Algorithm Overview

The basic idea of the algorithm is to "fold" an XML annotation into a SVG in order to create a tactile diagram. The XML annotation is a navigation structure that describes multiple hierarchical levels and contains speech annotations its elements. It is generally part of an XML document that combines it with scientific data (e.g., chemical data). The SVG is an ordinary SVG diagram, however, hierarchically structured with some elements having id attributes that correspond to elements of the annotation.

The folding process enriches the SVG, by adding title and description elements for the overall SVG as well as to the drawn elements. In addition, it introduces invisible regions by combining multiple drawn elements together. These are elements

E.g., a Benzene ring is composed of six bonds (some of which are double bonds), which are actual drawn elements in the SVG. The ring itself is simulated by adding an invisible polygon to the SVG that contains the single bonds but lies behind them (in terms of z layer). This invisible element furnishes title and description for the Benzene ring.

We have effectively three types of annotation elements. The motivation behind some of these elements is best observed with an example, e.g., `https://progressiveaccess.com/chemistry/generic.html?mole=data/aspirin-enr`.

active Single elements that form the base level of the navigation structure. Some active elements do not correspond to a drawn element in the SVG. They have to be computed from their neighbouring passive elements. E.g., Carbon atoms in a chemical molecule can be given by the junction of two or more bonds.

passive Single elements that can be attached to active elements but are not interesting enough to be navigatable in their own right in online navigation. However, for tactile diagrams they need to be explorable, i.e., get their own title and description.

They always correspond to existing, drawn elements in the SVG.

grouped Elements that form higher level nagivation elements. They correspond to compound structures, made up of active, passive or other grouped elements. They have initially no correspondence in the SVG and have to be created as invisible polygons.

Algorithm Outline

Input: Structured SVG
XML markup
Output: Audio-tactile SVG
Cases

1. For every `active` and `passive` element. If SVG element with corresponding id exists, add speech to SVG element. If necessary:
   (a) embed elements into a new SVG group element,
   (b) put an invisible bounding polygon around thin lines.
2. For every `active` element that does not exists
   (a) Compute a bounding box by considering the internal neighbours (i.e., for take the via elements of neighbour elements with location attribute "internal").
   (b) Create `active` into invisible rectangle with the computed bounding box.
   (c) Add speech to new rectangle.
3. For each `grouped` element with parent elements
   (a) Compute a bounding polygon by combining the SVG bounding boxes of its components.
   (b) Add an invisible polygon for the computed bounding polygon.
   (c) Add speech to new polygon.
4. For the `grouped` element without parent (top-most element) add speech as title and description of the SVG.

Add speech means add `speech` attribute as SVG title element and `speech2` attribute (if it exists) as SVG descr element.

Grammar

The following is a grammar outline in extended BNF for the annotation elements.

⟨*diagram*⟩ ::= '`<`' ⟨*type*⟩ '`>`' ⟨*data*⟩* ⟨*annotations*⟩ '`</`' type '`>`'

⟨*data*⟩ ::= . . . can be ignored . . .

⟨*type*⟩ ::= '`histogram`'
| '`circuit`'
| '`molecule`'
| . . .

⟨*annotations*⟩ ::= '`<annotations>`' ⟨*annotation*⟩* ⟨*messages*⟩* '`</annotations>`'

⟨*annotation*⟩ ::= '`<annotation speech="`' ⟨*speech*⟩ '`" speech2="`' ⟨*speech*⟩ '`">`' ⟨*element*⟩ ⟨*position*⟩ ⟨*parents*⟩ ⟨*children*⟩ ⟨*component*⟩ ⟨*neighbours*⟩ '`</annotation>`'

⟨*element*⟩ ::= ⟨*active*⟩
| ⟨*passive*⟩
| ⟨*grouped*⟩

⟨*active*⟩ ::= '`<active>`' ⟨*id*⟩ '`</active>`'

⟨*passive*⟩ ::= '`<passive>`' ⟨*id*⟩ '`</passive>`'

⟨*grouped*⟩ ::= '`<grouped>`' ⟨*id*⟩ '`</grouped>`'

⟨*position*⟩ ::= '`<position>`' ⟨*number*⟩ '`</position>`'

⟨*parents*⟩ ::= '`<parents>`' ⟨*grouped*⟩? '`</parents>`'

⟨*children*⟩ ::= '`<children>`' ⟨*element*⟩* '`</children>`'

⟨*component*⟩ ::= '`<component>`' ⟨*element*⟩* '`</component>`'

⟨*neighbours*⟩ ::= '`<neighbours>`' ⟨*neighbour*⟩* '`</neighbours>`'

⟨*neighbour*⟩ ::= '`<neighbour speech="`' ⟨*speech*⟩ '`" speech2="`' ⟨*speech*⟩ '`" location="`' ⟨*location*⟩ '`">`' ⟨*element*⟩ ⟨*via*⟩+ '`</neighbour>`'

⟨*location*⟩ = internal | external

⟨*via*⟩ ::= '`<via>`' ⟨*passive*⟩ ⟨*position*⟩ '`</via>`'

⟨*id*⟩ ::= ⟨*alpha*⟩ ⟨*id*⟩* | ⟨*digit*⟩ ⟨*id*⟩* | ⟨*symbol*⟩ ⟨*id*⟩*

⟨*number*⟩ ::= ⟨*digit*⟩ | ⟨*digit*⟩ ⟨*number*⟩

⟨*digit*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨*alpha*⟩ ::= a-z

⟨*Alpha*⟩ ::= A-Z

⟨*symbol*⟩ :: = - | _

⟨*speech*⟩ ::= ⟨*string*⟩ | ⟨*msgid*⟩

⟨*string*⟩ ::= '`Unicode`'*

⟨*messages*⟩ ::= '`<language>`' ⟨*language*⟩ '`</language>`' ⟨*message*⟩*

⟨*message*⟩ ::= '`<message msg="`' ⟨*msgid*⟩ '`">`' ⟨*string*⟩ '`</message>`'

⟨*msgid*⟩ ::= ⟨*Alpha*⟩ ⟨*msgid*⟩* | ⟨*symbol*⟩ ⟨*msgid*⟩* | ⟨*number*⟩ ⟨*msgid*⟩*

⟨*language*⟩ ::= iso-639-1 language code

Notes:

1. The grammar is defined in two namespaces:
   - *data* elements are definied in a namespace suitable for the expressed data.
   - *annotations* elements are defined in the `sre` namespaces. Hence elements are usually of the form `sre:annotations` etc.
2. The `component` element for active element annotations can only contain `passive` elements.
3. Multi-linguality is achieved via the message elements.
   - In case `messages` elements are present, the `<speech>` elements are `<id>`s
   - If there are no `messages` elements, the attributes `speech` and `speech2` contain each contain a ¡string¿ only.
4. Special case of molecule diagrams:
   - `active`, `bond`, `grouped`, are called `atom`, `bond`, `atomSet`, respectively.