RuleML 2013
7th International Rule Challenge

University of Washington,
July 11-13th, 2013, Seattle, WA

Martin Koch,
Sven Schmidt,
Harold Boley,
Rainer Herpers

Grailog KS Viz: A Grailog Visualizer for Datalog RuleML Using an XSLT Translator to SVG

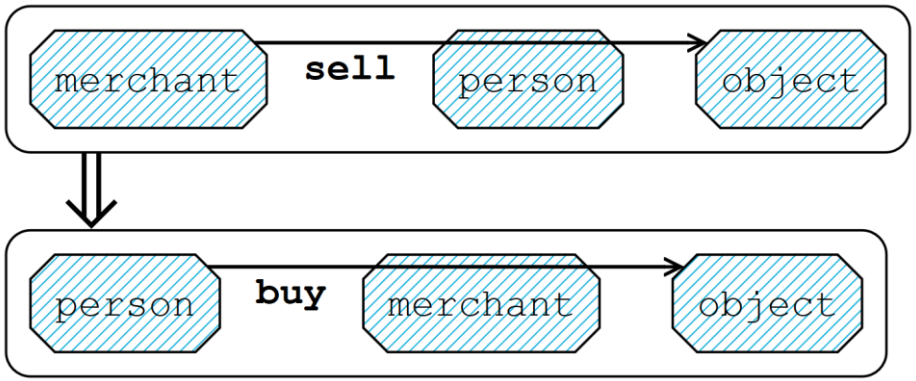
Agenda

- Introduction
  - Graph inscribed logic (Grailog)
  - Approach overview
  - Supported Grailog elements
  - Example
- SVG elements
  - Used elements
- Transformation using XSLT
  - Basic structure of Grailog KS Viz
- Demo
- Conclusion & Future Work

Graph inscribed logic (Grailog)

- Systematic combination of generalized graph constructs for **data & knowledge representation**
- 2-dimensional graph-logic **visualization for humans** in the loop of data & knowledge elicitation, specification, validation and reasoning

Grailog:



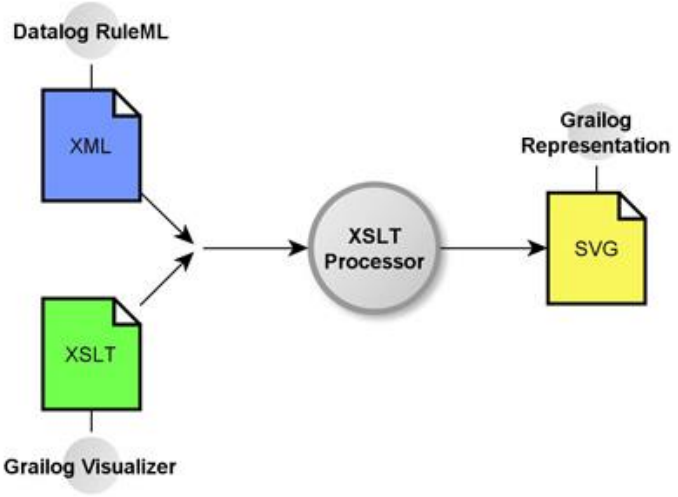
Datalog RuleML:

```
<Implies>
    <Atom>
      <Rel>sell</Rel>
      <Var>merchant</Var>
      <Var>person</Var>
      <Var>object</Var>
    </Atom>
    <Atom>
      <Rel>buy</Rel>
      <Var>person</Var>
      <Var>merchant</Var>
      <Var>object</Var>
    </Atom>
</Implies>
```

- For further information, please see: http://wiki.ruleml.org/index.php/Grailog

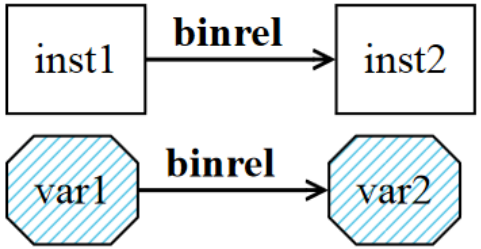
Approach overview

- Initial version of a "Grailog Knowledge-Source Visualizer "
- Mapping from Datalog RuleML/XML,
  to Scalable Vector Graphics (SVG)/XML:
  - subset to subset
  - target Grailog's "fully node copied " normal form
- Realized by using eXtensible Stylesheet Language Transformations 2.0 (XSLT 2.0)
- **Methodology:**
  - create Grailog representation in SVG
  - create transformation from Datalog RuleML XML to SVG Grailog representation

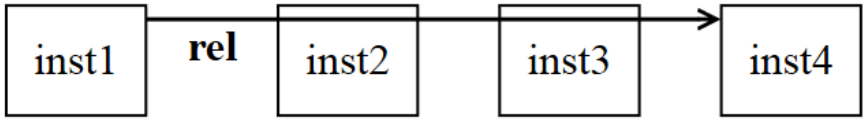


Supported Grailog elements

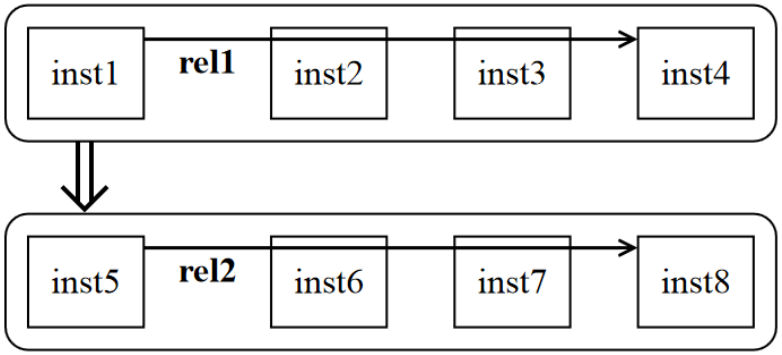
- Binary relations:



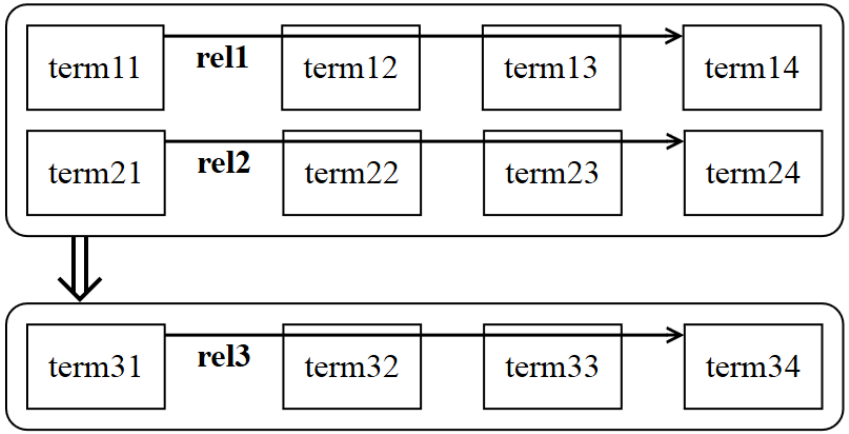
- N-ary relations (n>2):



- Single-premise rules:



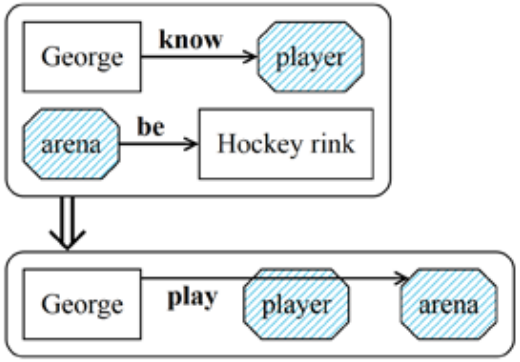
- Multi-premise rules:



Example

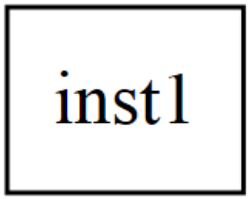
If George knows a player and an arena is a hockey rink, then George plays with the player in the arena.

```
<RuleML>
  <Assert mapClosure="universal">
    <Implies>
      <And>
        <Atom>
          <Rel>know</Rel>
          <Ind>George</Ind>
          <Var>player</Var>
        </Atom>
        <Atom>
          <Rel>be</Rel>
          <Var>arena</Var>
          <Ind>Hockey rink</Ind>
        </Atom>
      </And>
      <Atom>
        <Rel>play</Rel>
        <Ind>George</Ind>
        <Var>player</Var>
        <Var>arena</Var>
      </Atom>
    </Implies>
  </Assert>
</RuleML>
```



Individual constants and variables

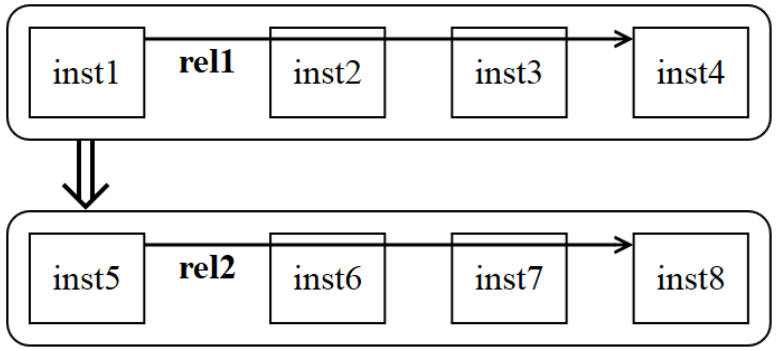
- ***text***
  - *id, x, y*
- ***rect***
  - *id, x, y, rx, ry, height, width, stroke (color, width)*
- ***polygon***
  - *id, points, stroke (color, width), style*
- ***pattern***
  - *id, patternTransform, x, y, width, height*
- ***path***
  - *id, d, stroke (color, width)*



- Lengths of the text elements and positions and scales of the rectangles / polygons were calculated and assigned with JavaScript

Arrows, double-arrows and rectangles with rounded corners

- ***marker***
  - *id, markerWidth, markerHeight*



- The arrow is an SVG path element with an arrow head, which is created with a marker element and another path-element, on top
- The double-arrow is created with a path on a particular track
- Positions and scales of these elements were also calculated and assigned with JavaScript

Basic structure of Grailog KS Viz (1)

- Start the SVG file:
  - definitions (id, version, namespace, marker, pattern)
- **Differentiate** between **rules and facts**:
  - searching for <Implies> , <And> and <Atom>
  - using parent and child relationships and positions for determining the type of the considered atom
    - fact
    - single-premise rule: head, single premise
    - multi-premise rule: head, multi premise
- **Differentiate** between **binary and n-ary (n>2)**:
  - using the number of children of an atom
  - used for determining if a simple arrow or an (intermediate-node-cutting) hyperarc arrow is needed

Basic structure of Grailog KS Viz (2)

- **Create unique variable names** for the used JavaScript code:
  - concatenation of the type and the position in the XML tree
- **Create** the needed **SVG elements**:
  - text (with its value from the source file), rect, polygon, path
- **Create** the needed **JavaScript code**:
  - using <xsl:if> for alternating the code
  - keeping track of the maximum height and width of the viewbox of the resulting SVG file
- End the SVG file

Demo

Conclusion

- Successful implementation of a **reliable, fast and easy-to-use** tool for transforming Datalog RuleML/XML rules and facts into their corresponding SVG/XML Grailog visualization(s)

Future Work

- **Merging** of the individual SVG elements of rules and facts to one graph
  - ✓ User could directly see connectivity
  - — High computational complexity could lead to response-time issues
- **Support** of:
  - ➢ unary relations, (positional-)slotted variants and typed variants
  - ➢ Grailog visualizers for other rule and ontology languages
- Realization of **inverse translators** parsing Grailog SVG/XML diagrams into RuleML/XML trees
  - ✓ Authoring tool that allows users to visually design rule bases in the graphically rendered SVG representation
  - ✓ Could ultimately lead to a complete Grailog IDE

References

(1) Boley, H.: **Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules. Preprint:** http://www.cs.unb.ca/~boley/papers/GrailogVisOntoRules.pdf, visited on May 9th, 2013. To appear: Proc. RuleML 2013, Springer LNCS 8035, July 2013

(2) Boley, H.: **Grailog**. http://wiki.ruleml.org/index.php/Grailog, visited on May 24th, 2013

(3) Boley, H., T. Athan: **RuleML Primer**, August 2012. http://ruleml.org/papers/Primer/RuleMLPrimer2012-08-09/RuleMLPrimer-p0-2012-08-09.html, visited on October 19th, 2012

(4) Boley, H., Athan, T., Paschke, A., Tabet, S., Grosof, B., Bassiliades, N.,Governatori, G., Olken, F., Hirtle, D.: **Schema Specification of Deliberation RuleML Version 1.0**. http://ruleml.org/1.0/, visited on October 19th, 2012

(5) Dahlstroem, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D., Watt, J., Ferraiolo, J., Fujisawa, J., Jackson, D.: **Scalable Vector Graphics (SVG) 1.1 (Second Edition) - W3C Recommendation**, August 2011. http://www.w3.org/TR/2011/REC-SVG11-20110816/, visited on October 19th, 2012

(6) Kay, M.: **XSL Transformations (XSLT) Version 2.0 - W3C Recommendation**, January 2007. http://www.w3.org/TR/2007/REC-xslt20-20070123/, visited on October 19th, 2012

(7) RuleML: The Rule Markup Initiative - Library of Datalog Examples. http://www.ruleml.org/1.0/exa/Datalog, visited on June 12th, 2013

Thank you!

Bonn-Rhein-Sieg University of Applied Sciences

Project website: http://www2.unb.ca/~mkoch/cs6795swt/index.html

(The project website provides detailed documentation of the project, concrete examples, the Grailog KS Viz tool to download and further links and information to all related topics)

Authors:

Martin Koch: martin.koch[AT]unb.ca
(University of New Brunswick)

Sven Schmidt: sven.schmidt[AT]unb.ca
(University of New Brunswick)

Harold Boley: harold.boley[AT]ruleml.org
(University of New Brunswick)

Rainer Herpers: rainer.herpers[AT]h-brs.de
(Bonn-Rhein-Sieg University of Applied Sciences)

Appendix

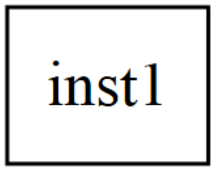
SVG

- Features of SVG used to create the Grailog elements:
  - static graphics
  - text and vector graphic shapes as graphical objects
  - assigning styles and transformations
- All described in XML

JavaScript

- Was essential…
  - to get the lengths of the different texts
  - to scale the elements
  - to position the elements
  - to assign other dynamic features
- The importance of JavaScript was not obvious in the first considerations of the project

Example: individual constant

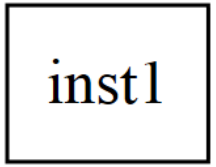



```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">

    <text id="text1" x="50" y="50">inst1</text>
    <rect id="rect1" style="stroke:#000000; fill: none; stroke-width:1;"/>

    <script>
        document.getElementById("rect1").setAttribute("width",
        parseFloat(document.getElementById("text1").getComputedTextLength())
            + 20);
        document.getElementById("rect1").setAttribute("height",40);
        document.getElementById("rect1").setAttribute("x",
            parseFloat(document.getElementById("text1").getAttribute("x")) - 10);
        document.getElementById("rect1").setAttribute("y",
            parseFloat(document.getElementById("text1").getAttribute("y")) - 25);
    </script>

</svg>
```

XSLT

- Features of XSLT used to create the SVG Grailog representation:
  - XSLT Elements
    - stylesheet, output
    - apply-templates, template
    - value-of, for-each
    - variable, if, text
  - XSLT Functions
    - current()
    - concat(string, string, …)
    - not(arg)
    - count(*(item, item, …)*)
    - position()
    - last()

Example: individual constant




```
<xsl:if test="parent::r:Assert">
  <xsl:variable name="countRelations" select="count(preceding-sibling::*) + 1"/>
    <xsl:for-each select="./*">
      <xsl:if test="position()=2"> <!-- First term -->
        <xsl:if test="(ancestor-or-self::r:Ind) or (ancestor-or-self::r:Data)">
          <xsl:element name="rect">
            <xsl:attribute name="id">
              <xsl:value-of select="concat('rect','Relation',$countRelations,position())"/>
            </xsl:attribute>
            <xsl:attribute name="style">
              stroke:#000000; fill: none; stroke-width:1;
            </xsl:attribute>
          </xsl:element>
          <xsl:element name="text">
            <xsl:attribute name="id">
              <xsl:value-of select="concat('text','Relation',$countRelations,position())"/>
            </xsl:attribute>
              <xsl:value-of select="."/>
          </xsl:element>
          <script type="text/javascript" language="JavaScript">
            … (relevant JavaScript code with adjusted variable names etc.) …
          </script>
  … (second term, next terms, last term; each for „Ind“, „Data“, „Var“ and „Rel“) …
</xsl:if>
```

Difficulties and their solutions

- Support of two different source document versions:
  - Datalog RuleML file without and with RuleML namespace and schema
- Support of all recent major webbrowsers:
  - two different methods for computing the length of the text elements
    - ➢ needed for determining the width and position of the graphical elements
  - normal version of the Grailog Visualizer
    - uses *getComputedTextLength()*
      - ➢ does not work in Firefox
  - monospaced font version of the Grailog Visualizer
    - uses *XMLSerializer()* and *serializeToString()*
      - ➢ does work in Firefox, but not in Safari or Chrome

Supported Web Browsers

- Supported (green) and unsupported (red) Web browsers of the normal version of Grailog KS Viz :

Representation in Browser	Firefox (16.0.2)	Google Chrome (23.0.1271.64 m)	Internet Explorer (9.0.8112.16421)	Opera (12.10)	Safari (5.1.7)
After on-the-fly transformation	not supported	not supported	supported	supported	supported
Retrospectively saved as SVG	supported	supported	supported	supported	supported

- Supported (green) and unsupported (red) Web browsers of the monospaced font version of Grailog KS Viz :

Representation in Browser	Firefox (16.0.2)	Google Chrome (23.0.1271.64 m)	Internet Explorer (9.0.8112.16421)	Opera (12.10)	Safari (5.1.7)
After on-the-fly transformation	supported	not supported	supported	supported	not supported
Retrospectively saved as SVG	supported	not supported	supported	supported	not supported