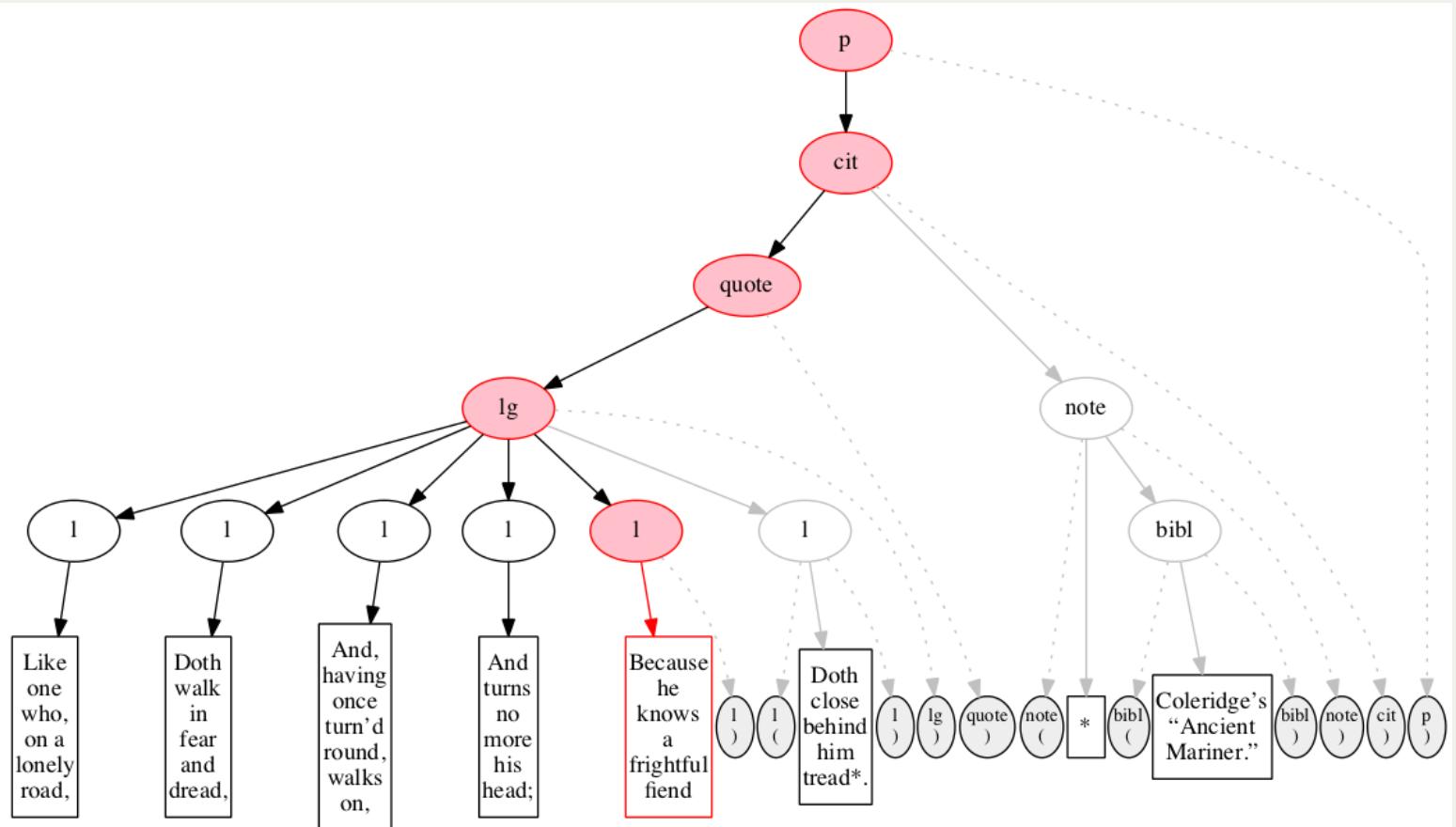
End-marker for the fourth line.

Left to right



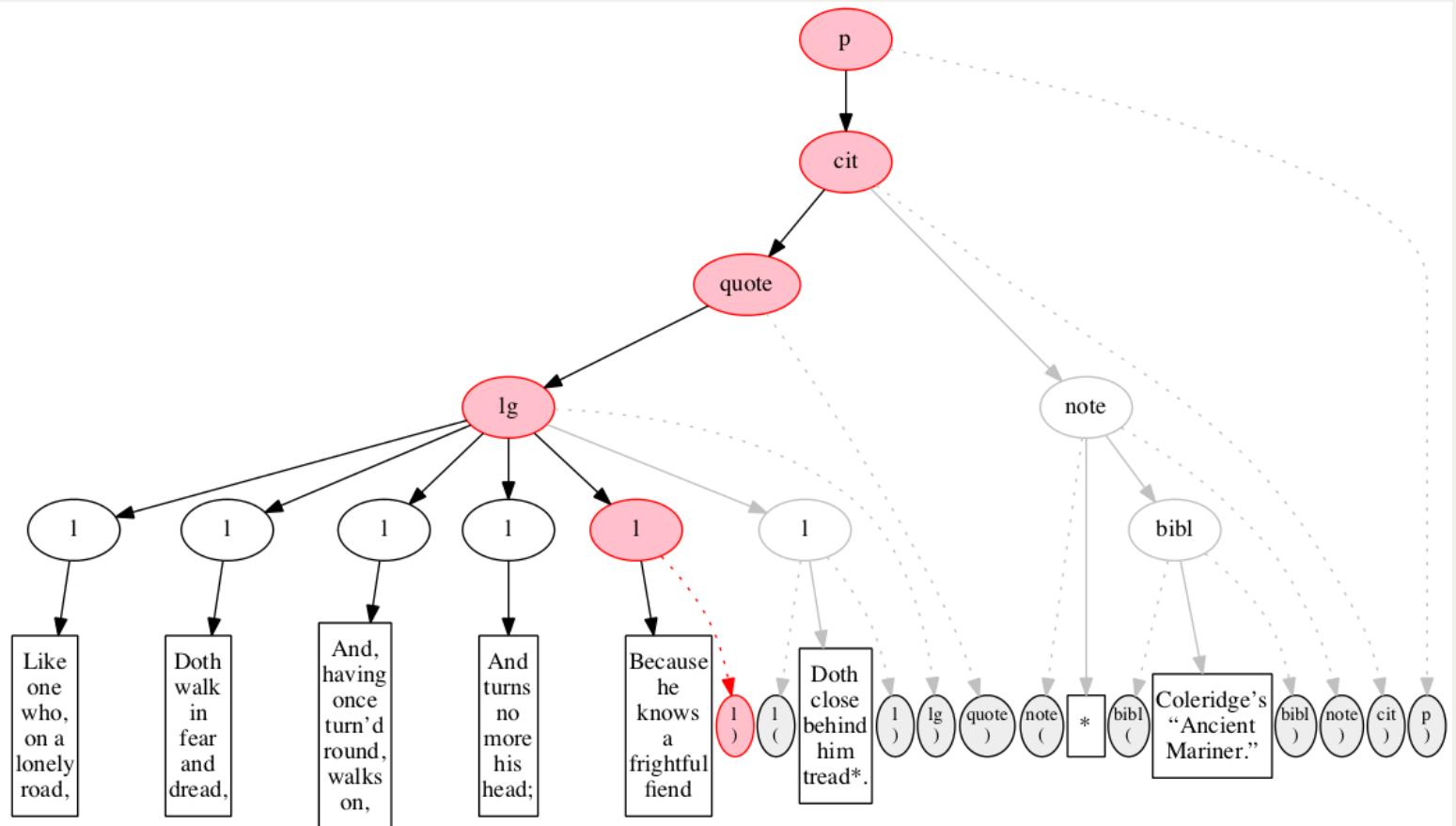
Start-marker for the fifth line of verse.

Left to right



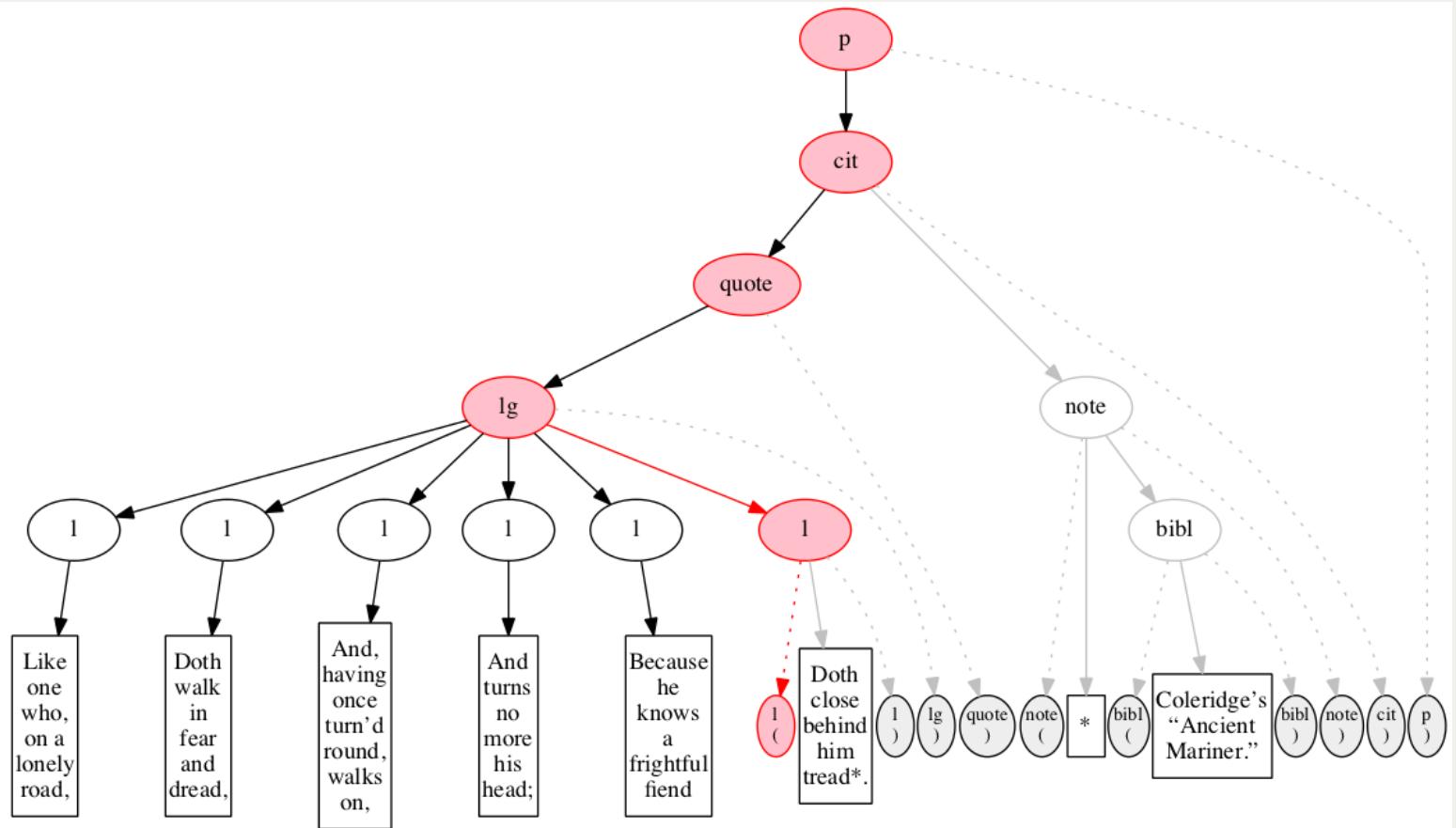
Text of the line.

Left to right



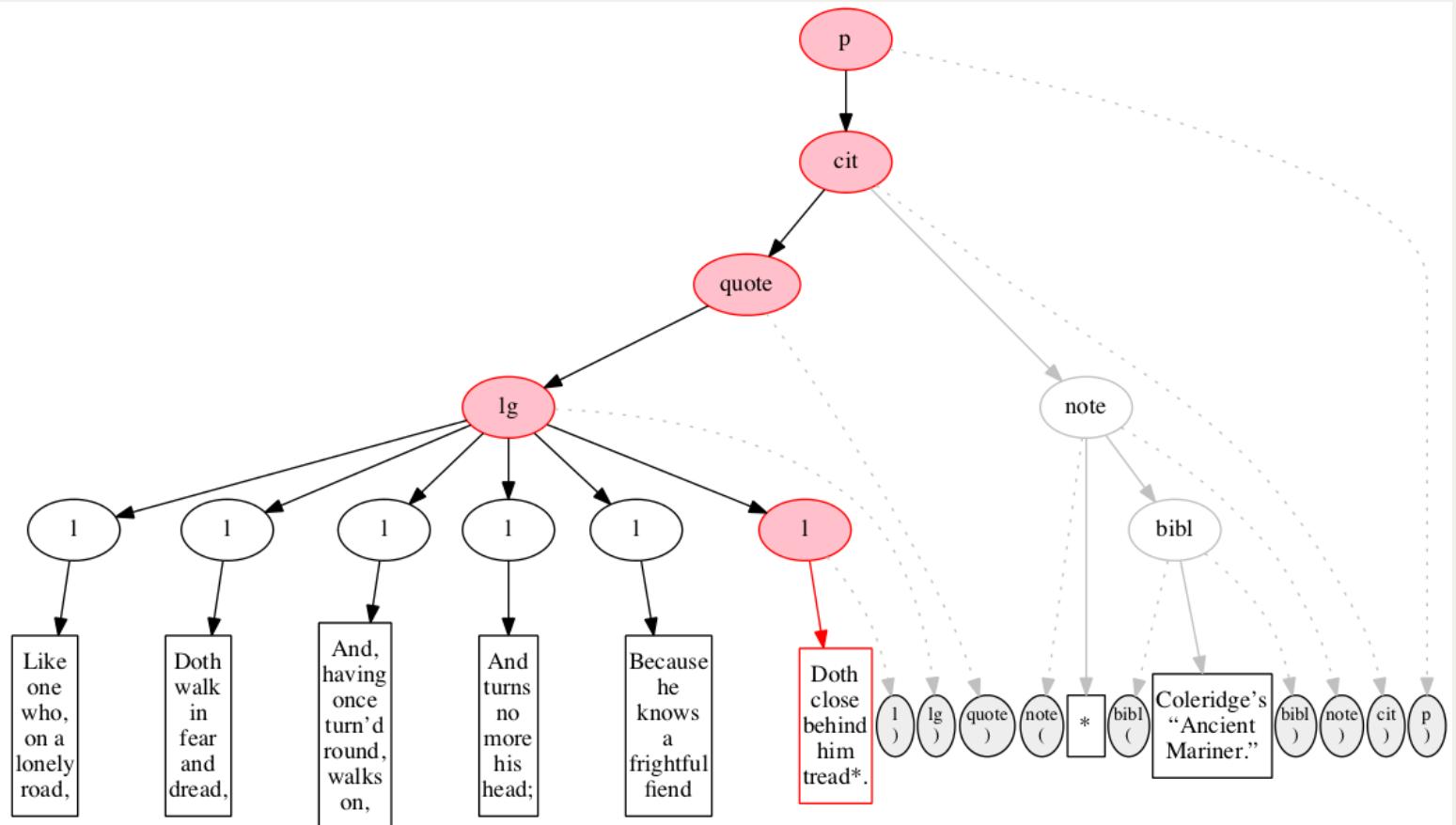
End-marker for the fifth line.

Left to right

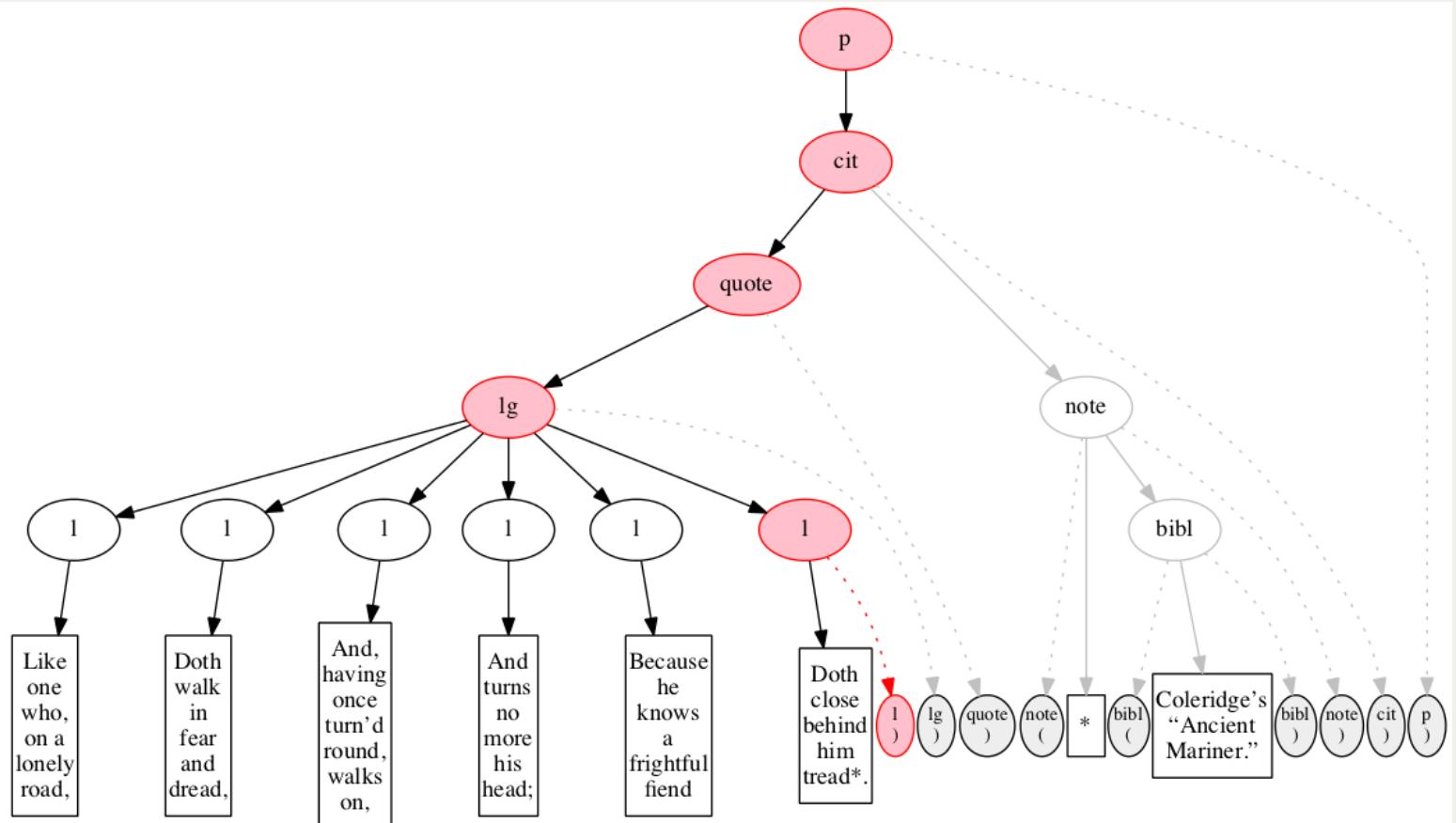


Start-marker for the last line of verse.

Left to right

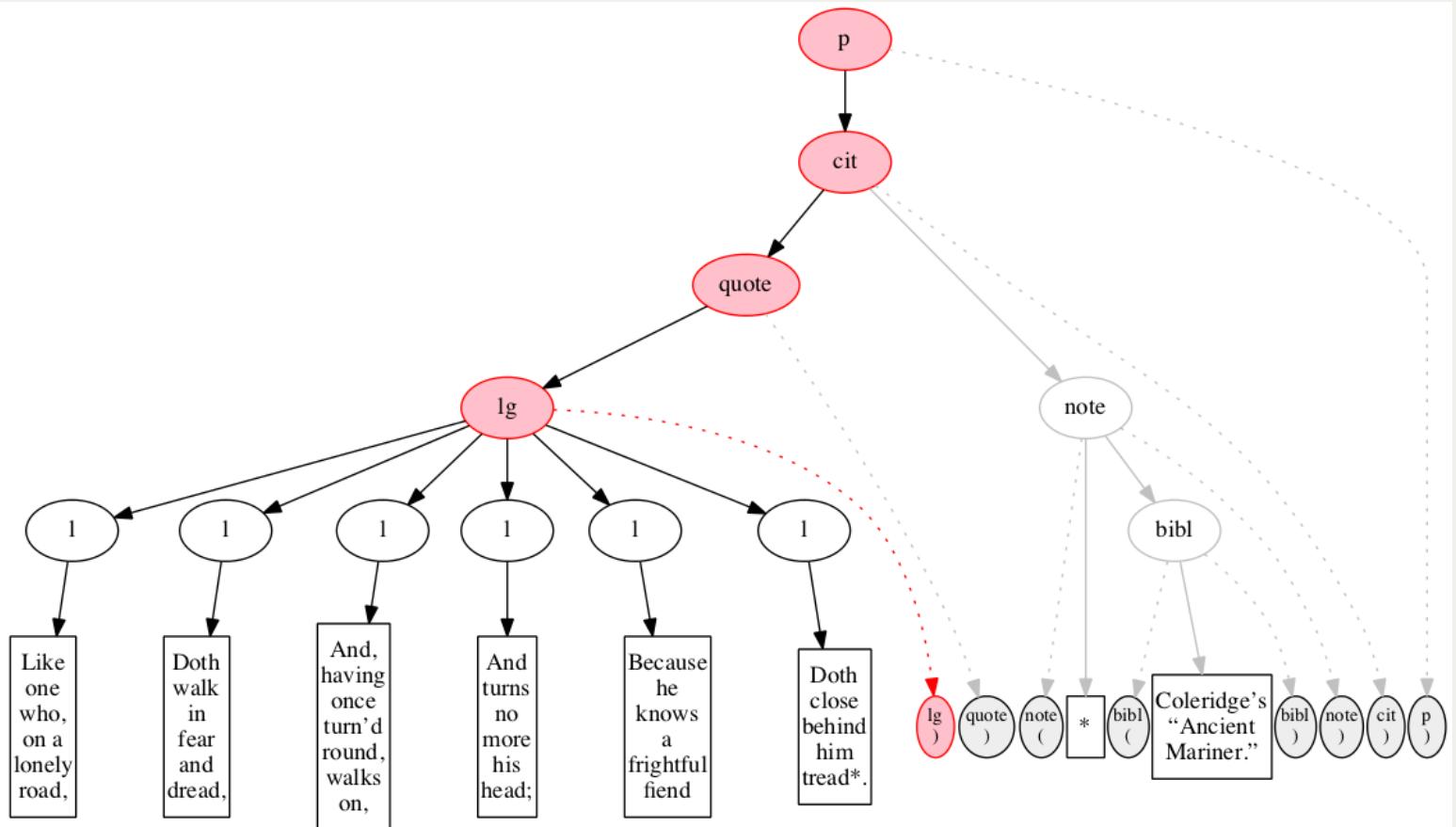


Left to right



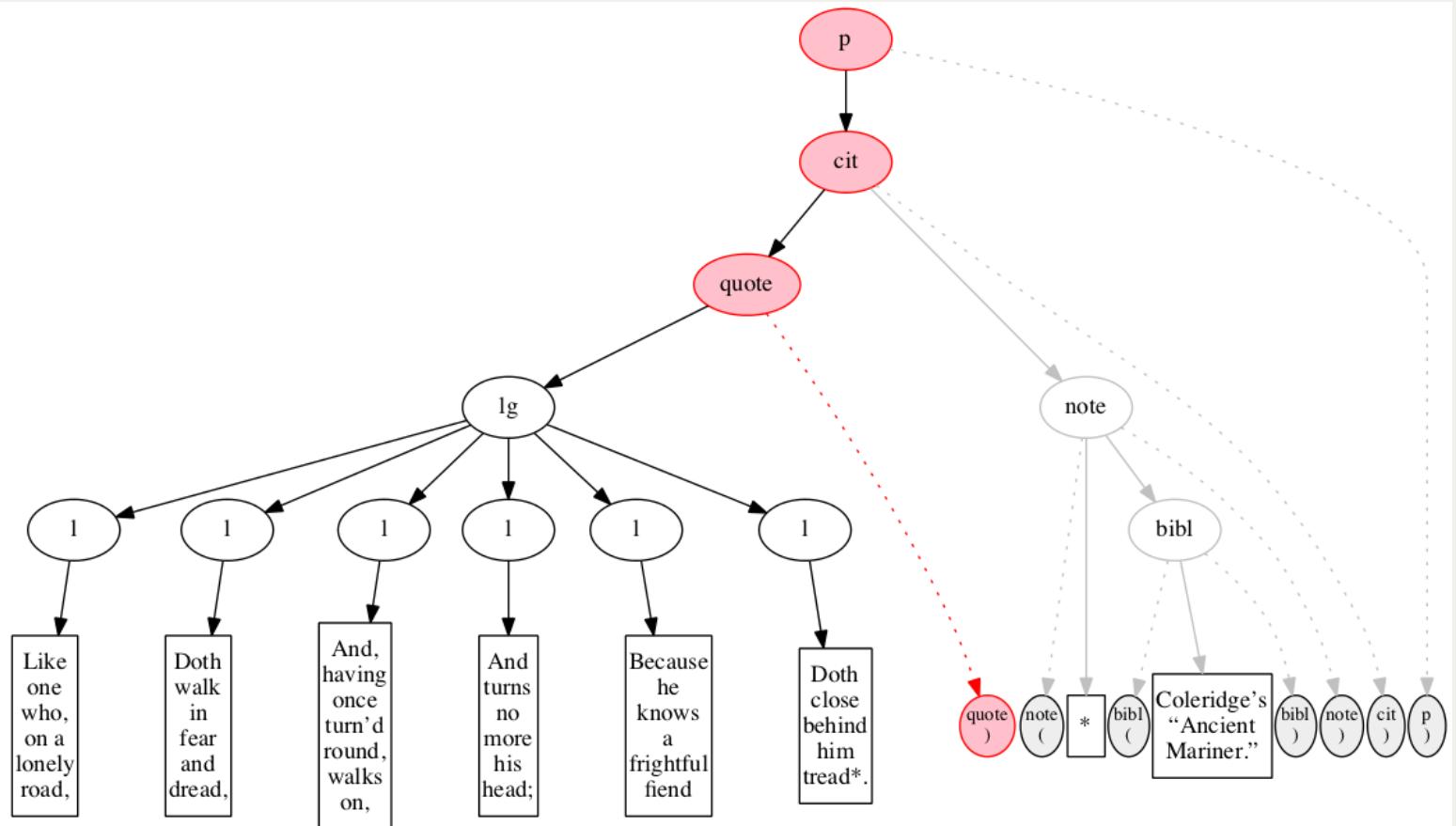
End-marker for the line.

Left to right



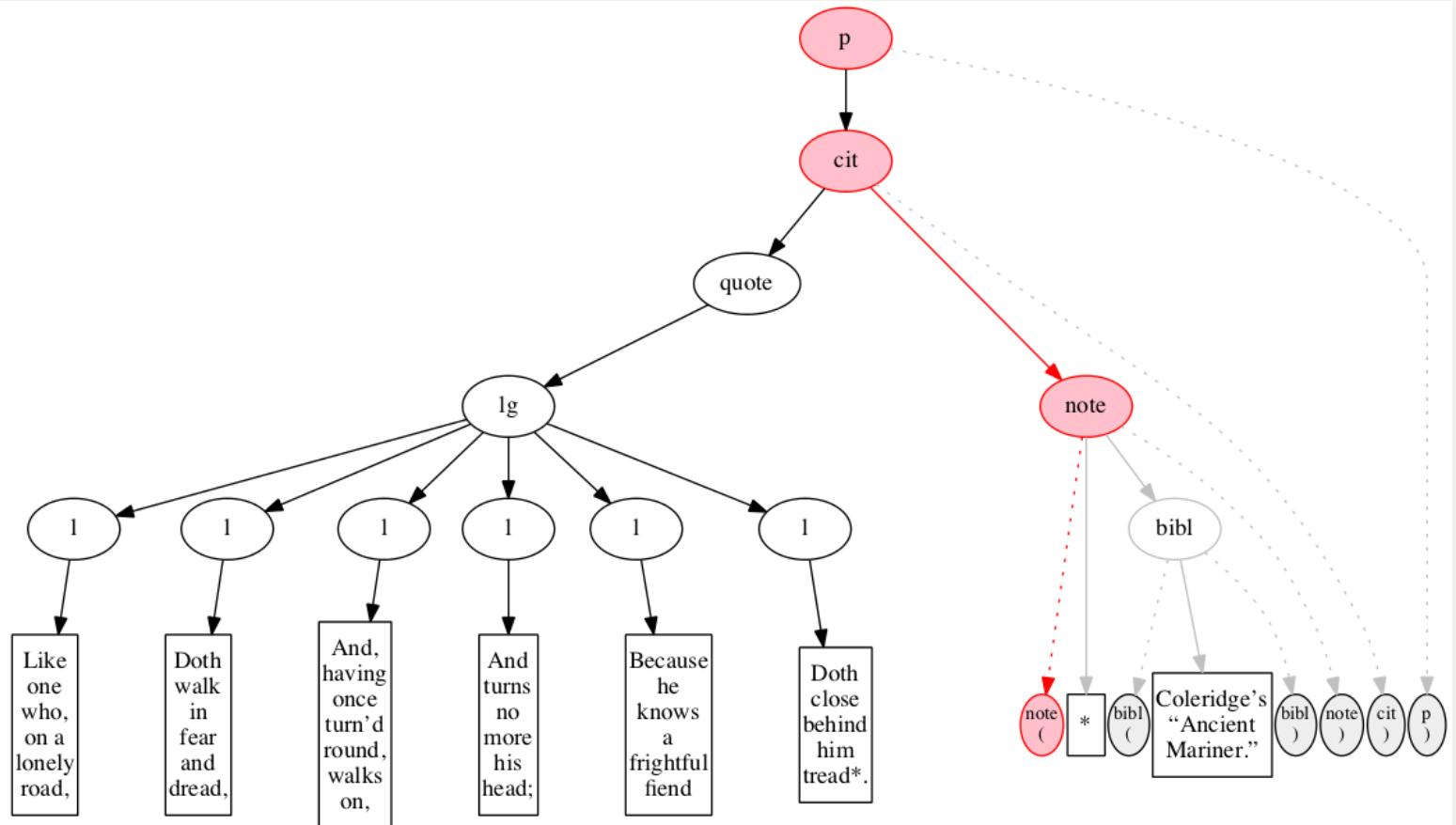
The end-marker for the line group.

Left to right



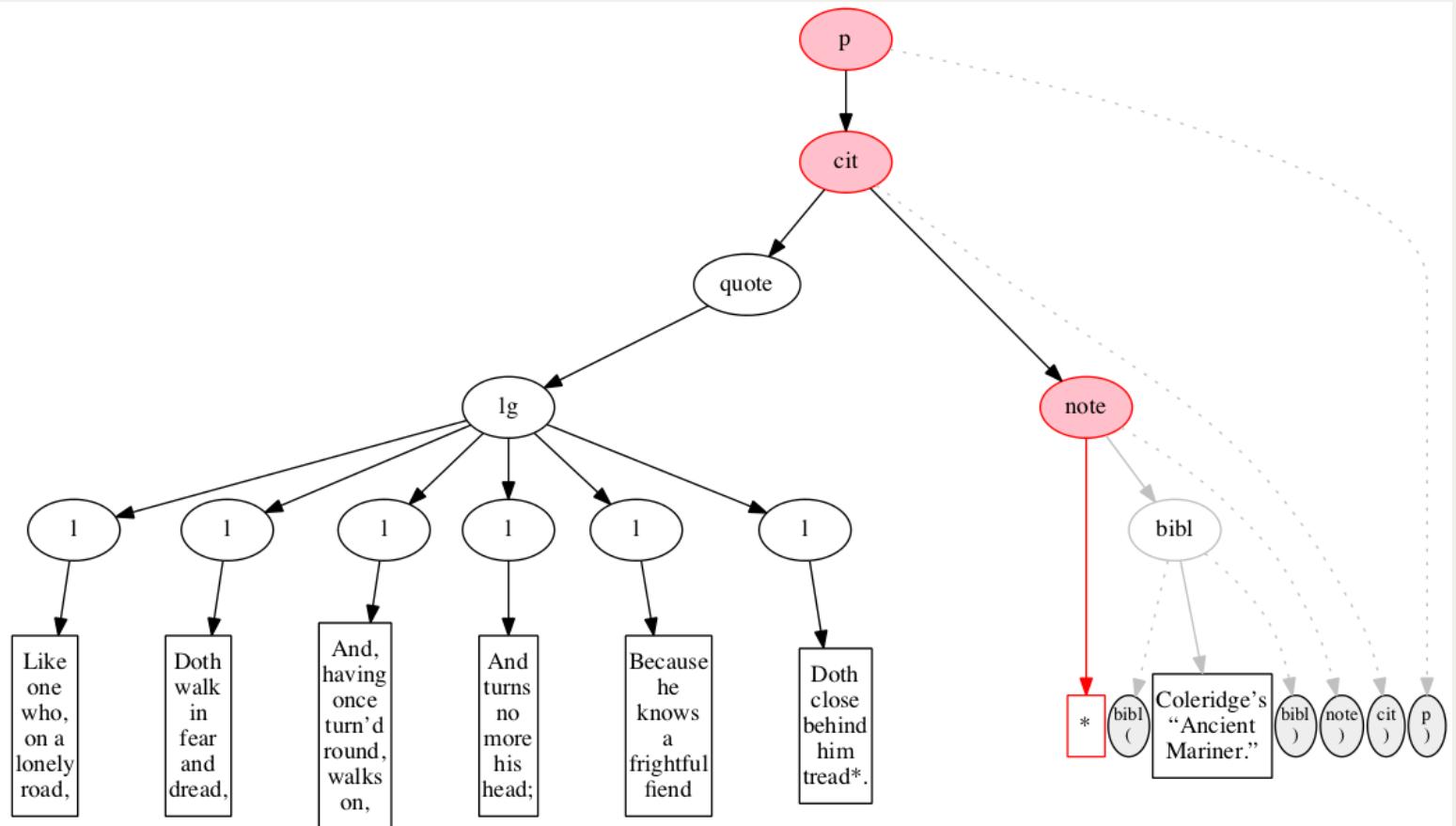
End-marker for the quotation.

Left to right



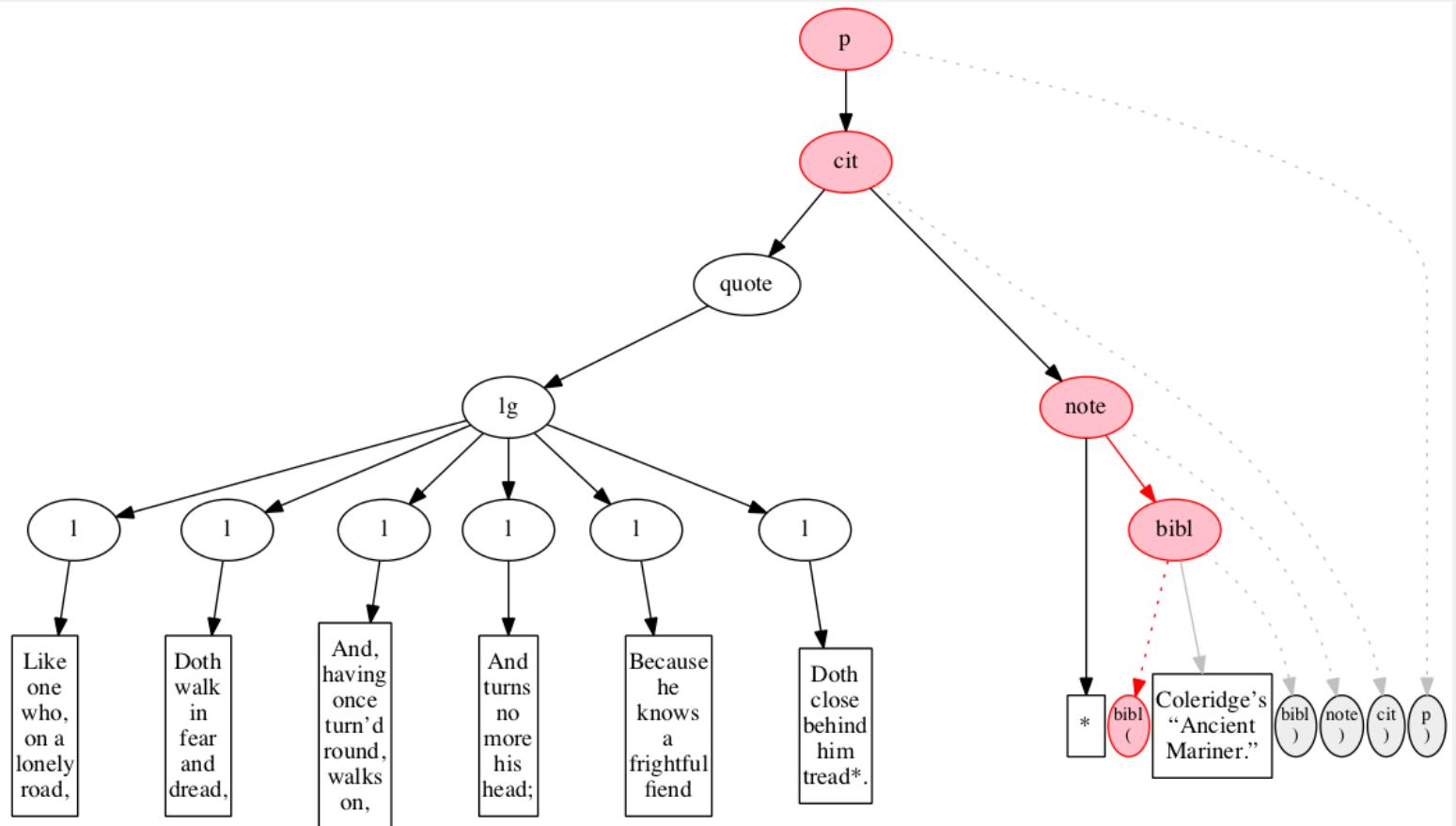
Start-marker for the note attributing the quotation.

Left to right



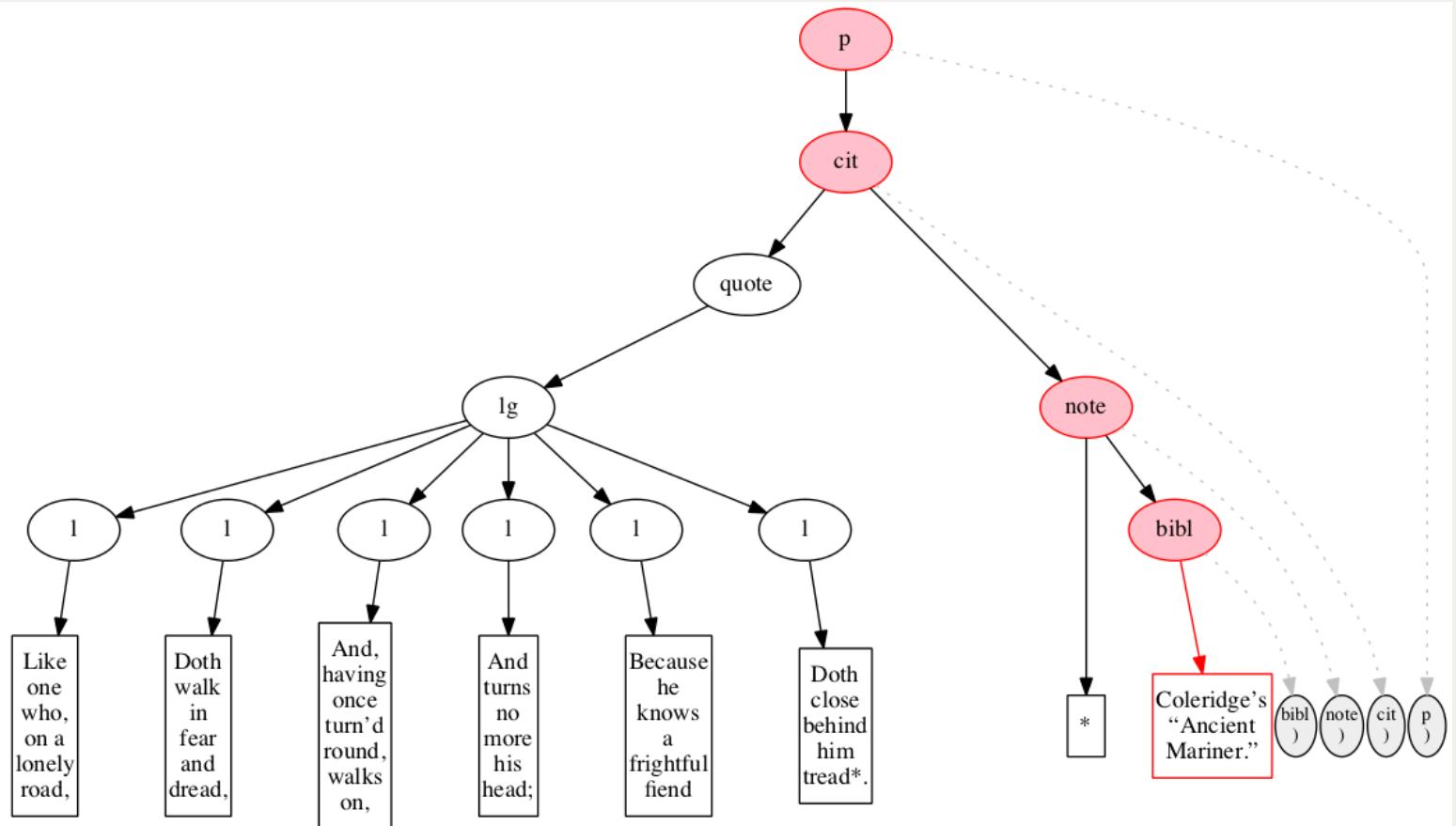
Text directly within the note.

Left to right



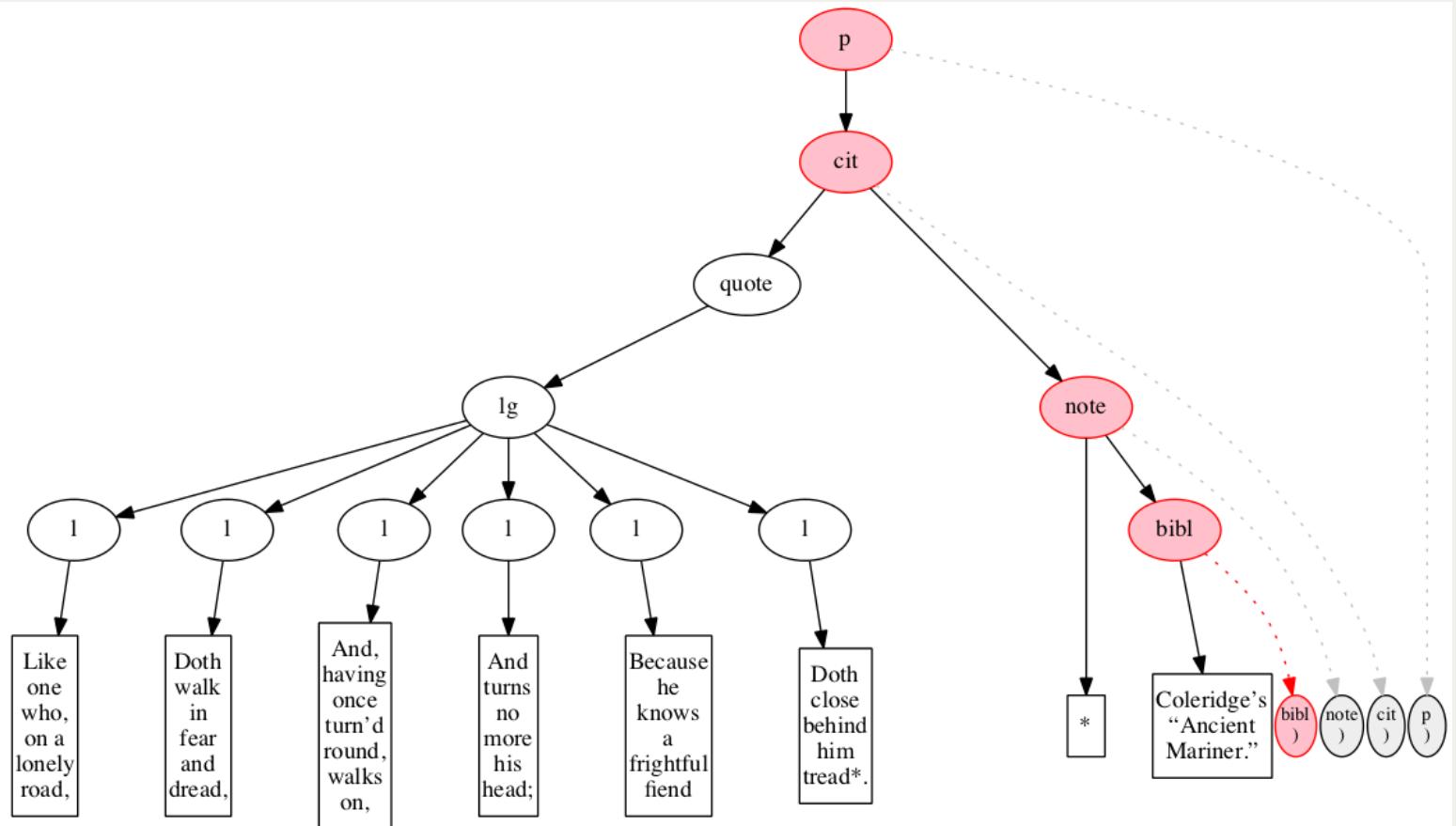
Start-marker for bibliographic reference.

Left to right



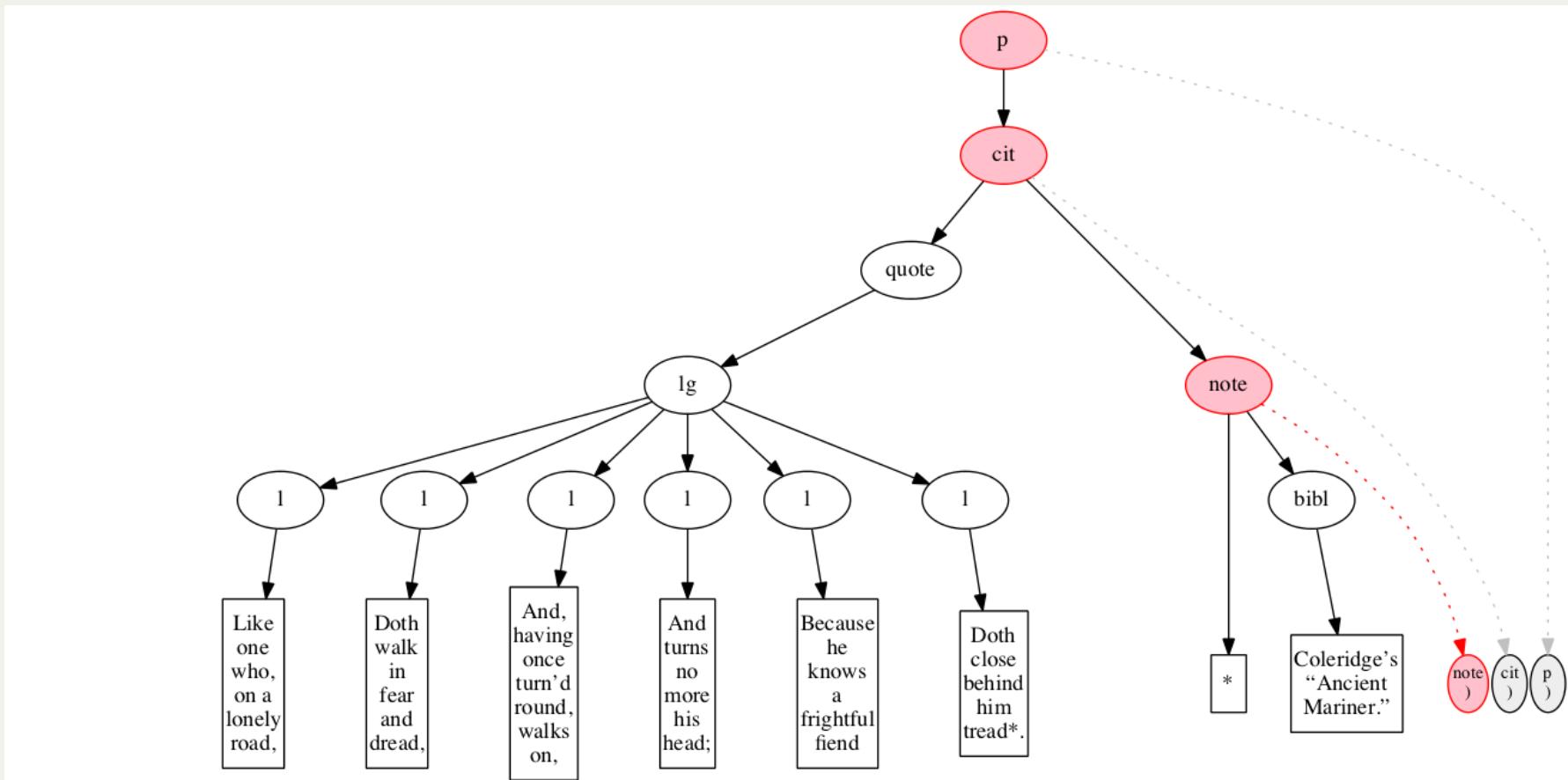
Text of the bibliographic reference.

Left to right



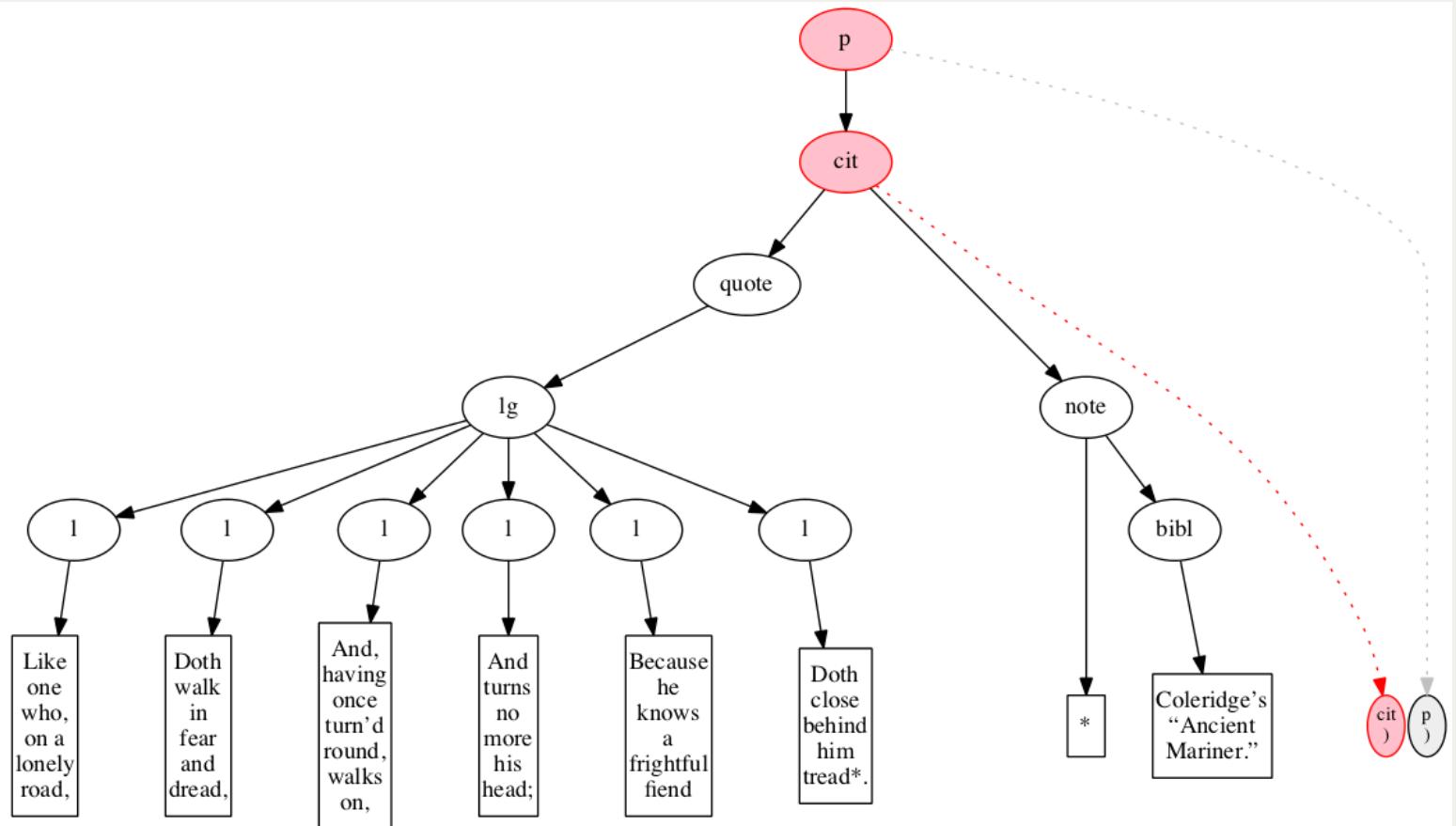
End-marker for the bibliographic reference.

Left to right



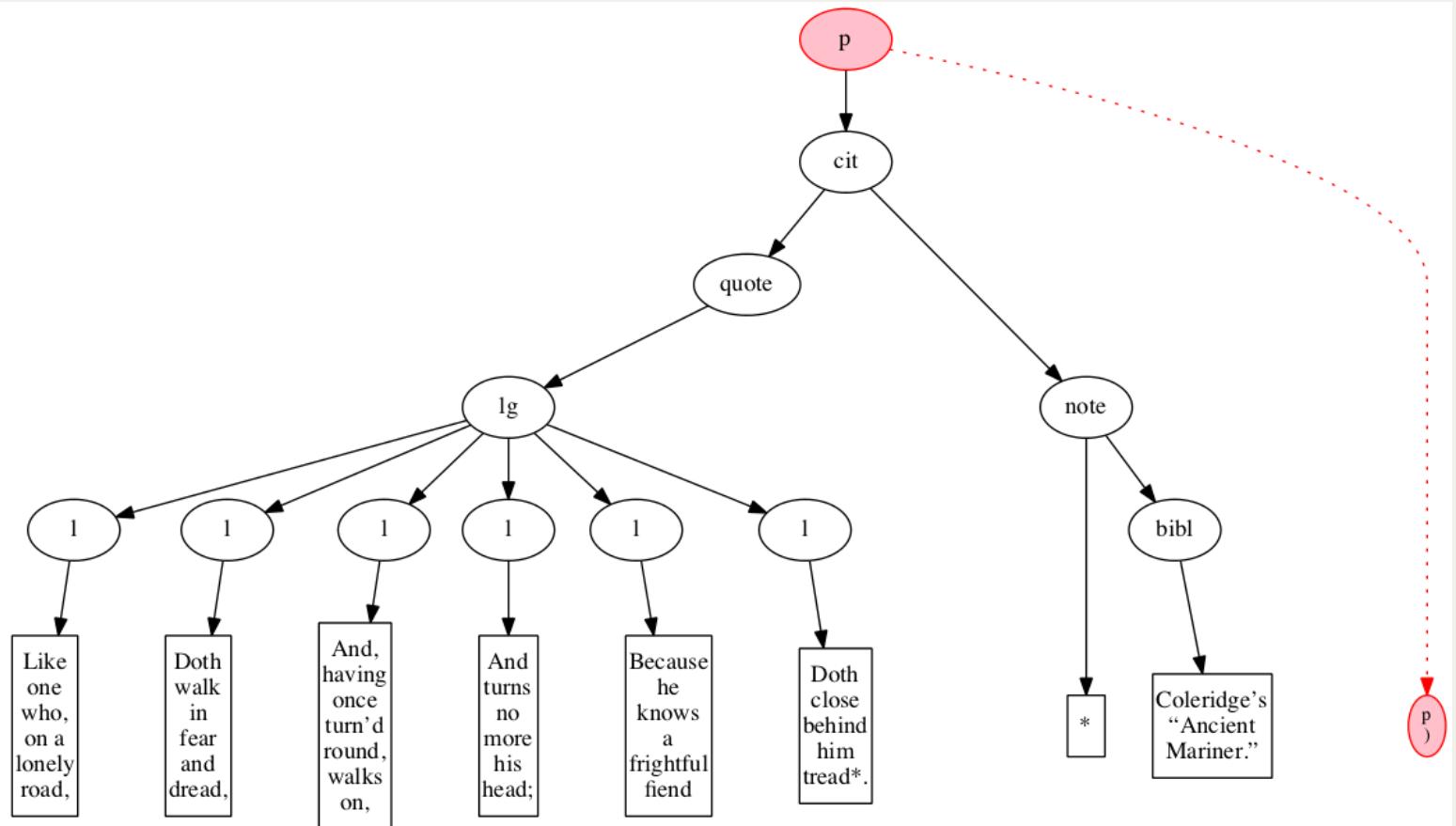
End-marker for the note.

Left to right



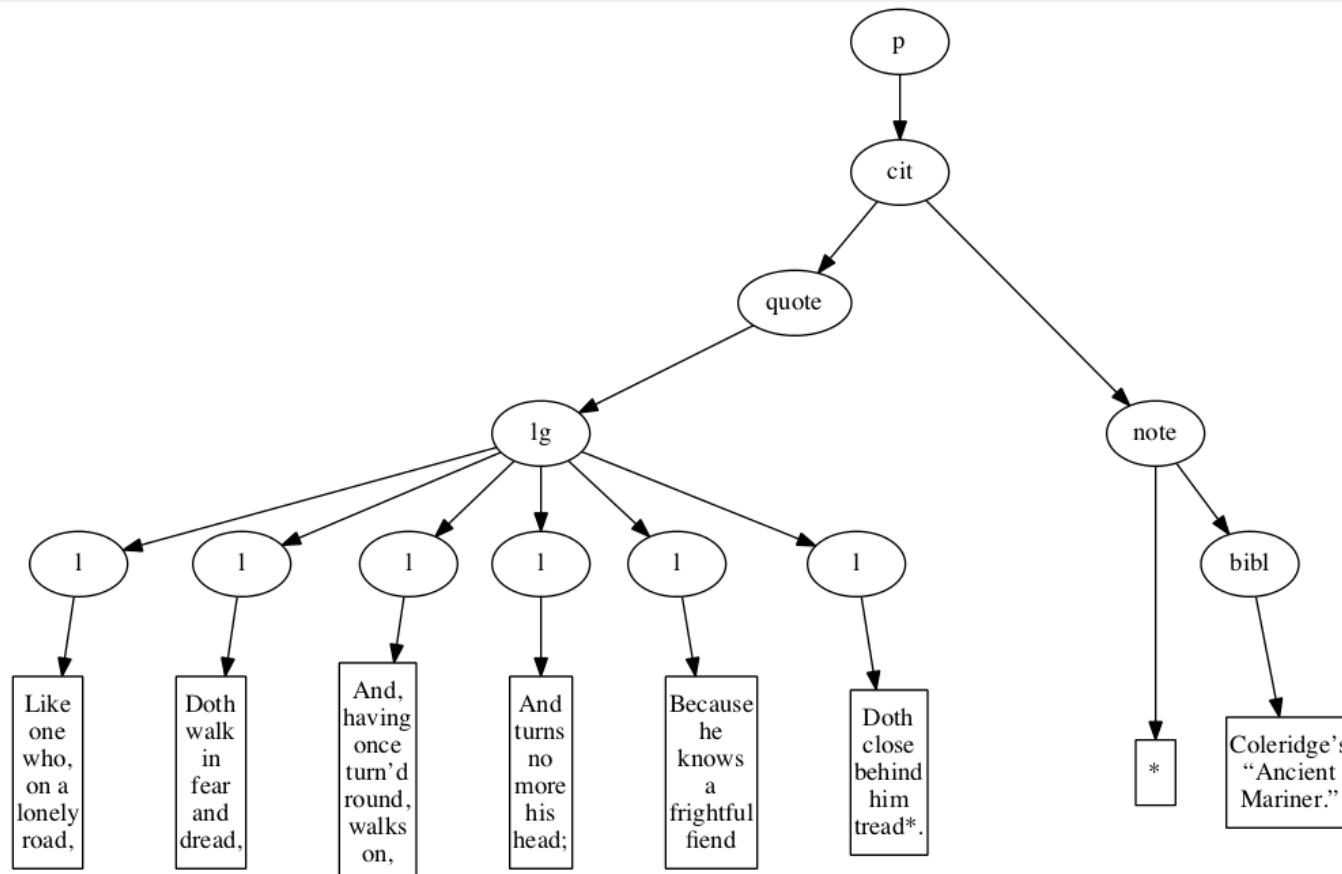
End-marker for the citation.

Left to right



End-marker for the paragraph.

Left to right



Final raised state.

Right-sibling traversal

- Single pass over input
- Useful but unusual idiom in XSLT

Right-sibling traversal

- Single pass over input
- Useful but unusual idiom in XSLT
- Call apply-templates on one node at a time

Right-sibling traversal

- Single pass over input
- Useful but unusual idiom in XSLT
- Call apply-templates on one node at a time
- On start-marker:
 - copy the element
 - process the immediate right sibling as child
 - after element closes, continue with the immediate right sibling of the end-marker

Right-sibling traversal

- Single pass over input
- Useful but unusual idiom in XSLT
- Call apply-templates on one node at a time
- On start-marker:
 - copy the element
 - process the immediate right sibling as child
 - after element closes, continue with the immediate right sibling of the end-marker
- On end-marker: stop

Right-sibling traversal

- Single pass over input
- Useful but unusual idiom in XSLT
- Call apply-templates on one node at a time
- On start-marker:
 - copy the element
 - process the immediate right sibling as child
 - after element closes, continue with the immediate right sibling of the end-marker
- On end-marker: stop
- On anything else: copy to output

Right-sibling traversal code

On start-marker

```
<xsl:template match="*[th:is-start-marker(.)]" mode="raising">

    <!--* 1: handle this element *-->
    <xsl:copy copy-namespaces="no">
        <xsl:copy-of select="@* except @th:*" />
        <xsl:apply-templates select="following-sibling::node()[1]"
            mode="raising">
            </xsl:apply-templates>
        </xsl:copy>

    <!--* 2: continue after this element *-->
    <xsl:apply-templates select="th:matching-end-marker(.)
        /following-sibling::node()[1]"
        mode="raising">
        </xsl:apply-templates>
    </xsl:template>
```

Right-sibling traversal code

On end-marker

```
<xsl:template match="*[th:is-end-marker(.)]" mode="raising">

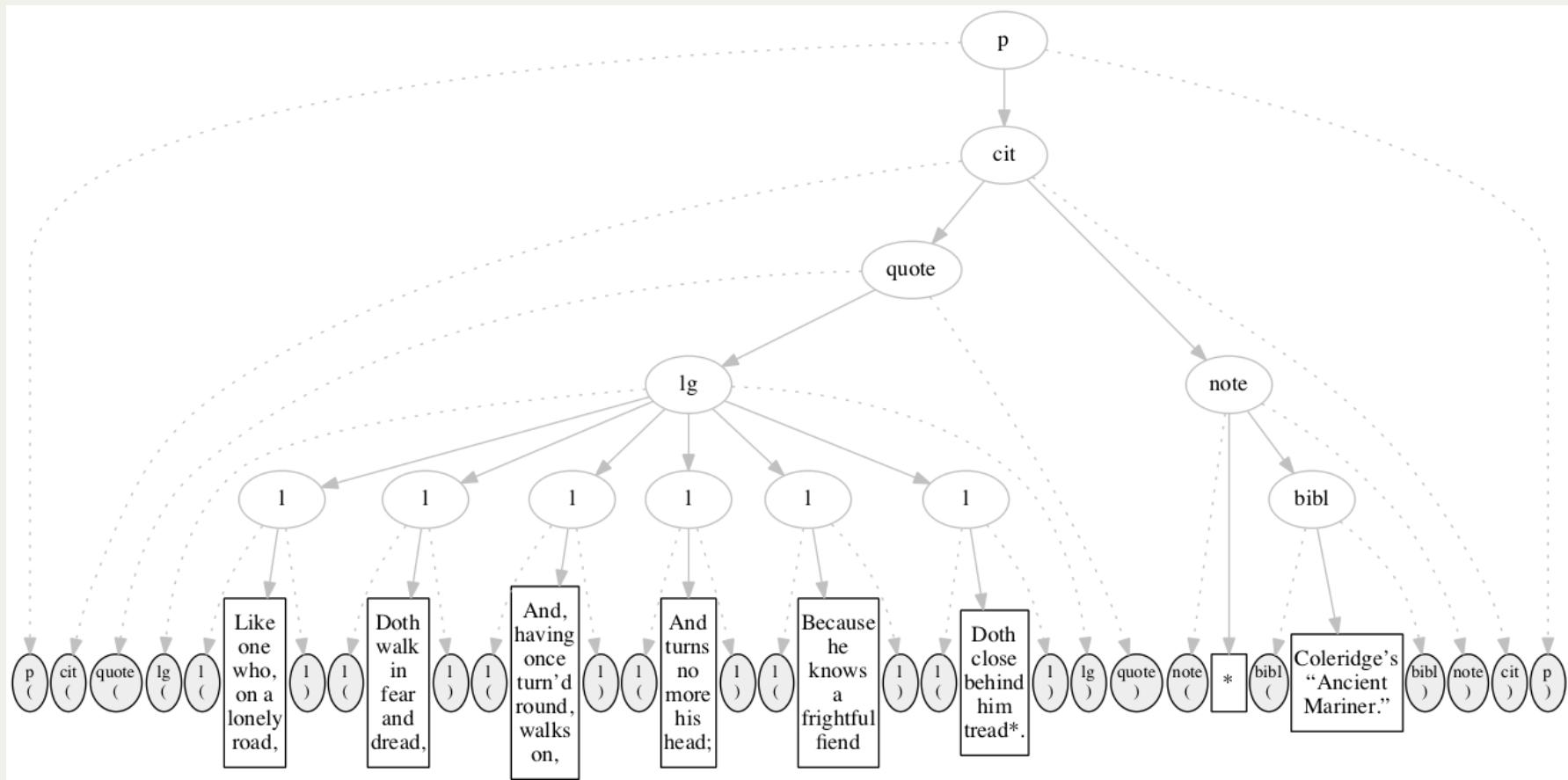
    <!--* no action necessary *-->
    <!--* we do NOT recur to our right. We leave it to our parent
        to do that. *-->

</xsl:template>
```

On other nodes

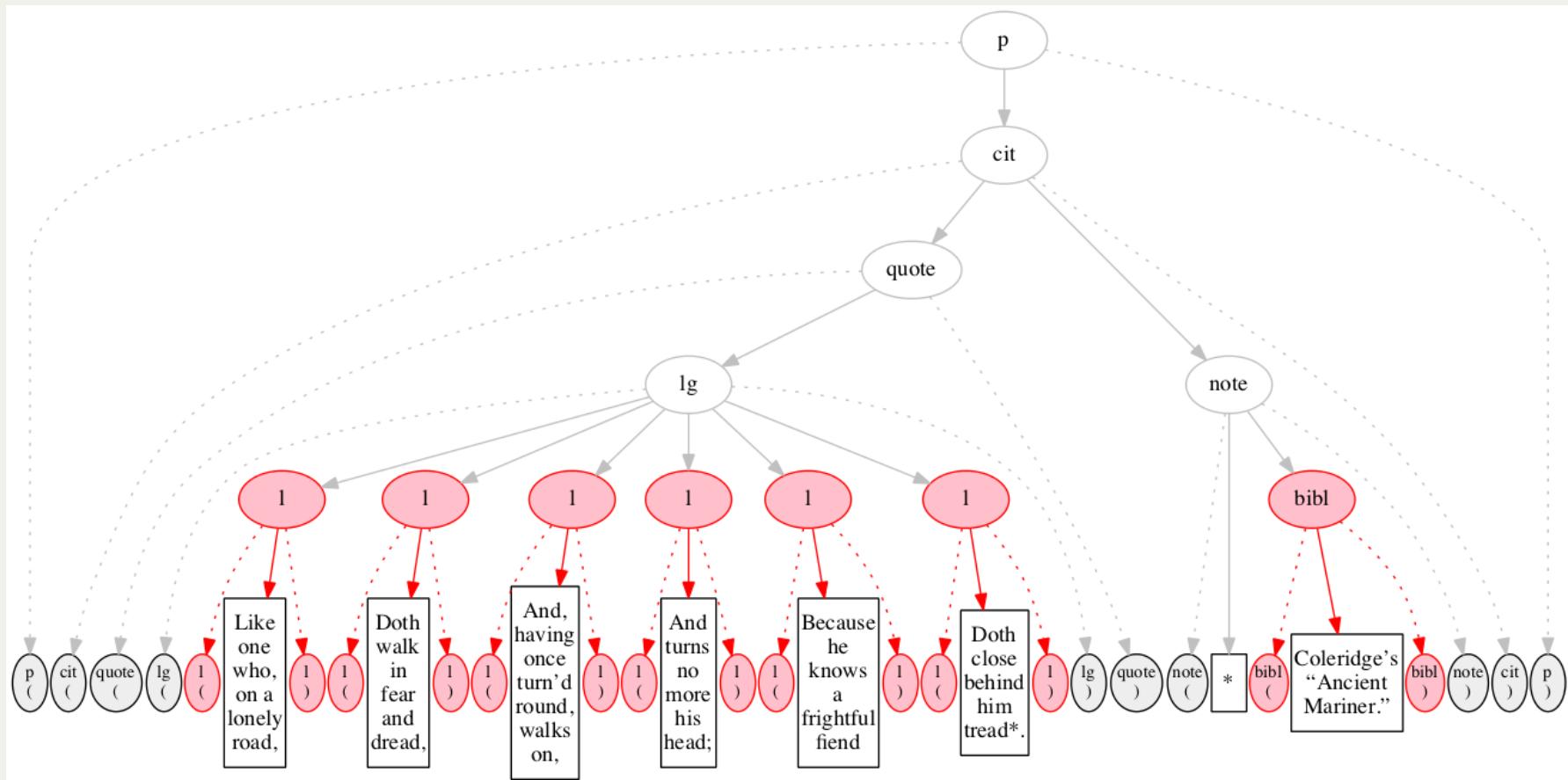
```
<xsl:template match="text() | comment() | processing-instruction()"
    mode="raising">
    <xsl:copy/>
    <xsl:apply-templates select="following-sibling::node()[1]"
        mode="raising"/>
</xsl:template>
```

Inside out



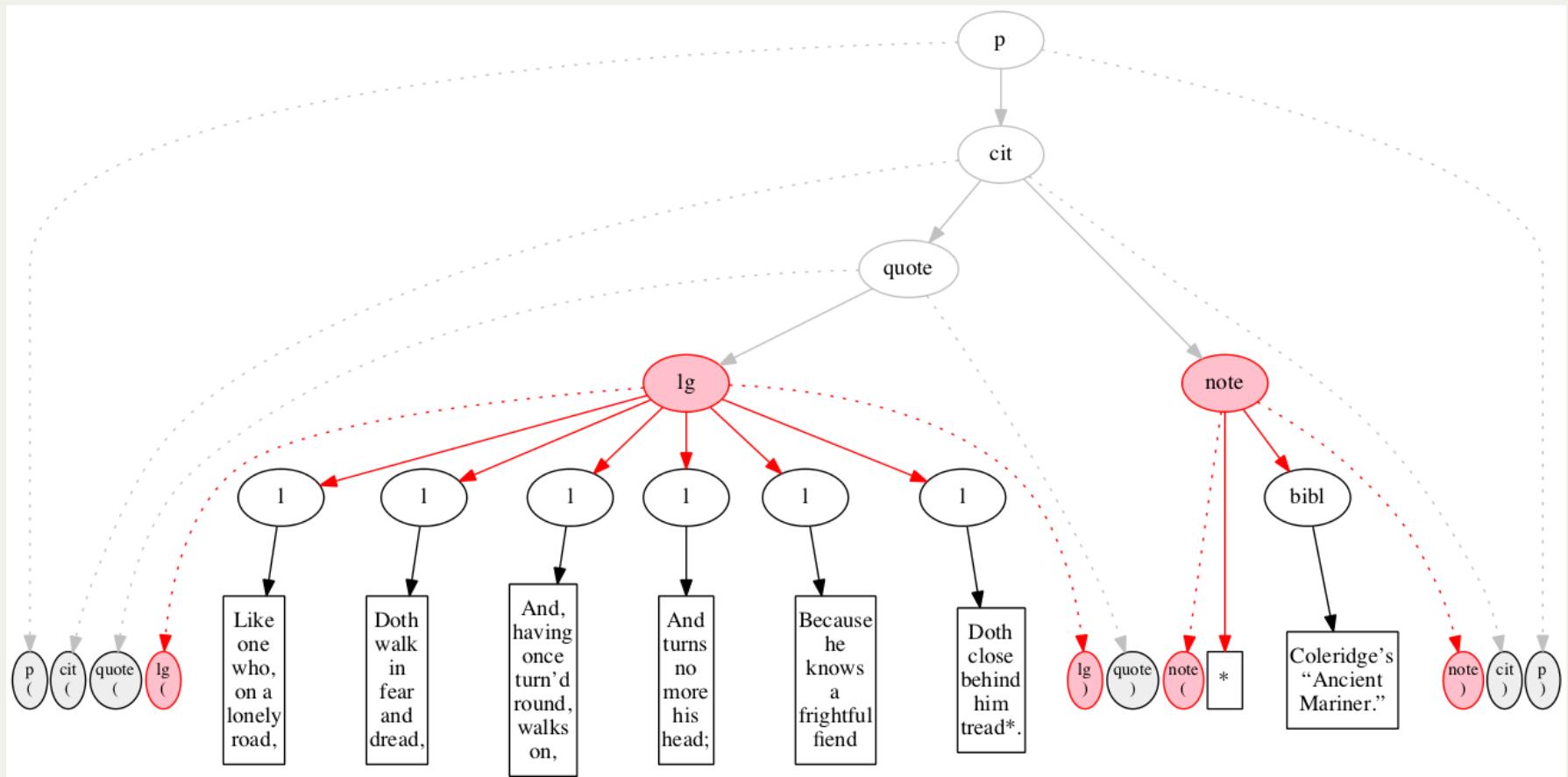
At the outset, we have a flat sequence of nodes.

Inside out



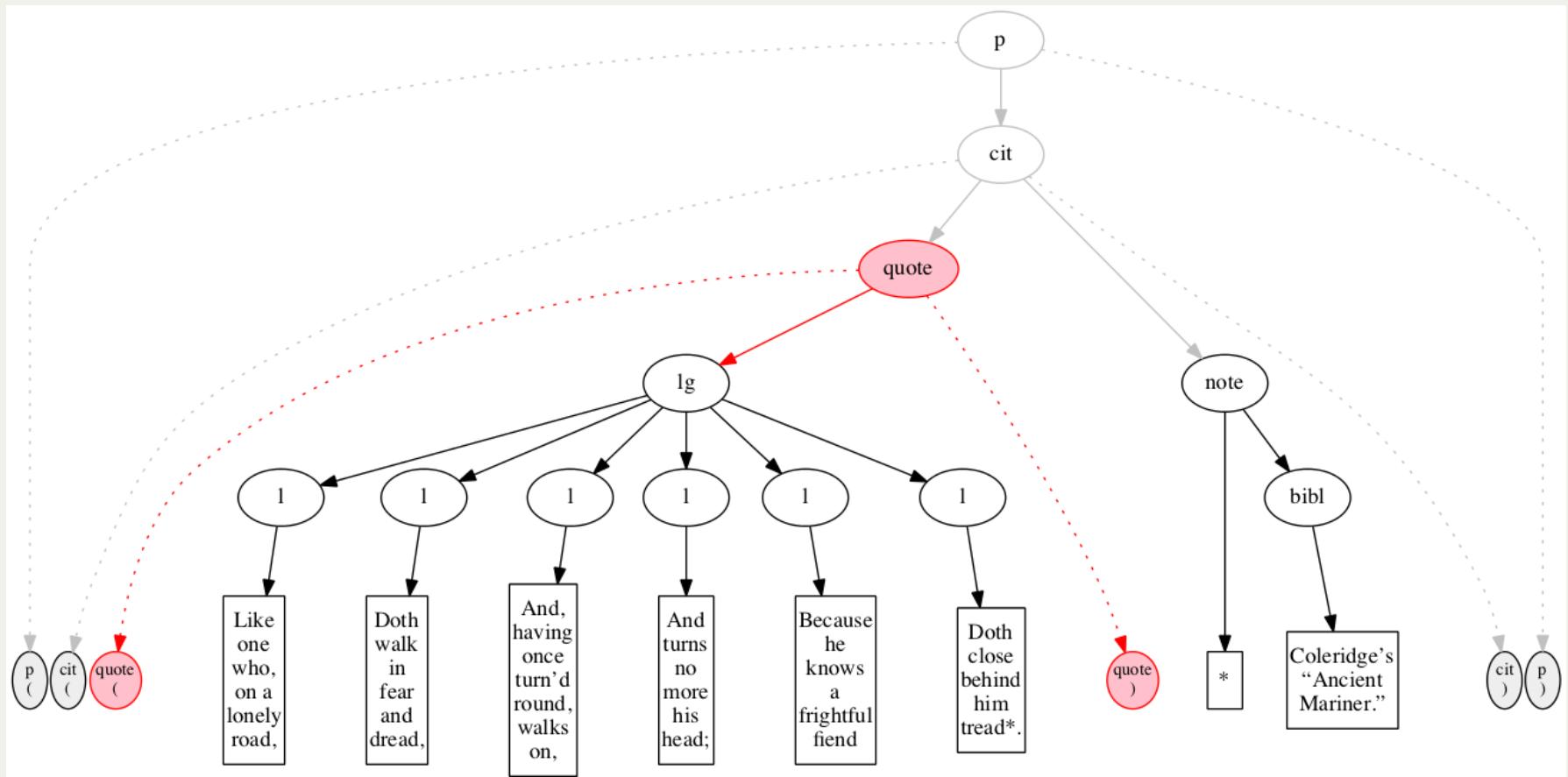
On the first pass we raise all character-only elements.

Inside out



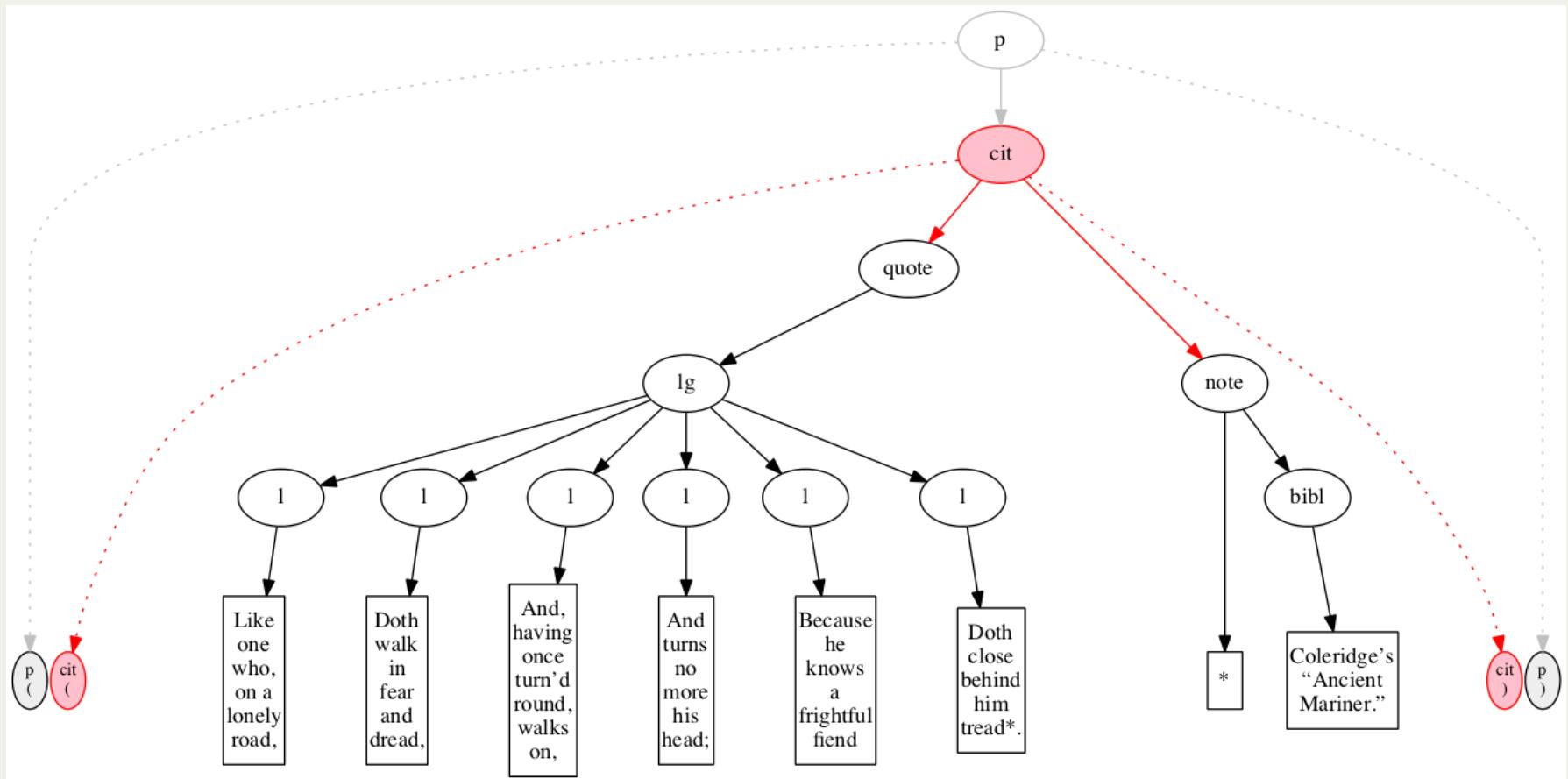
On the second pass, we raise the line group and the note.

Inside out



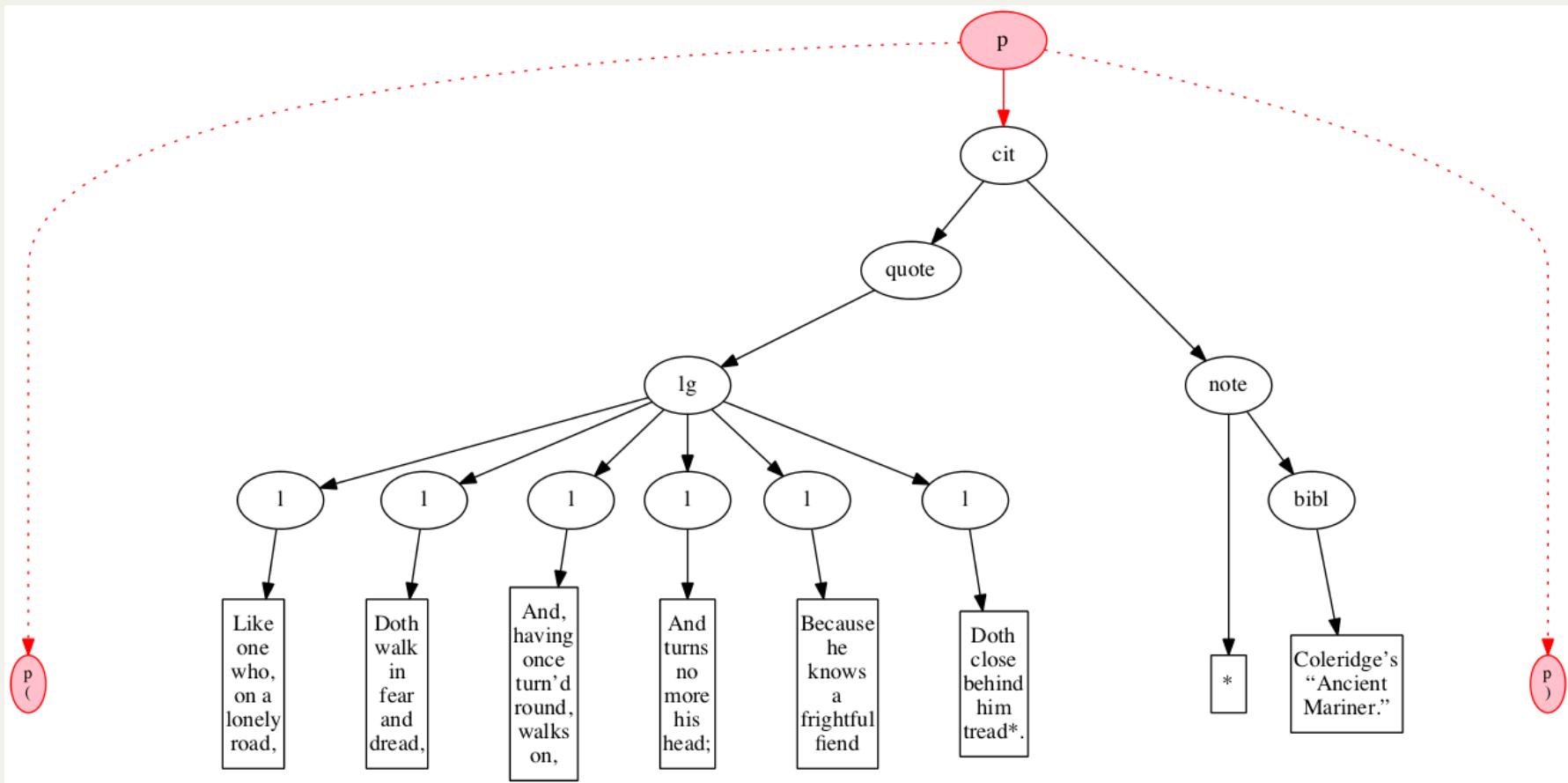
On the third pass, we raise the quote element.

Inside out



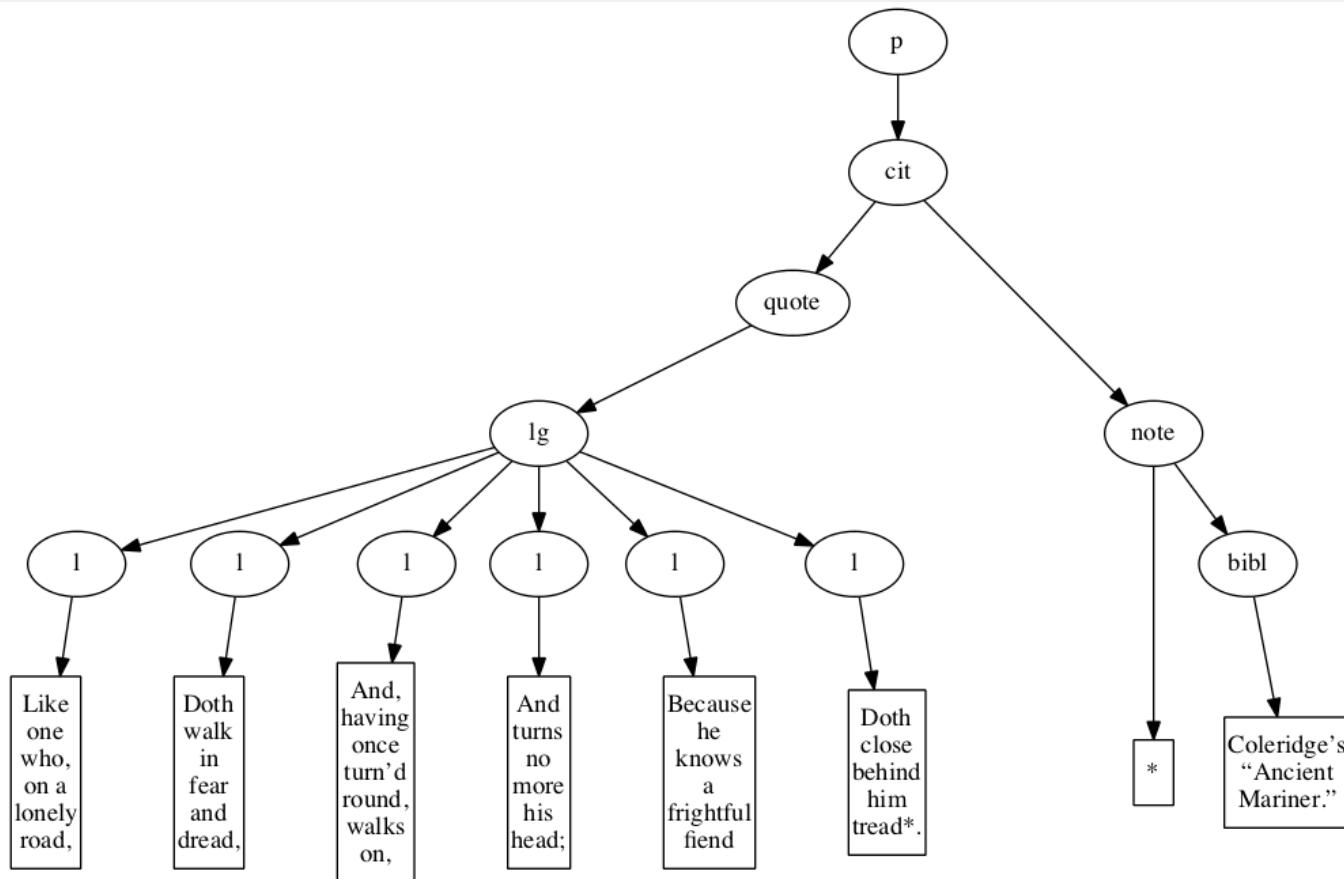
On the fourth pass, the citation element.

Inside out



On the fifth and final pass, we raise the paragraph.

Inside out



Final state. Number of passes = depth of tree.

Inside out construction

- Multiple passes
- On each pass:
 - Locate all the (would-be) innermost virtual elements
 - Raise them
- If no more nodes to process, return the result
- If more nodes to process, make another pass (recursion)

Inside-out: code

Initiation: call th:raise() on the document node:

```
<xsl:template match="/">
  <xsl:sequence select="th:raise(.)"/>
</xsl:template>
```

th:raise() function: either applies templates and then recurs or stops:

```
<xsl:function name="th:raise">
  <xsl:param name="input" as="document-node()"/>
  <xsl:choose>
    <xsl:when test="exists($input//*[@@th:sID eq
      following-sibling::*[@th:eID][1]/@th:eID])">
      <!-- If we have more work to do, do it -->
      <xsl:variable name="result" as="document-node()">
        <xsl:document>
          <xsl:apply-templates select="$input" mode="loop"/>
        </xsl:document>
      </xsl:variable>
      <xsl:sequence select="th:raise($result)"/>
    </xsl:when>
    <xsl:otherwise>
      <!-- We have no more work to do, return the input unchanged. -->
      <xsl:sequence select="$input"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:function>
```

Inside-out: code

Matching the virtual innermost pairs

```
<xsl:template match="*[@th:sID eq  
    following-sibling::*:[@th:eID][1]/@th:eID]" priority="1">  
    <xsl:copy copy-namespaces="no">  
        <xsl:copy-of select="@* except @th:sID"/>  
        <xsl:variable name="end-marker" as="element()"  
            select="following-sibling::*:[@th:eID][1]"/>  
        <xsl:copy-of select="following-sibling::node()  
            [ . &lt;&lt; $end-marker ]"/>  
    </xsl:copy>  
</xsl:template>
```

If the first end-marker we see after a start-marker matches, we have a virtual innermost pair. Raise it!

Inside-out: code

Suppress anything inside a virtual innermost

```
<xsl:template match="node()  
  [preceding-sibling::*[@th:sID][1]/@th:sID  
   eq  
   following-sibling::*[@th:eID][1]/@th:eID]"  
  priority="1"/>
```

Suppress the end-marker:

```
<xsl:template match="*  
  [@th:eID eq preceding-sibling::*[@th:sID][1]/@th:sID]"  
  priority="1"/>
```

Copy anything not inside an innermost pair:

```
<!-- identity template (all modes) -->  
<xsl:template match="@* | node()" mode="#all">  
  <xsl:copy>  
    <xsl:apply-templates select="@* | node()"/>  
  </xsl:copy>  
</xsl:template>
```

Outside in

Q: Can we make a mirror-image of inside-out from the other way around?

Outside in

Q: Can we make a mirror-image of inside-out from the other way around?

A: Sort of . . .

Outside in

Q: Can we make a mirror-image of inside-out from the other way around?

A: Sort of . . .

- First recognize leftmost outermost pair and handle it, then recur to handle next pair.

Outside in

Q: Can we make a mirror-image of inside-out from the other way around?

A: Sort of . . .

- First recognize leftmost outermost pair and handle it, then recur to handle next pair.
- To handle any pair, make element, then recur on (virtual) children.

Outside in

Q: Can we make a mirror-image of inside-out from the other way around?

A: Sort of . . .

- First recognize leftmost outermost pair and handle it, then recur to handle next pair.
- To handle any pair, make element, then recur on (virtual) children.
- Length of argument sequence will be smaller on each call.
So: faster than inside-out?

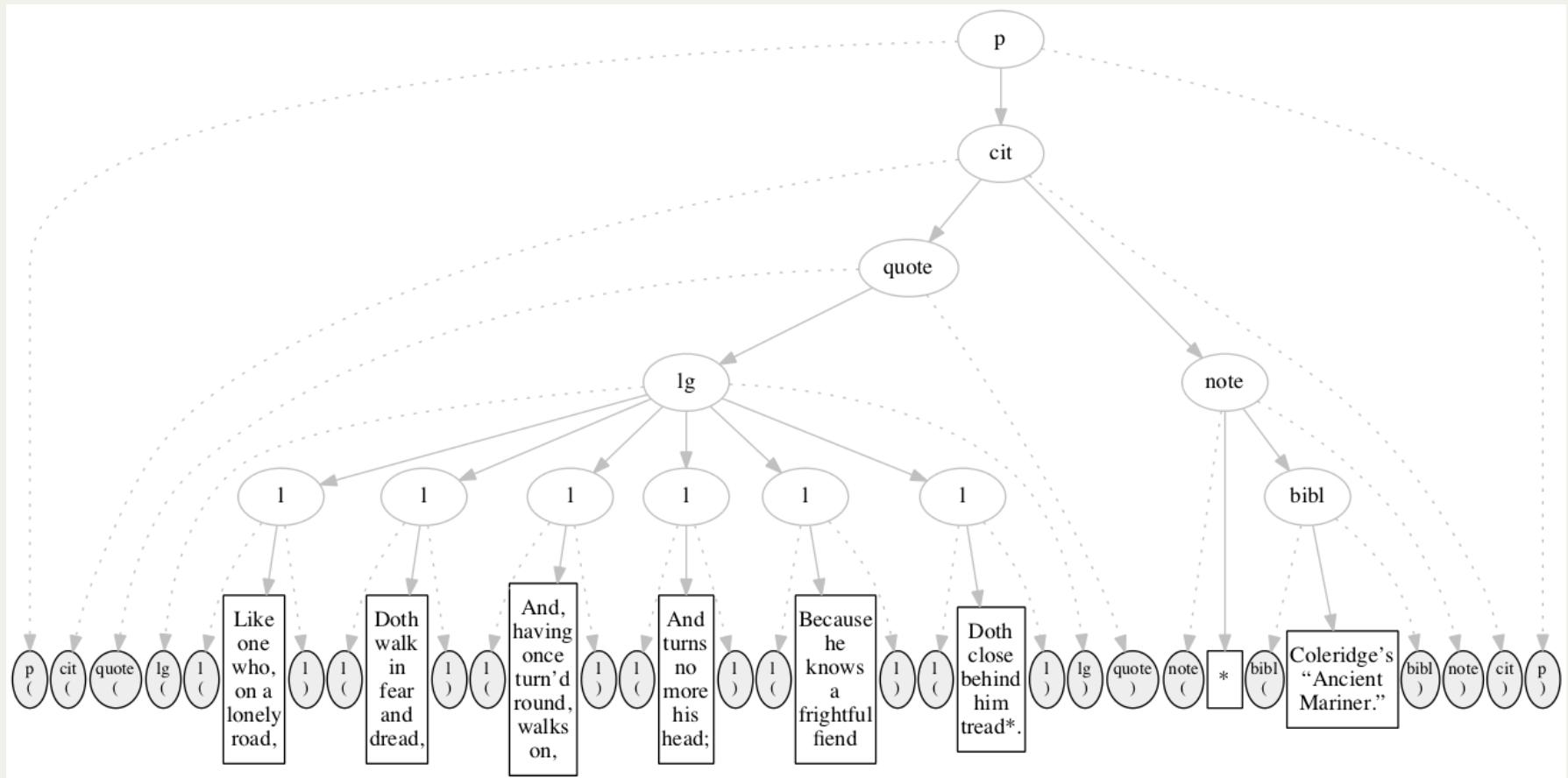
Outside in

Q: Can we make a mirror-image of inside-out from the other way around?

A: Sort of . . .

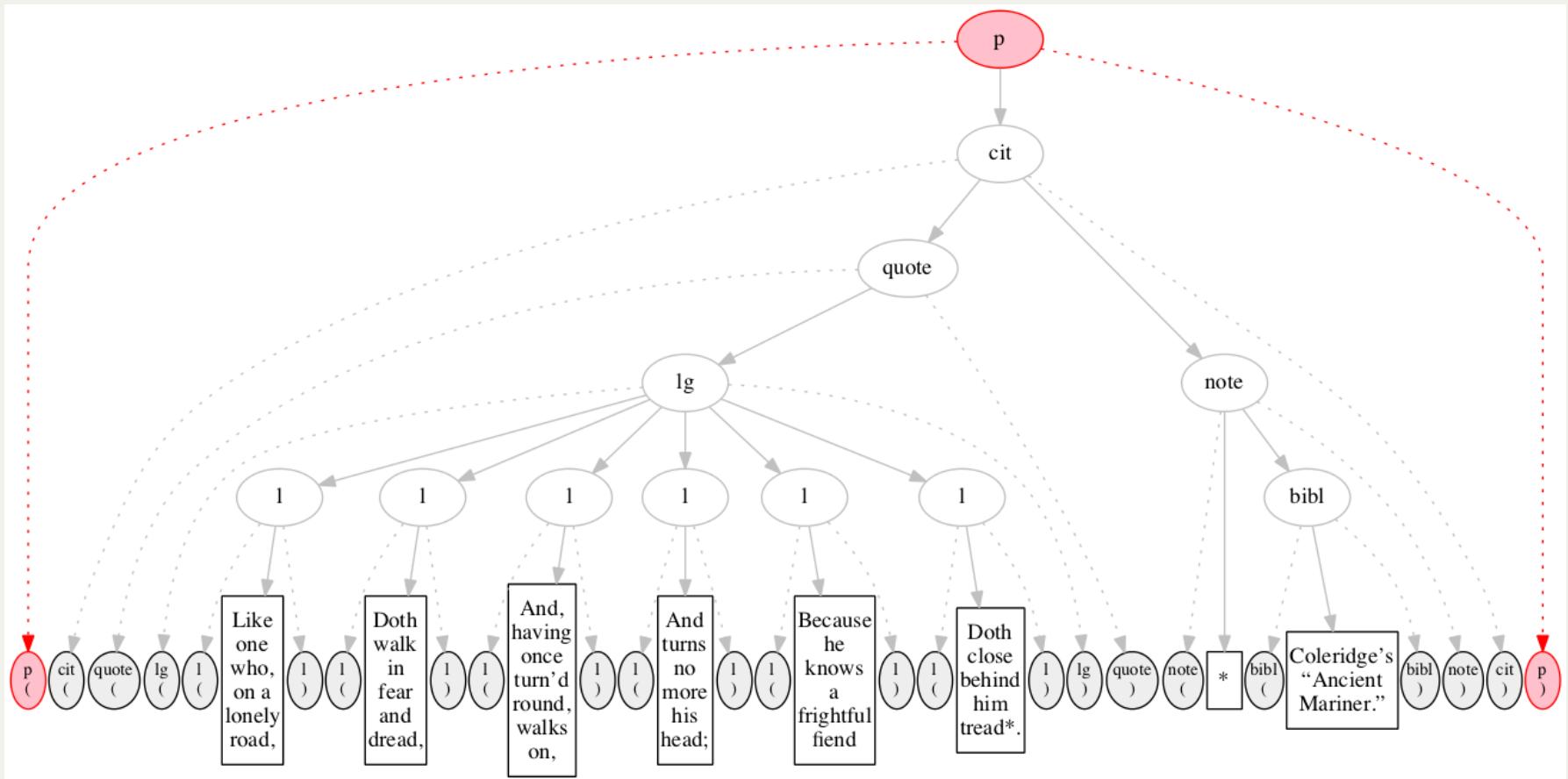
- First recognize leftmost outermost pair and handle it, then recur to handle next pair.
- To handle any pair, make element, then recur on (virtual) children.
- Length of argument sequence will be smaller on each call.
So: faster than inside-out?
- Two recursive calls for each virtual element.
So: slower than inside-out?

Outside in



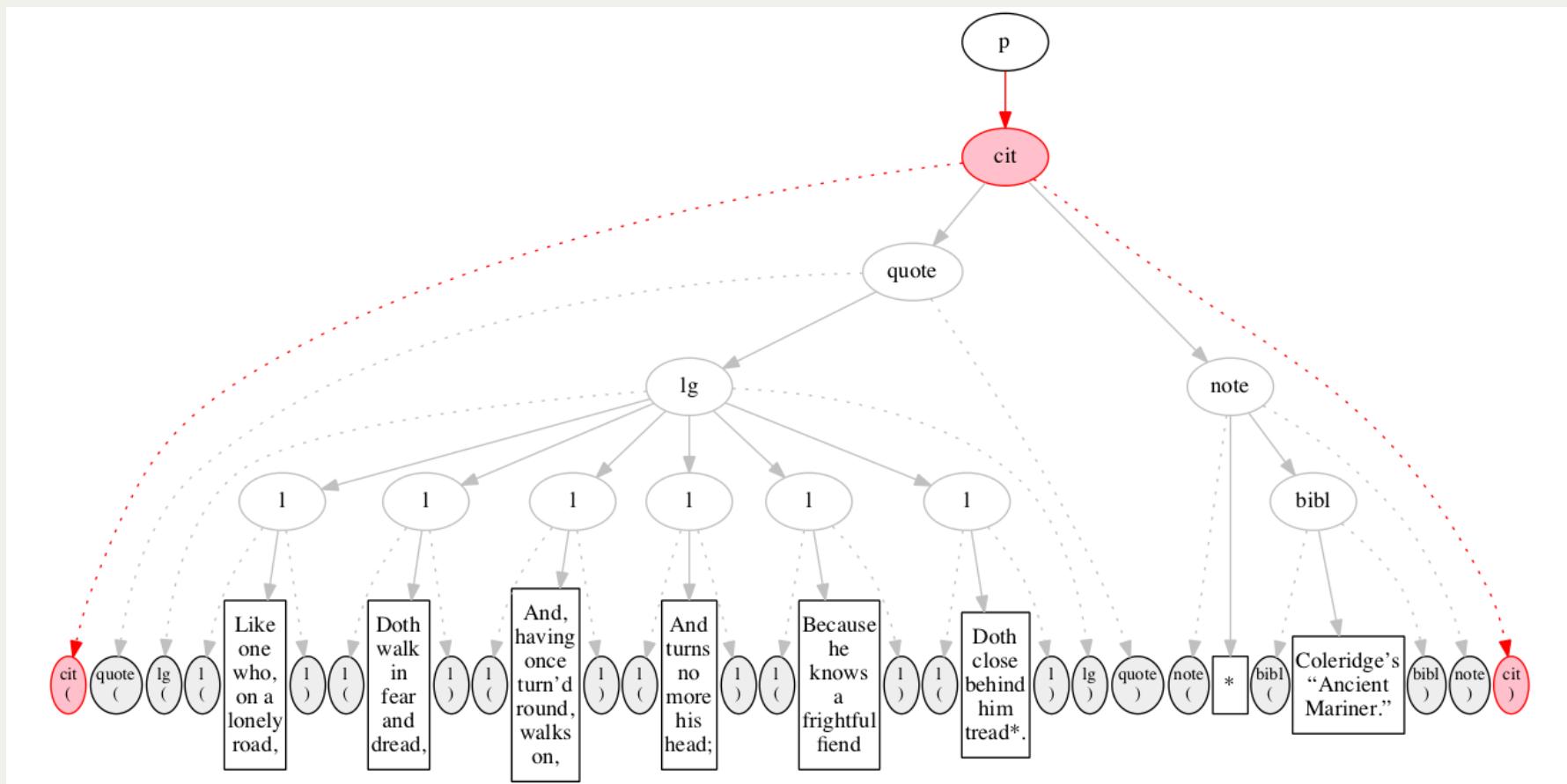
At the outset, we have a flat sequence of nodes.

Outside in



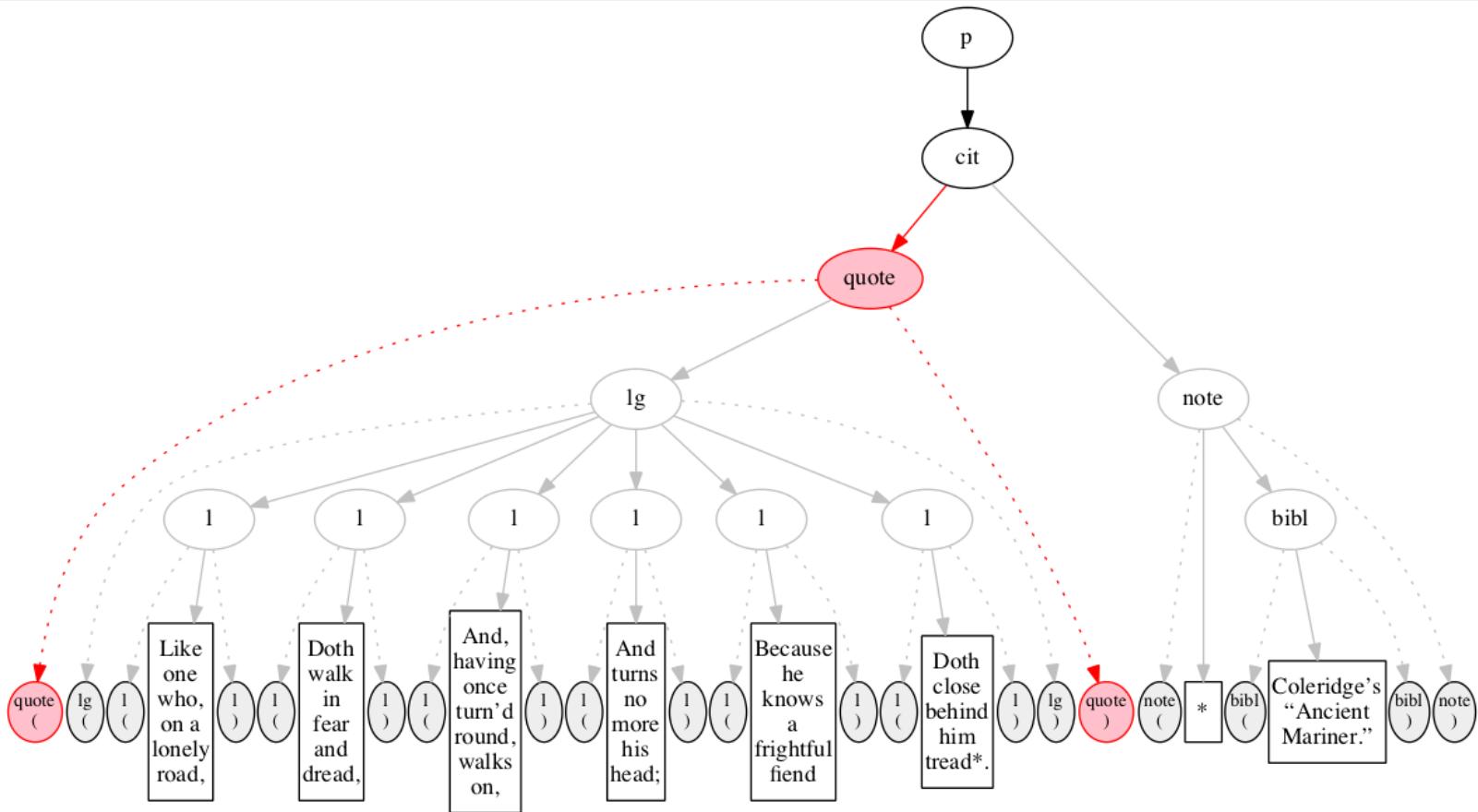
On the first pass we raise the outermost element, the <p>.

Outside in



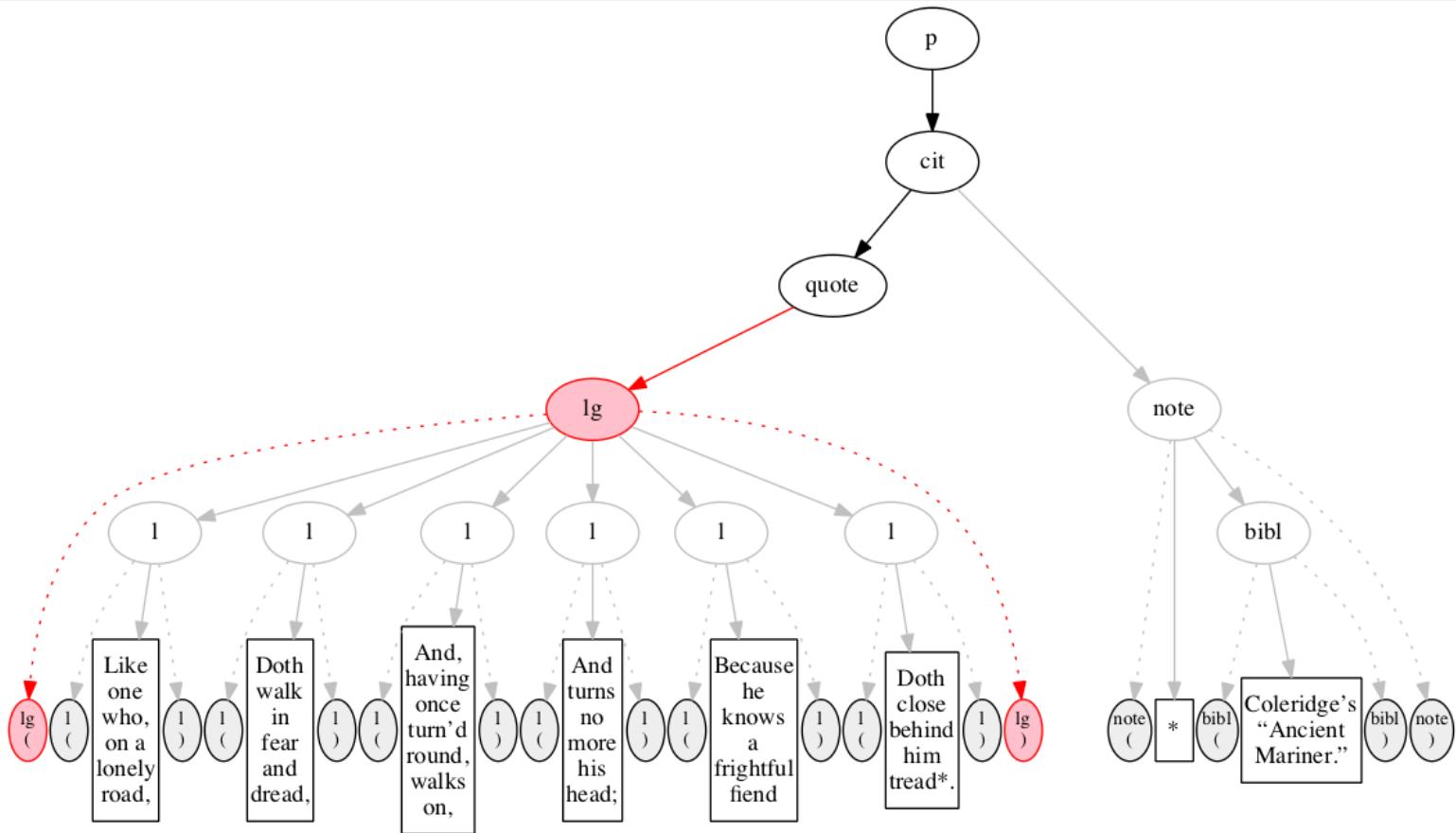
On the second pass, we raise the citation element. The `<p>` element is finished, though its children are still in process.

Outside in



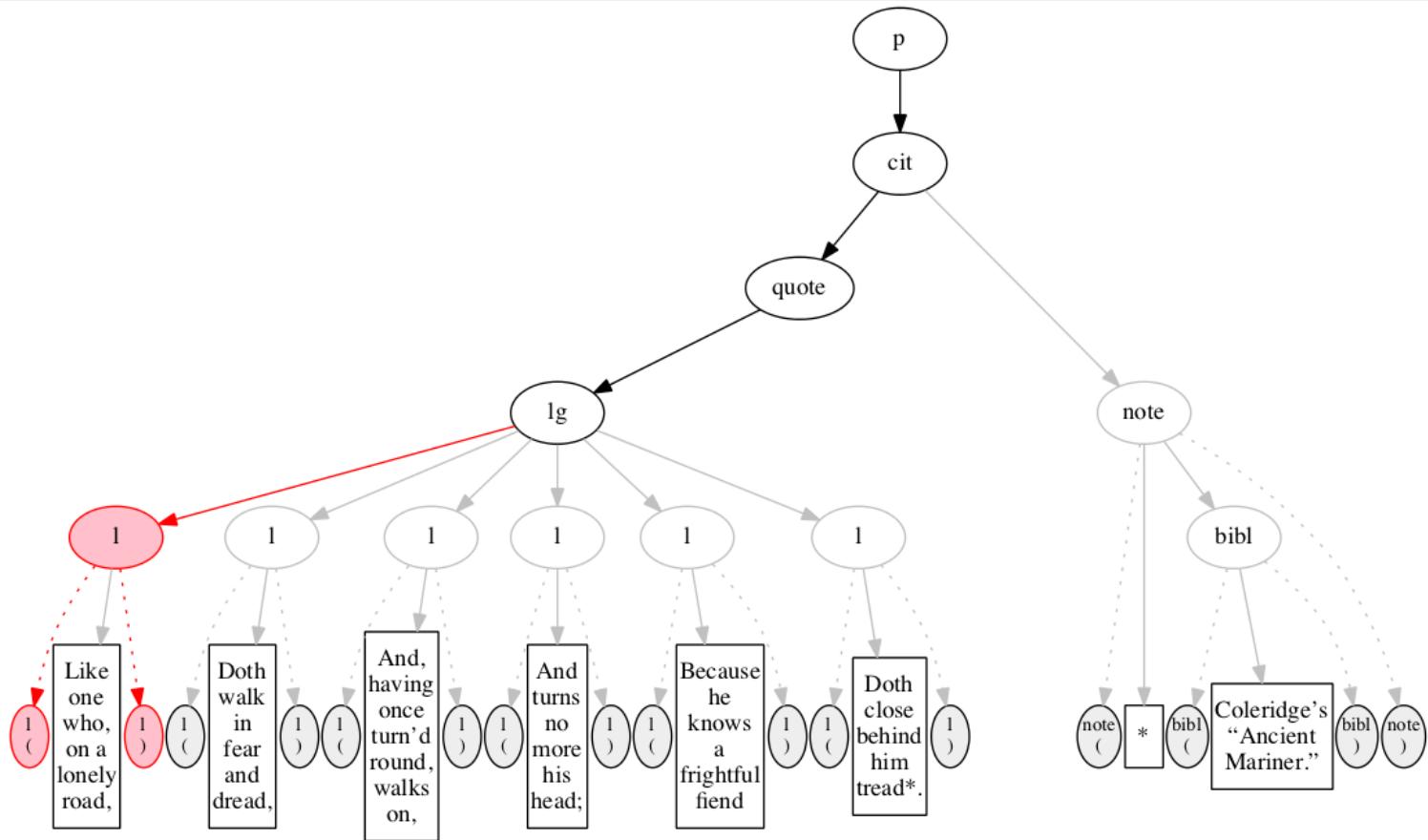
On the third pass, we raise the <quote> element, but we don't know how to raise the <note> at the same time.

Outside in



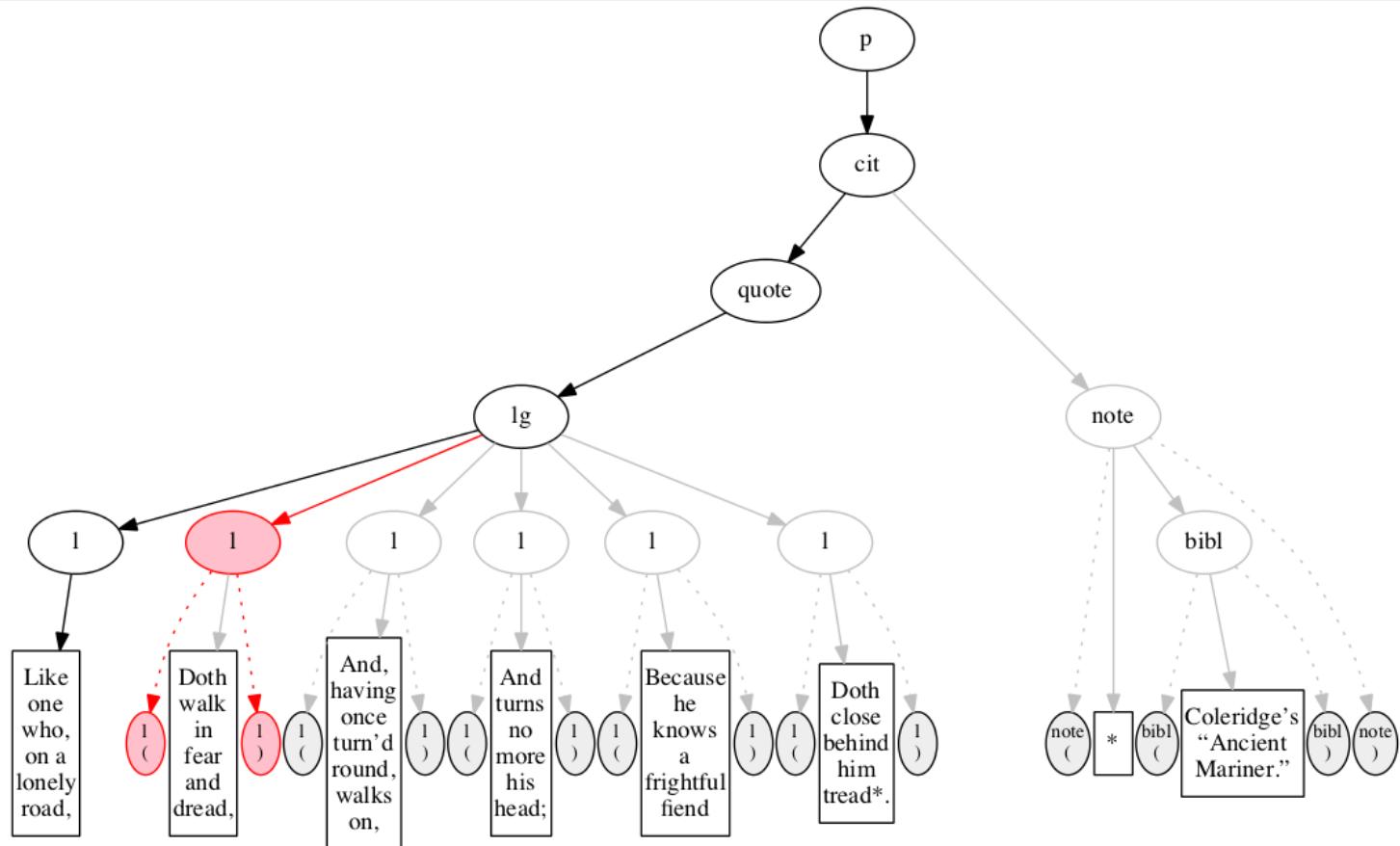
On the fourth pass, we recur on the content of the quotation and raise the line group.

Outside in



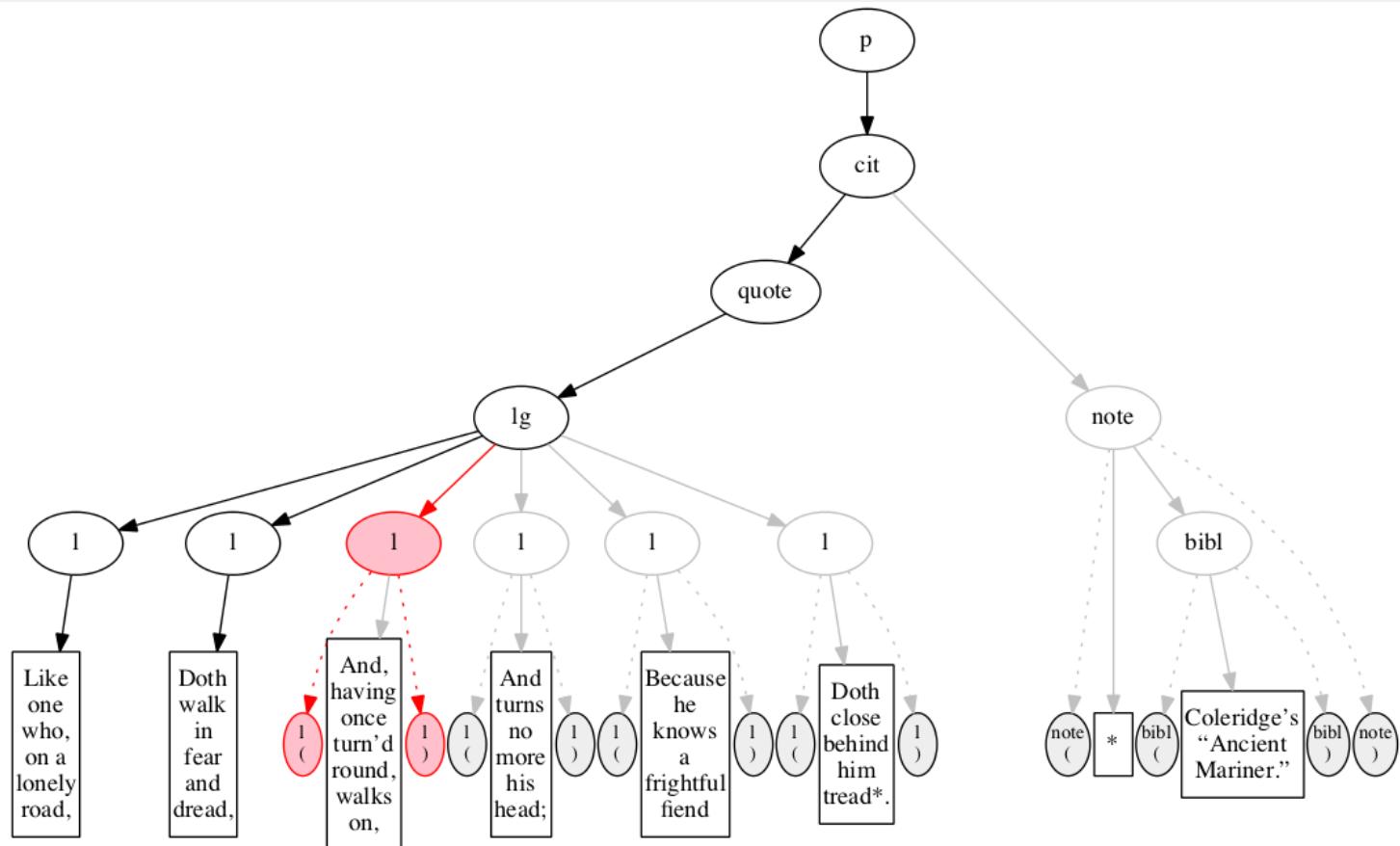
On the fifth pass, we raise the first line.

Outside in



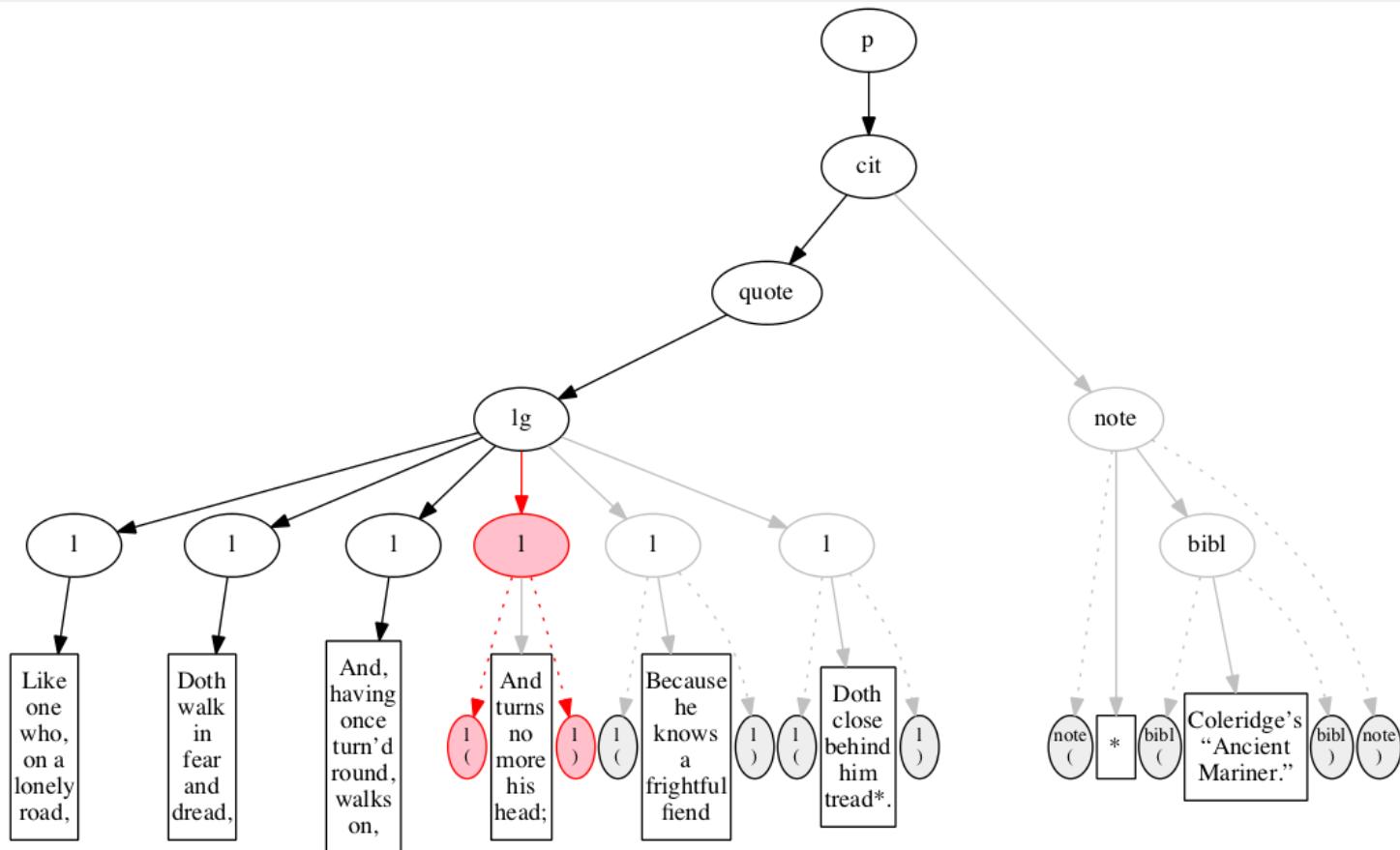
We raise the second line (6th pass).

Outside in



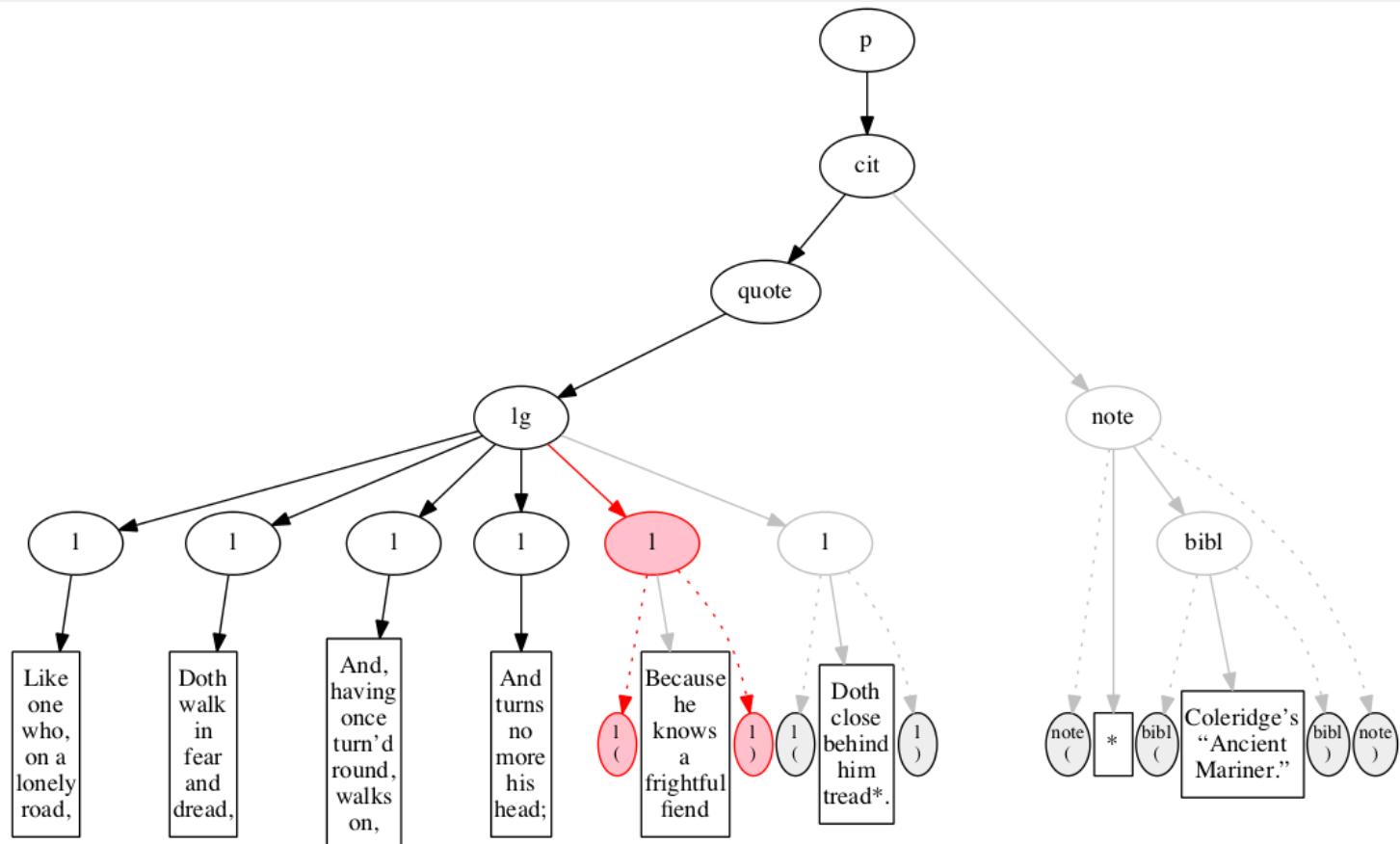
Third line (7th pass).

Outside in



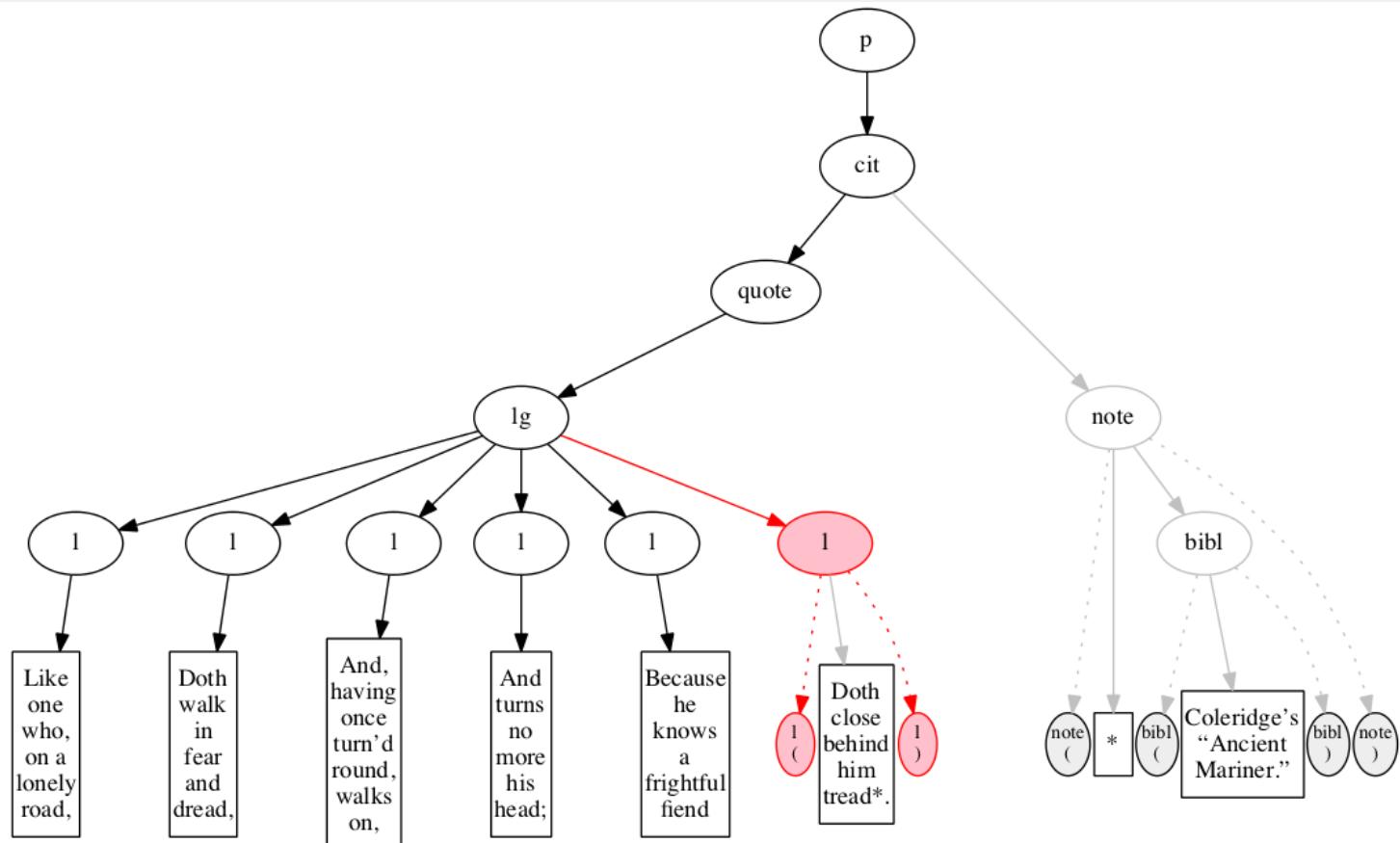
Fourth line (8th pass).

Outside in



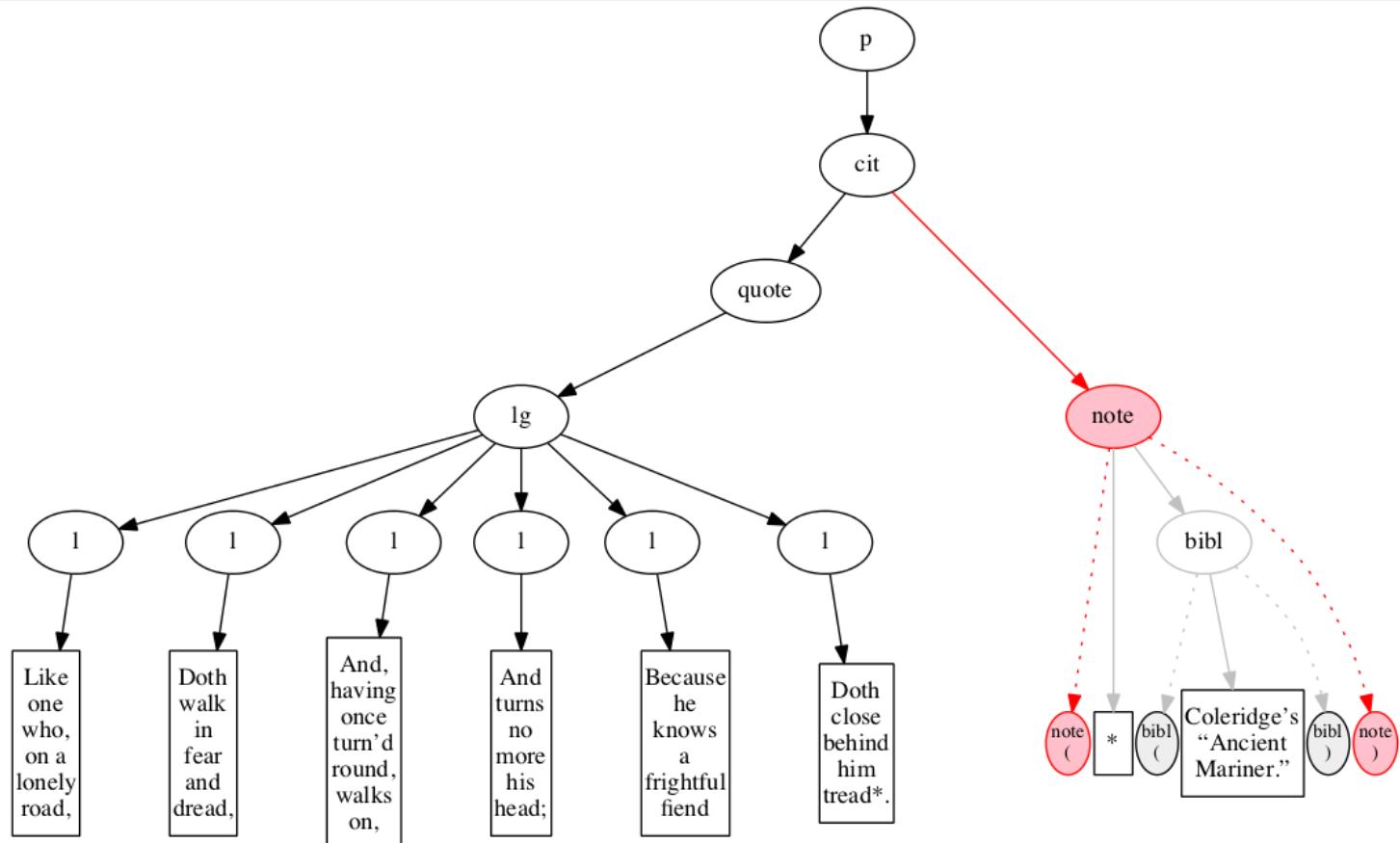
Fifth line (9th pass).

Outside in



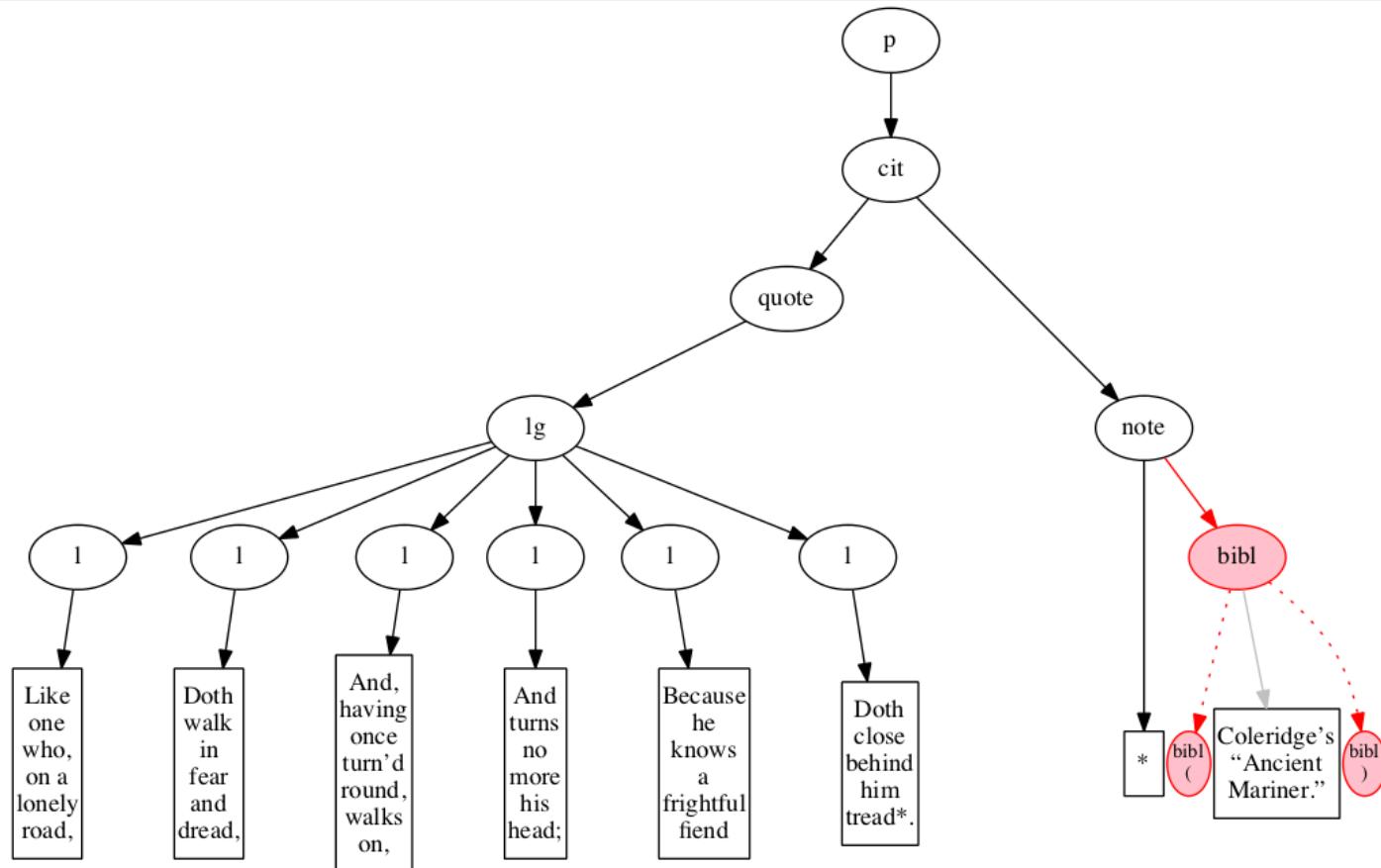
Sixth line (10th pass)

Outside in



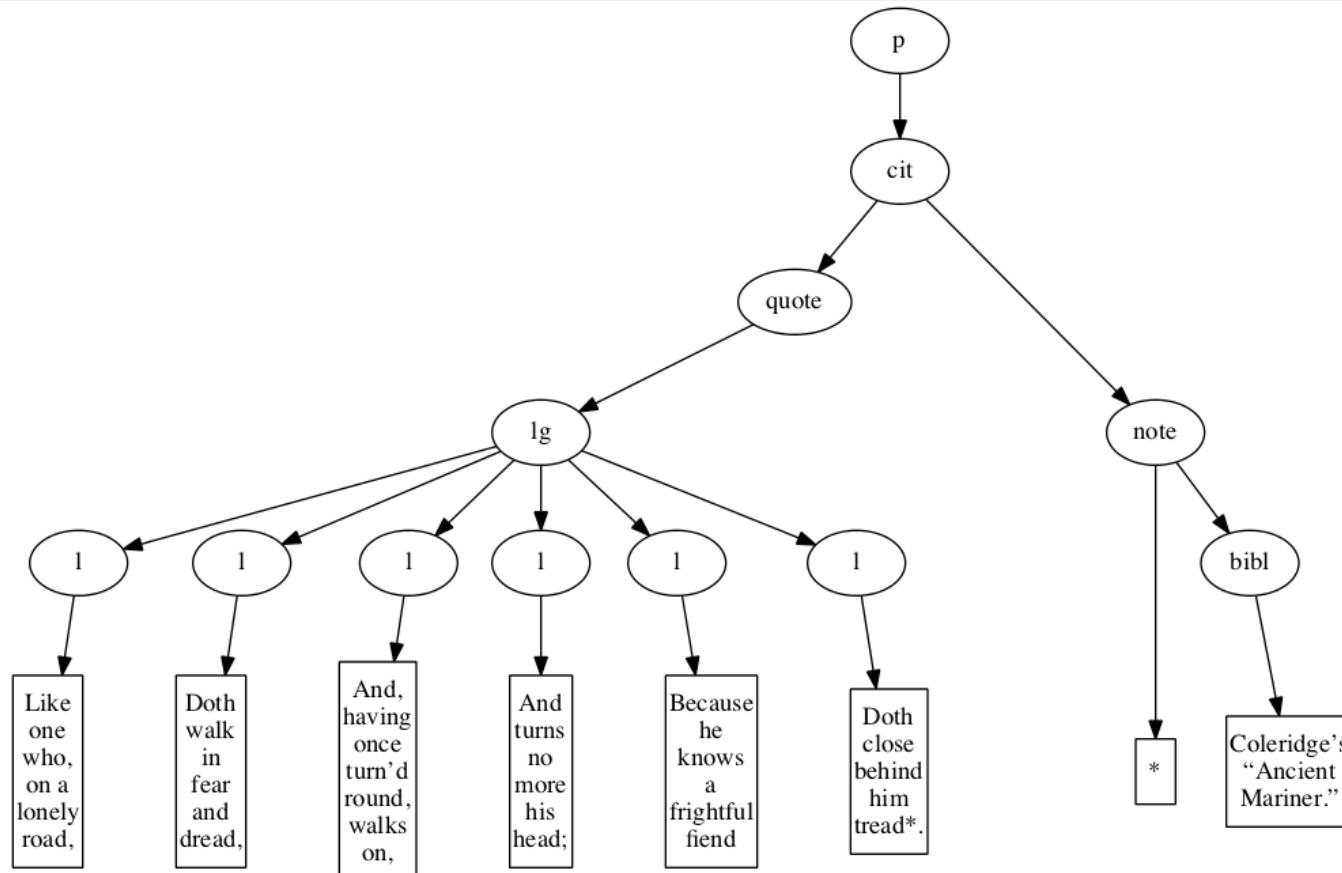
Now we finally make it to the note. (11th pass)

Outside in



On the 12th and final pass, we raise the bibliographic reference within the note.

Outside in



Final raised state.

Outside-in construction

On any content element, check for markers and raise them:

```
<xsl:template match="*[exists(node()))]">
  <xsl:copy>
    <xsl:sequence select="@*, th:raise-sequence(child::node())" />
  </xsl:copy>
</xsl:template>
```

Outside-in th:raise-sequence()

The th:raise-sequence() function checks for matching start-/end-marker pairs. If none, it terminates, returning its input. Otherwise, it has work to do.

```
<xsl:function name="th:raise-sequence" as="node(*)">
  <xsl:param name="ln" as="node()"/>

  <!--* lidStarts, lidEnds: lists of IDs for start- and end-markers *-->
  <xsl:variable name="lidStarts" as="xs:string*"
    select="for $n in $ln[th:start-marker(.)]
      return th:id($n)"/>
  <xsl:variable name="lidEnds" as="xs:string*"
    select="for $n in $ln[th:end-marker(.)]
      return th:id($n)"/>

  <xsl:choose>
    <!--* base case: no start-marker / end-marker pairs present *-->
    ...
    <!--* 'normal' case: take first start-marker with matching end-marker *-->
    ...
  </xsl:choose>
</xsl:function>
```

Outside-in: base case

If there are no matching markers present, we're done.

```
<!--* base case: no start-marker / end-marker pairs present *-->
<xsl:when test="empty($lidStarts[. = $lidEnds])">
  <!--* The sequence may contain elements with markers inside,
      * so we apply templates, instead of just returning $ln *-->
  <xsl:apply-templates select="$ln"/>
</xsl:when>
```

Outside-in: normal case

In the normal case, we

Outside-in: normal case

In the normal case, we

1. Find the first start-marker with a matching end-marker.
Divide the sequence into left, center, right subsequences

Outside-in: normal case

In the normal case, we

1. Find the first start-marker with a matching end-marker.
Divide the sequence into left, center, right subsequences
2. Apply templates to left subsequence

Outside-in: normal case

In the normal case, we

1. Find the first start-marker with a matching end-marker.
Divide the sequence into left, center, right subsequences
2. Apply templates to left subsequence
3. Make element from start-marker

Outside-in: normal case

In the normal case, we

1. Find the first start-marker with a matching end-marker.
Divide the sequence into left, center, right subsequences
2. Apply templates to left subsequence
3. Make element from start-marker
4. Recur on center subsequence (children of new element)

Outside-in: normal case

In the normal case, we

1. Find the first start-marker with a matching end-marker.
Divide the sequence into left, center, right subsequences
2. Apply templates to left subsequence
3. Make element from start-marker
4. Recur on center subsequence (children of new element)
5. Recur on right subsequence

Outside-in: find the left-most start marker

Find the first start-marker with a matching end-marker. Then divide the sequence into left, center, right subsequences.

```
<!--* 'normal' case: take first start-marker with matching end-marker *-->
<xsl:otherwise>
    <!--* find ID of first start-marker with matching end-marker *-->
    <xsl:variable name="id" as="xs:string"
        select="$lidStarts[. = $lidEnds][1]" />
    <!--* find position of start- and end-markers with that ID *-->
    <xsl:variable name="posStartEnd" as="xs:integer+"
        select="for $i in 1 to count($ln) return
            if ($ln[$i][(th:start-marker(..)
                or th:end-marker(..))
                and th:id(..) eq $id])
                then $i else ()" />
    <xsl:variable name="posStart" as="xs:integer"
        select="$posStartEnd[1]" />
    <xsl:variable name="posEnd" as="xs:integer"
        select="$posStartEnd[2]" />
```

Outside-in: lean to the left, lean to the right...

Apply templates to left subsequence.

```
<!--* Apply templates to all items to left of start. These may
     * include markers, but if so they are not matched and not
     * raisable. They may also include elements which contain
     * markers, so we need to apply templates, not just return
     * them. *-->
<xsl:apply-templates select="$ln[position() lt $posStart]" />
```

Outside-in: raise one element

Make element from start-marker; recur on center subsequence
(as children of new element).

```
<!--* Raise the element and call raise-sequence() on its
     * content. *-->
<xsl:copy select="$ln[$posStart]>
  <!--* copy the attributes (filtering as needed) *-->
  <xsl:sequence select="$ln[$posStart]/(@* except @th:*)"/>

  <!--* handle children *-->
  <xsl:sequence select="th:raise-sequence(
    $ln[position() gt $posStart
      and position() lt $posEnd]
  )"/>
</xsl:copy>
```

Outside-in:

Recur to find the next outermost pair

Recur, moving to the right subsequence:

```
<!--* call raise-sequence() on all items to right of end *-->
<xsl:sequence select="th:raise-sequence(
    $ln[position() gt $posEnd]
)"/>
</xsl:otherwise>
```

Accumulators

- Another left-right single-pass process
- Streamable
- Slightly spooky action at a distance
- Algorithm resembles pulldom rules:
 - For start-marker: push marker onto stack
 - For text nodes, comments: append to top sequence in stack
 - For end-marker: make element, pop stack, append to (new) top sequence in stack

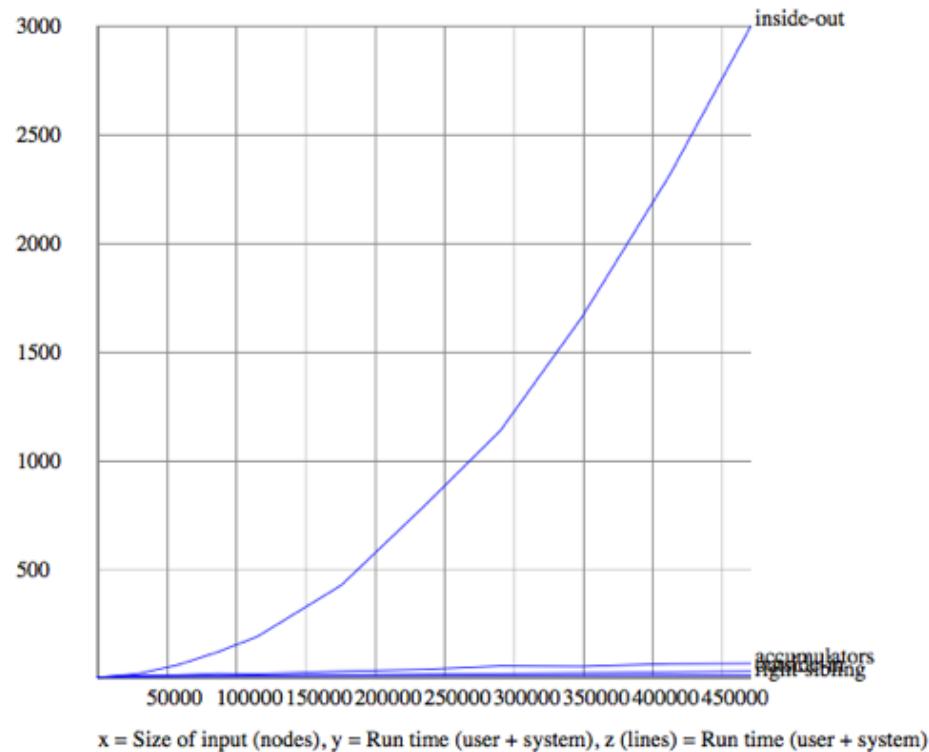
Tumbling windows

- Another grouping construct, can be based on a count (e.g. every pair of markers)
- Can be written inside-out or outside-in.

A photograph of a traditional Japanese bamboo water feature (shishi-odoshi). A large, curved bamboo pipe lies horizontally across a shallow pool of water. One end of the pipe is open, with water flowing out and creating ripples in the pool. The pipe is surrounded by lush green ferns and moss-covered rocks. The background is filled with dense green foliage.

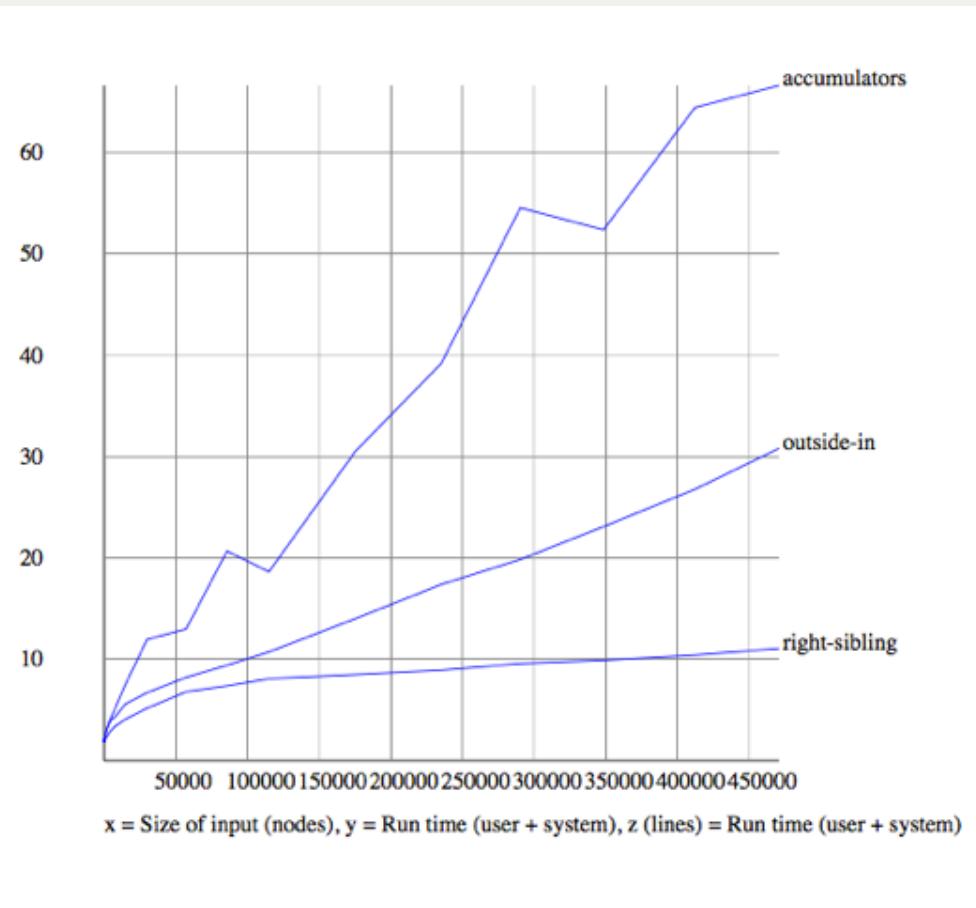
So, how do they perform?

Comparison (four-way)



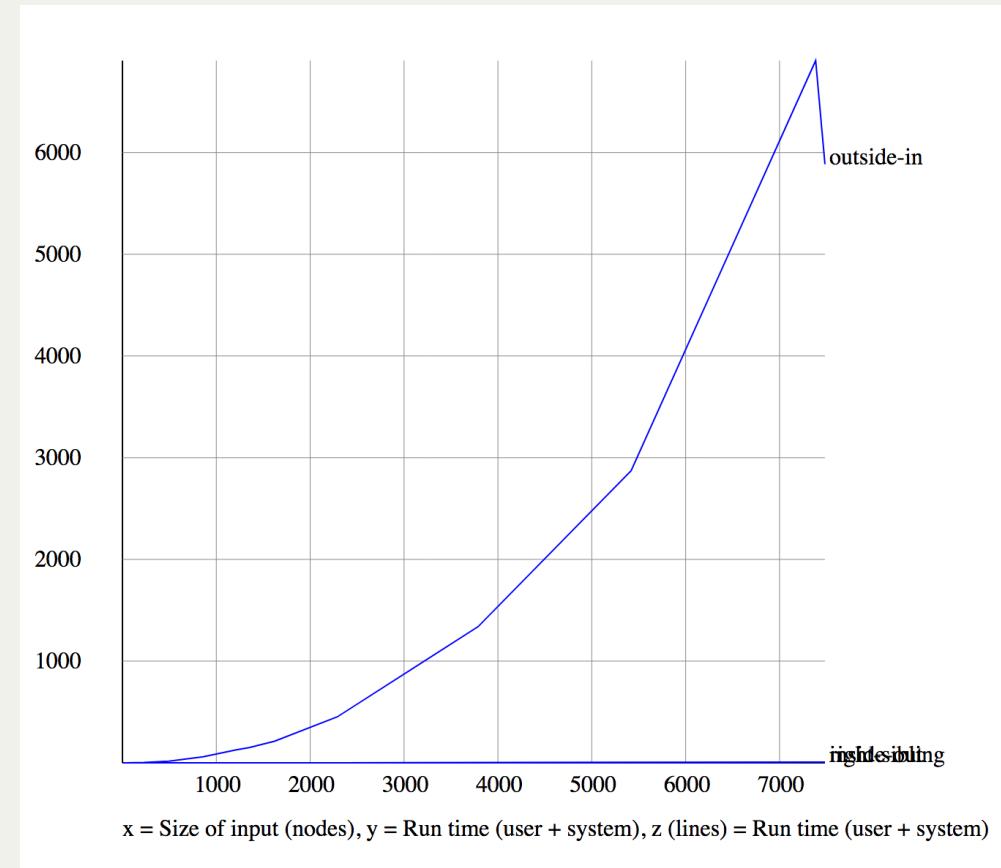
Right-sibling, outside-in, and accumulator are linear
Inside-out is quadratic

Comparison (three-way)



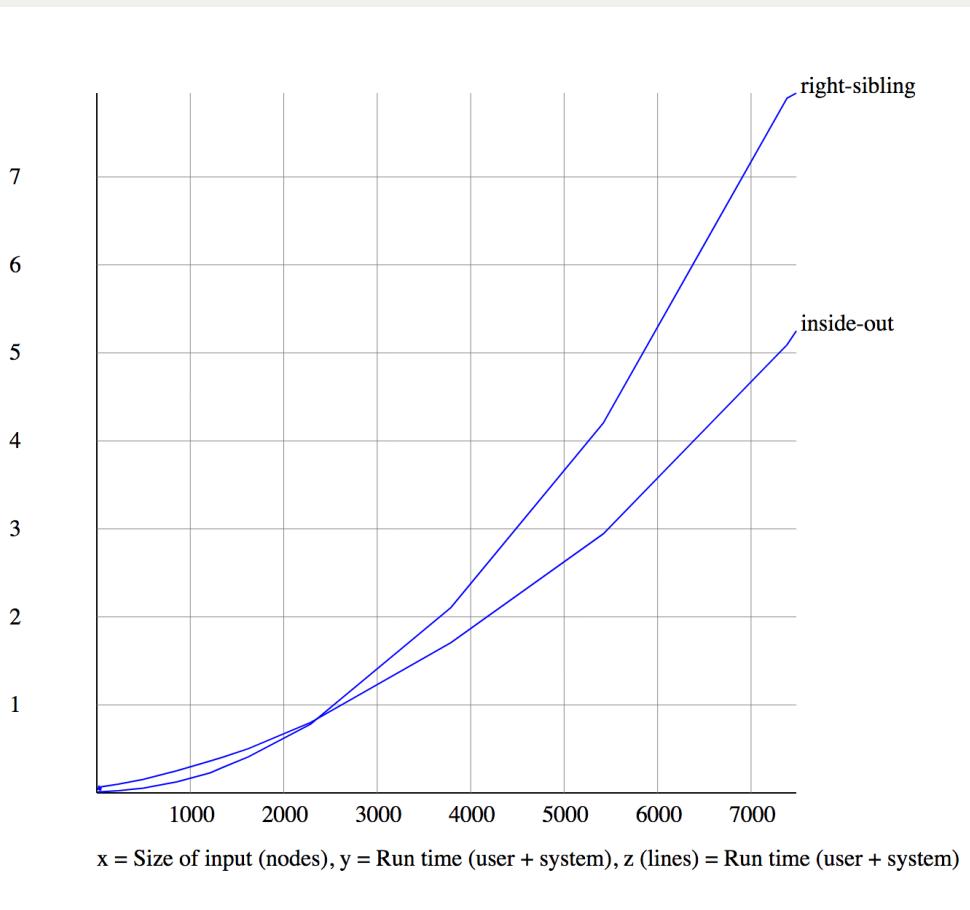
Right-sibling, outside-in, and accumulator

XSLT 1.0 comparison



Outside-in is now the problem child

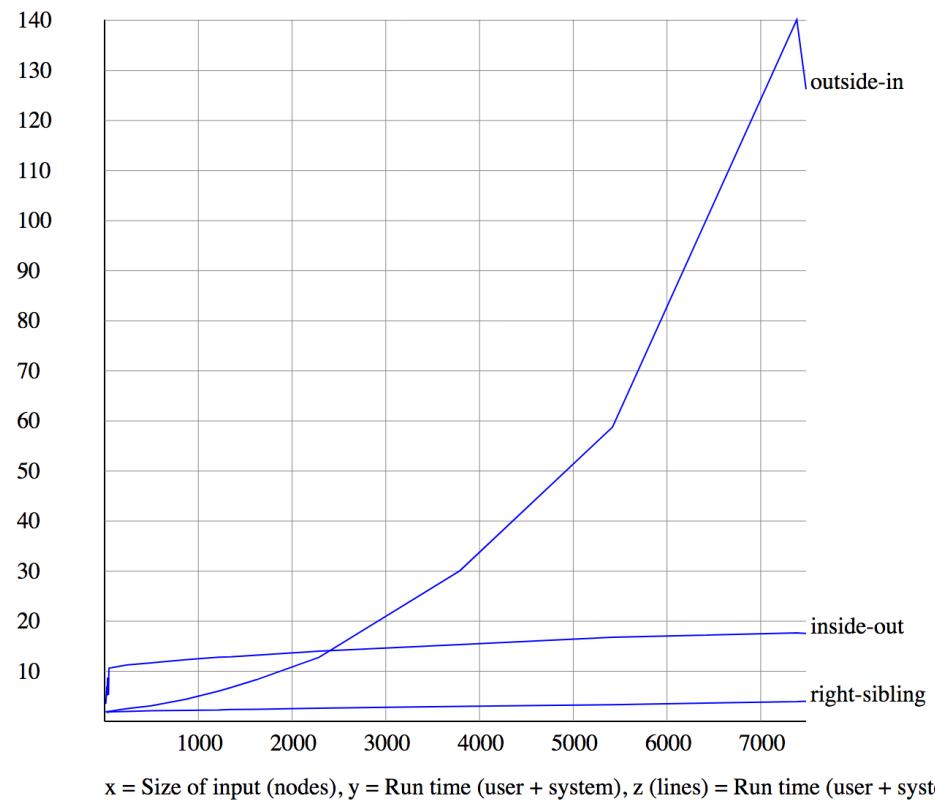
XSLT 1.0 comparison



When we remove outside-in, inside-out is better than right-sibling

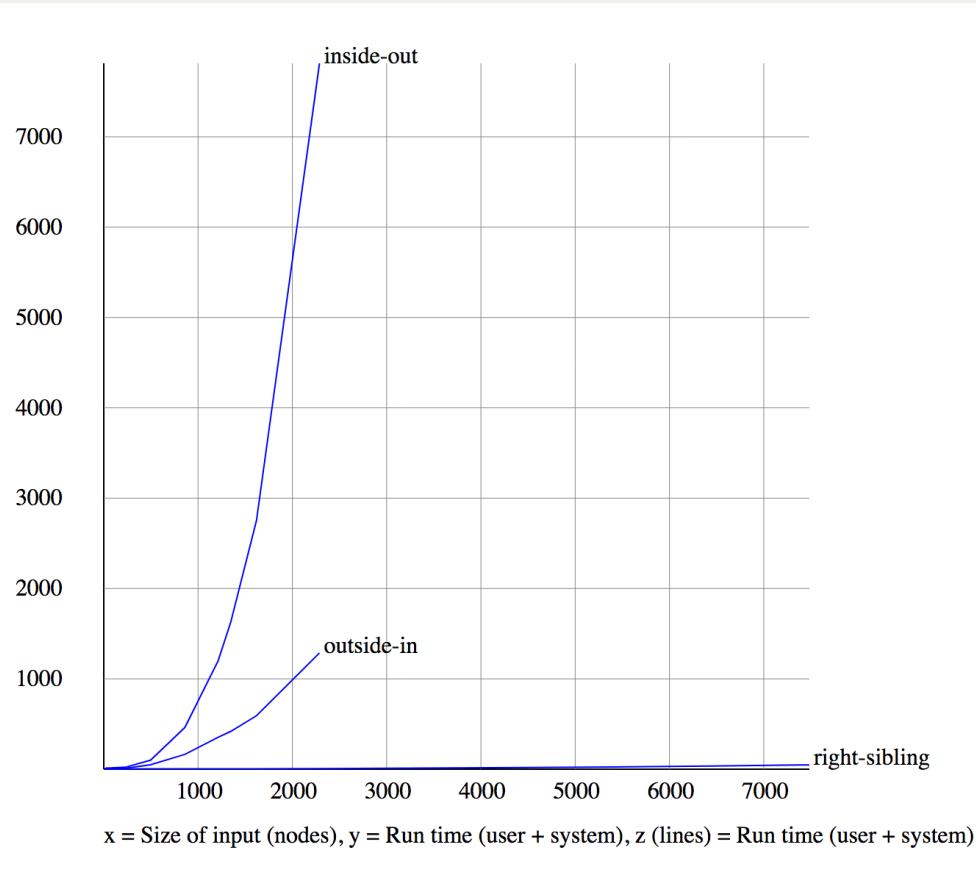
But look at the Y-axis scale

XSLT 1.0 comparison



Processor 2

XSLT 1.0 comparison



We gave up on inside-out and outside-in

Zen meditations on raising methods

Zen meditations on raising methods

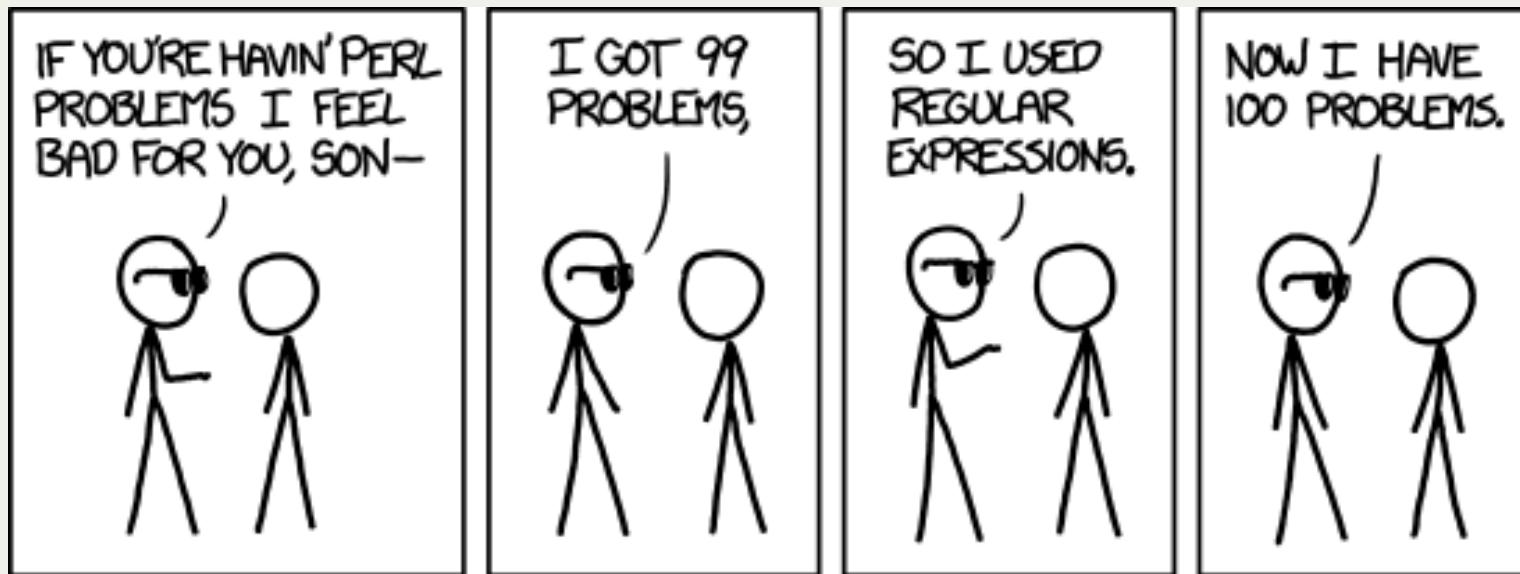
What could *possibly* go wrong?!

Feature comparison

Method	Input	Output	Raising Order	Passes / Calls	Overlap handling
Right-sibling traversal	XML	XML	Left to right	Single	Silent error
Inside-out recursion	XML	XML	Inside-out	Depth of tree	Partial raising, well formed
Outside-in recursion	XML	XML	Outside-in	Number of virtual elements	Partial raising, well formed
Accumulator	XML	XML	Left to right	Single	Silent error
Regex replacement	String	String	Left to right	Single or double	Full raising, ill formed
Python pull parser 1	XML	String	Left to right	Single	Full raising, ill formed
Python pull parser 2	XML	XML	Left to right	Single	Silent error

String-processing tags

- Pro: quick, light-weight
- Con: brittle



Pull parsing in Python

- Writing XML output as a string
 - too easy to generate overlapping tags and ill-formed attribute values
- Writing XML output as XML: more robust, but ...
 - pulldom operates on mutable objects, so ...
 - Don't remove Trojan attributes at START_ELEMENT event (you need them to process END_ELEMENT events correctly).
 - Trojan markers have a single tag (with sole-tag notation), but fire both START_ELEMENT and END_ELEMENT events.
- Namespaces: pulldom and minidom are not fully namespace-aware

Left to right sibling traversal

- Easy to forget to look for the *first* following sibling and select `following-sibling::node()` instead of `following-sibling::node()[1]`
 - makes for *gigantically enlarged* output!
- Easy to forget to apply the `@mode = "raising"`
 - causes loss of input nodes (*reduced* output)!

Inside-out recursion

- Double processing
 - must remember to suppress nodes that are being copied, since these are also candidates for applying templates
- Endless looping
 - To prevent: test for markers that can be raised without breaking hierarchy, not for all markers

Inside-out recursion

Scans too far in case end-marker repeats (it doesn't)

```
<xsl:copy-of  
    select="following-sibling::node()  
    [following-sibling::*[@th:eID eq current()/@th:sID]]"/>
```

Finds the one match quickly

```
<xsl:variable  
    name="end-marker"  
    as="element()"  
    select="key('end-markers', @th:sID)"/>
```

Accumulators

- Be sure your Saxon processor is fully XSLT 3.0.
 - Don't try this with Saxon HE 9.6.0.5
- Don't forget to specify `@use-accumulators= "stack"` on default mode declaration.

For further investigation

- Performance analysis

For further investigation

- Performance analysis
 - Memory usage? Stack space?

For further investigation

- Performance analysis
 - Memory usage? Stack space?
 - Are functions faster than named templates? Slower? Indifferent?

For further investigation

- Performance analysis
 - Memory usage? Stack space?
 - Are functions faster than named templates? Slower? Indifferent?
 - Are patterns of performance constant across processors?

For further investigation

- Performance analysis

- Memory usage? Stack space?
- Are functions faster than named templates? Slower? Indifferent?
- Are patterns of performance constant across processors?
- Do XQuery function implementations of inside-out and outside-in show same performance patterns?

For further investigation

- Performance analysis

- Memory usage? Stack space?
- Are functions faster than named templates? Slower? Indifferent?
- Are patterns of performance constant across processors?
- Do XQuery function implementations of inside-out and outside-in show same performance patterns?
- Why is outside-in tumbling window performance so bad in preliminary tests?

For further investigation

- Performance analysis
 - Memory usage? Stack space?
 - Are functions faster than named templates? Slower? Indifferent?
 - Are patterns of performance constant across processors?
 - Do XQuery function implementations of inside-out and outside-in show same performance patterns?
 - Why is outside-in tumbling window performance so bad in preliminary tests?
- More solutions?