SPLASH Conference Version

UNDERSTANDING PROGRAM VISUALIZATIONS IN THE WILD

Olohi John, Joel Castro

BACKGROUND

Research Question: What makes program visualizations effective?

Limitations of previous work:

- Little focus on scalability
  - Pivotal to our predictive theory about visualization interpretability
- Focus on surface-level features
  - Features that do not get at generalizable underlying properties
- Small sample of visualizations
  - Spanning across a single or few domains

Table 4: Assessing the "Form" of the Systems — JAIN *ET AL.*

(3) FORM	*Color*	*Dimensions*	*Animations*	*Sound*	*Granularity*	*Multiple Views*	*Program Synchronization*

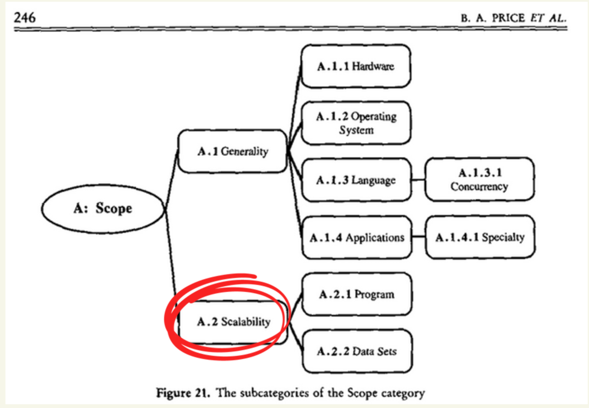


Figure 21. The subcategories of the Scope category

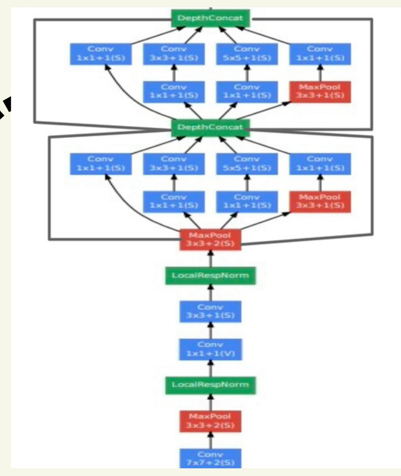
Why ***scalability*** is important in programming visualizations:

- Debugging usually involves large inputs
- Visualizations must be able to handle significantly large inputs to be effective

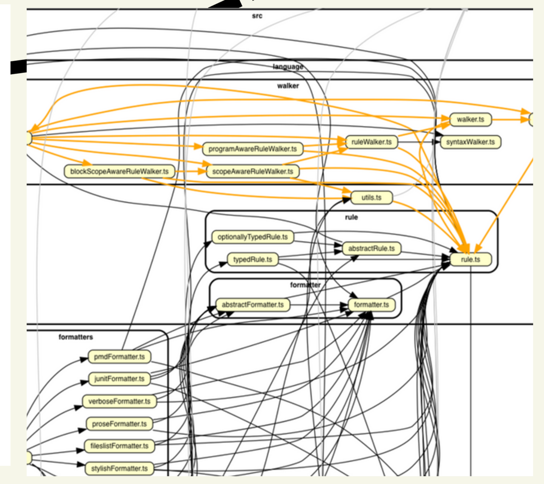
The effectiveness of visualizations is determined by how well they can scale for significantly large inputs.
Goal: Measure scalability, and thus, effectiveness of program visualizations

.

Within program visualizations:



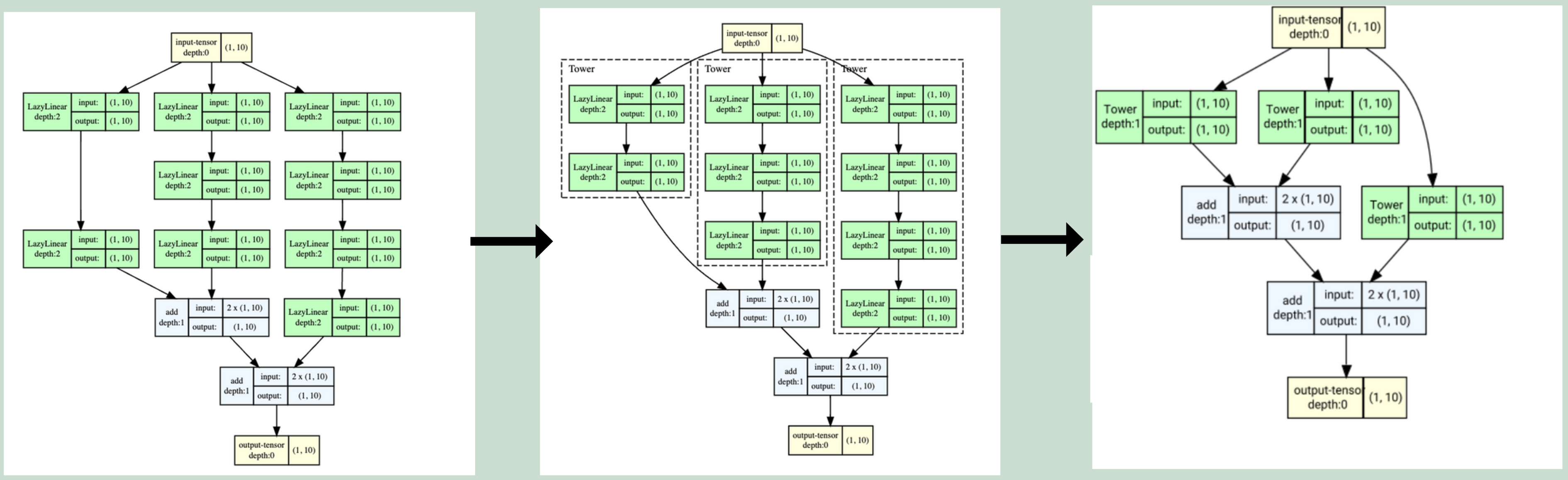
Why **does** this visualization **scale well**?



Why **doesn't** this visualization **scale well**?

THE THEORY OF SLICEABILITY

Sliceability: The ease with which the local pieces that make up a visualization can be identified and used to understand the whole structure. The theory aims to understand visualizations as compositions of simpler structures.



HYPOTHESIS

"Scalability ~ Sliceability"

The sliceability of a visualization is an approximate measure of its scalability. Generally, the higher the complexity of an analyzed visualization's structure, the lower the visualization's sliceability and, consequently, the less its scalability.

RESEARCH PHASES

PHASE 1

Develop visualizations' complexity hierarchy and descriptive taxonomy

PHASE 2

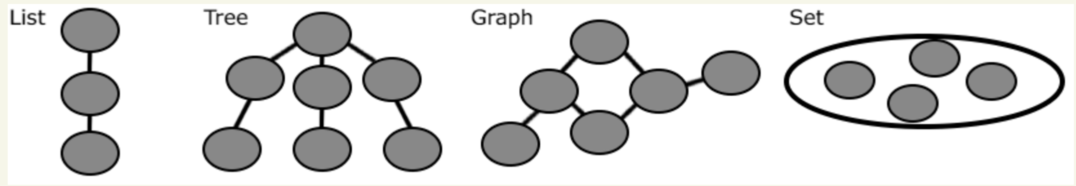
Use codebook of taxonomy from phase 1 to answer the question: "Are visualizations composed of simpler basic stuctures more scalable, and thus, more effective?"

PHASE 1

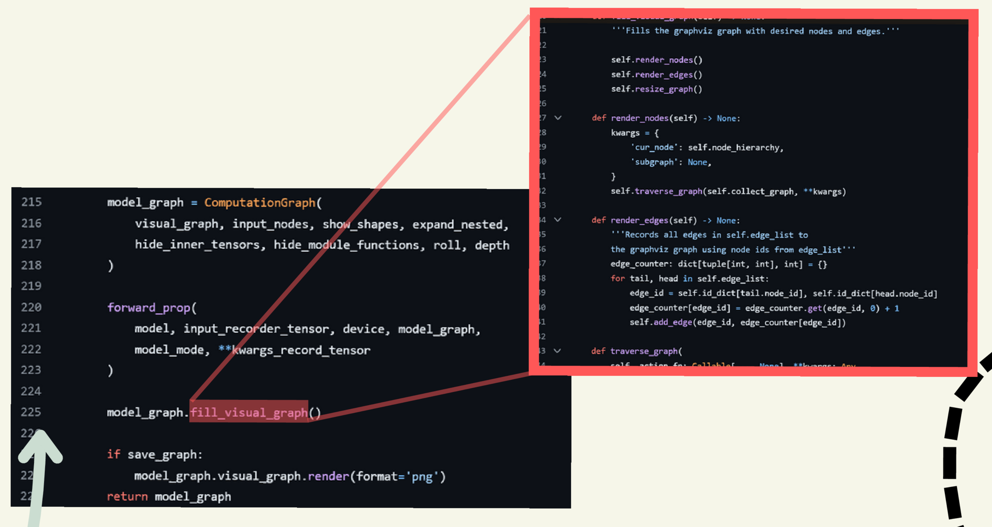
Sub-hypothesis: It is possible to group all programming visualizations into a finite number of categories (codebook).

METHODOLOGY

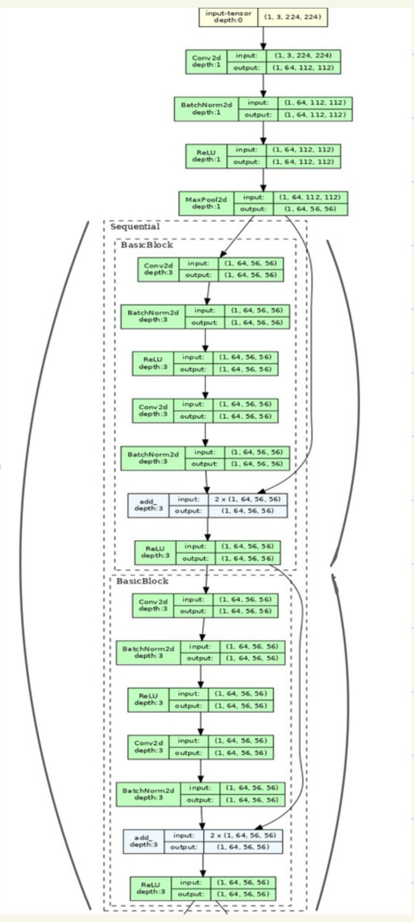
Studied visualizations across 2 levels of abstraction: Literal Visualization and Abstract interpretation.



- **Collected 150 (mechanically generated) examples** across **7 domains**: Machine Learning, Graphics, Web Dev, Game Dev, Video, Animation, and Compilers.
  - Verified they were mechanically generated by looking at their source code
- **Categorically coded** examples as some composition of **4 primary visualization structures**: *sets, lists, trees, and graphs.*
- **Incorporated compositional operators** into our codebook to describe the relationship between parts of a visualization that had different basic structures. These included:
  - Sequential operator (→)
  - Parallel operator (+)
- Codebook iteratively updated to fit new data until reaching a point of saturation
- Operators allowed to describe the literal visualization and abstract interpretation uniformly, growing them towards each other, thus creating a comprehensive codebook describing both.
  - Examples:



__Description__: graphviz.Digraph object populated with module hierarchy, torch_functions, shapes, and tensor data recorded during a forward prop. Note: TensorNodes saved in NodeContainer(s); graph



CONCLUSION/RESULTS

- Codebook got to a level of saturation after 80 examples and could describe every further visualization found
- In many cases, structures composed of simpler primary compositions were preferred when possible--- promising of sliceability
- When used in the restructuring of visualizations, sliceability's predictive power can lend itself to constructive applications:
  - Education
  - Debugging

FUTURE WORK: PHASE 2

- Design a game theory experiment to determine if debuggers debug more efficiently with visualizations composed of simpler basic structures
- Interview creators of debugging tools to glean into inherent applications of sliceability employed by same

REUSE REU Version

UNDERSTANDING PROGRAM VISUALIZATIONS IN THE WILD

Olohi John, Joel Castro

BACKGROUND

Research Question: What makes program visualizations effective?

Limitations of previous work:

- Little focus on scalability
  - Pivotal to our predictive theory about visualization interpretability
- Focus on surface-level features
  - Features that do not get at generalizable underlying properties
- Small sample of visualizations
  - Spanning across a single or few domains

Table 4: Assessing the "Form" of the Systems — JAIN *ET AL.*

(3) FORM	Color	Dimensions	Animations	Sound	Granularity	Multiple Views	Program Synchronization

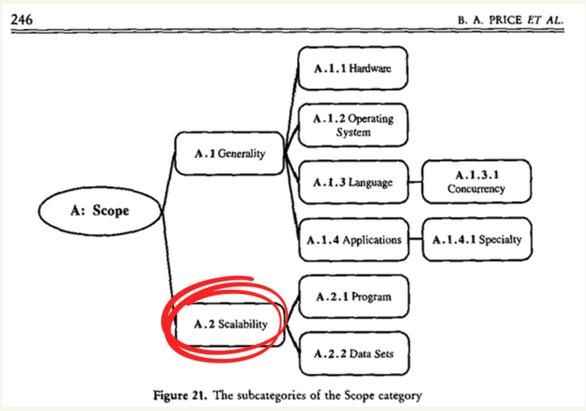


Figure 21. The subcategories of the Scope category

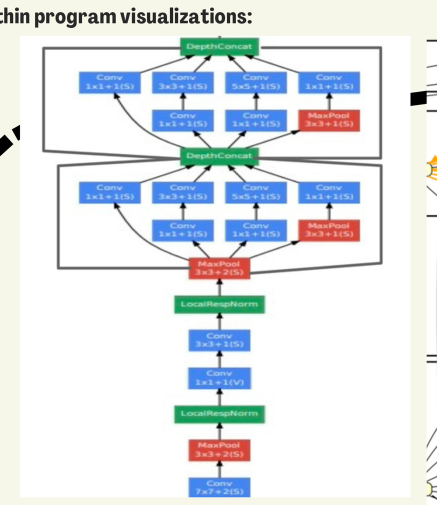
Why ***scalability*** is important in programming visualizations:

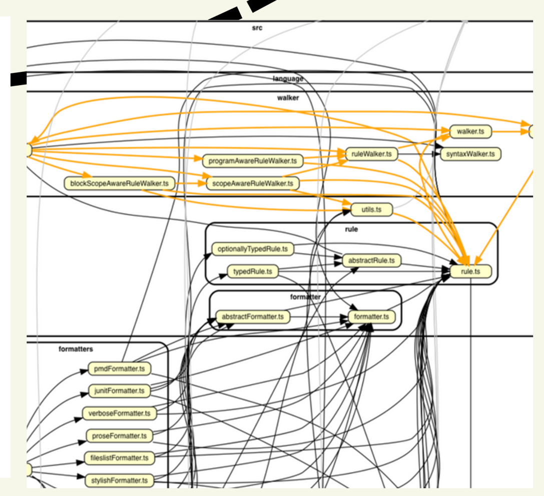
- Debugging usually involves large inputs
- Visualizations must be able to handle significantly large inputs to be effective

The effectiveness of visualizations is determined by how well they can scale for significantly large inputs.

Goal: Measure scalability, and thus, effectiveness of program visualizations

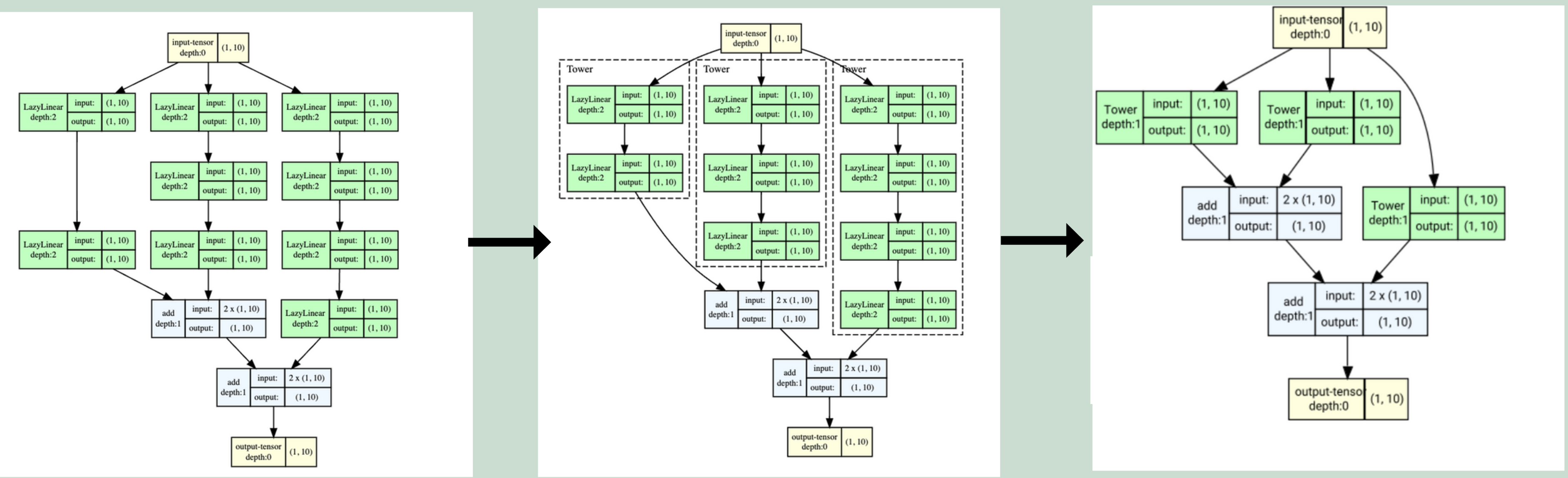
Within program visualizations:


Why **does** this visualization **scale well**?


Why **doesn't** this visualization **scale well**?

THE THEORY OF SLICEABILITY

Sliceability: The ease with which the local pieces that make up a visualization can be identified and used to understand the whole structure. The theory aims to understand visualizations as compositions of simpler structures.

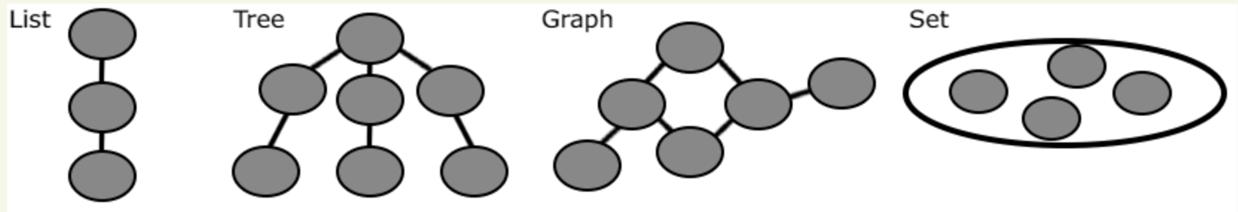


HYPOTHESIS

"Scalability ~ Sliceability"

The sliceability of a visualization is an approximate measure of its scalability. Generally, the higher the complexity of an analyzed visualization's structure, the lower the visualization's sliceability and, consequently, the less its scalability.

METHODOLOGY



Visualizations are studied along **3 levels of abstraction**: *Underlying program, literal visualization, and cognitive interpretation.*

- **Collected 150 examples** across **7 domains:** *Machine Learning, Graphics, Web Dev, Game Dev, Video, Animation, and Compilers.*
- Examples are **categorically coded** as some composition of **4 primary visualization structures**: *sets, lists, trees, and graphs.*
- **Analyze** changes in abstraction across each level, looking for patterns that show markers of sliceability.

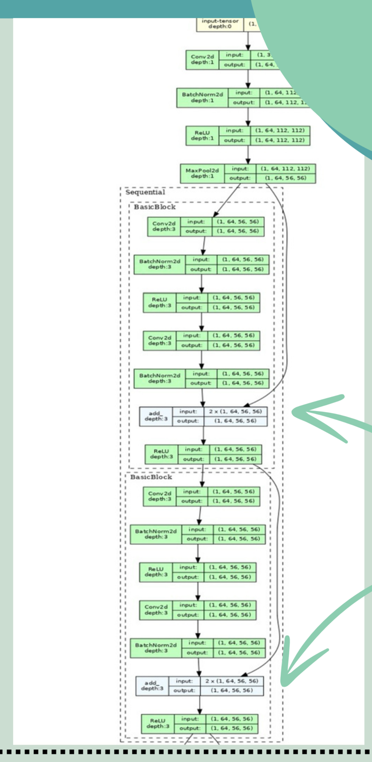
PROGRAM | VISUALIZATION | HUMAN

Codes $C ::= \texttt{List of } C \mid \texttt{Tree of } C \mid \ldots \mid C + C \mid C \to C$

e.g.

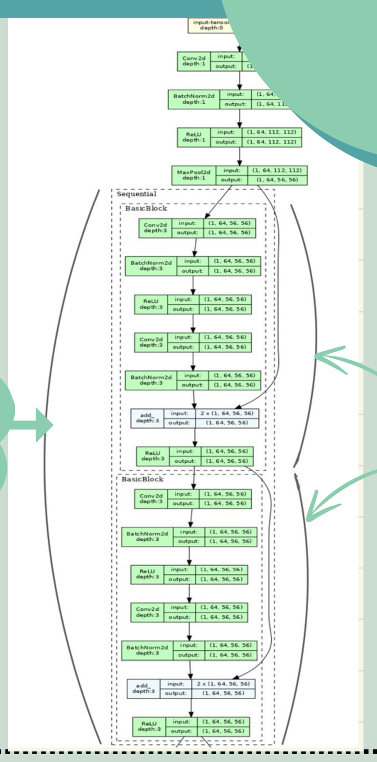
Torchview

Description: graphviz.Digraph object populated with module hierarchy, torch_functions, shapes, and tensor data recorded during a forward prop. Note: TensorNodes saved in NodeContainer(s); graph



A Directed Acyclic Graph of a feed-forward RNN network-- visualized by torchview

Boxes to highlight nodes under the same segment



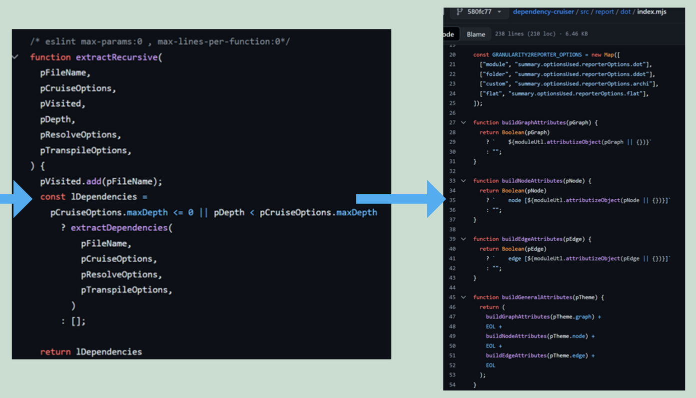
A single node in parent list [contains another list---nested lists]

Two nodes nested into a single node in parent list

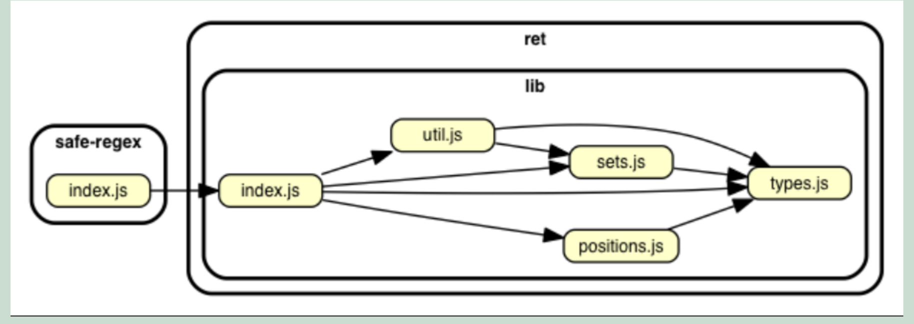
Can be read as a single parent list with some nodes containing nested lists

Dependency-Cruiser

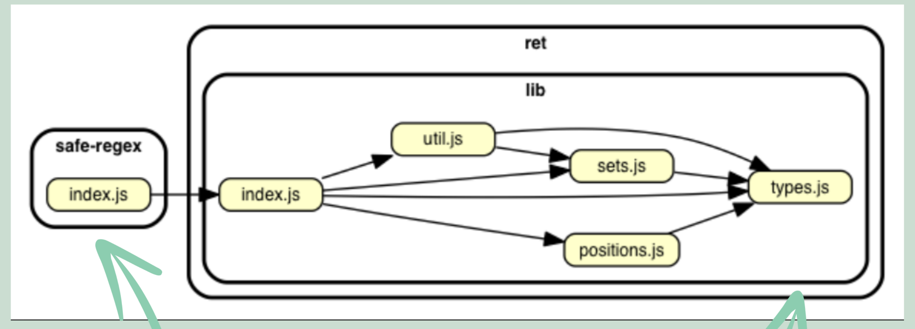
PROJECT CODEBASE



Description: buildGraphAttributes, buildNodeAttributes, and buildEdgeAttributes are used to generate DOT snippets from nested object structures (that were generated by dependency-cruiser) representing modules, their dependencies, and additional information; graph.



A Directed Acyclic Graph (DAG)



Visualized as a list of two elements, one of which is a DAG

Node 1: contains a single element (index.js)

Node 2: contains a DAG (Delve into details to know more)

PRELIMINARY RESULTS

Among coded examples...

- About *55%* of visualizations show some form of reduction in complexity across the **vis-human** gap.
- Structure transformation across **program-vis** gap (Program - Vis: *count*):
  - Graphs - Graphs/DAGs: *2*; Trees: *1*
  - Tables - Graphs: *1*
  - Dictionaries - Graphs: *1*; Trees: *3*; Lists: *1*
  - Lists - Trees: *2*; Lists: *1*

CONCLUSION

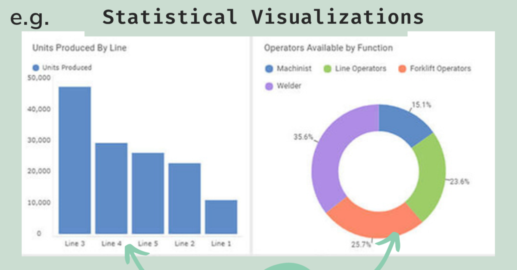
- Preliminary results are promising of sliceability
- While used in the restructuring of visualizations, sliceability's predictive power can lend itself to constructive applications:
  - Education
  - Debugging

FUTURE WORK

- Eye Tracking Study
  - Do reading patterns corroborate coded human-cognitive models?
- Visualization tool developer interview study
  - Is sliceability implicitly considered in the development of visualization tools?

SLICEABILITY OUTSIDE OF PROGRAM VISUALIZATIONS

e.g. Statistical Visualizations



Designed with a guarantee to "scale well" for any arbitrarily large set of data

Mathematical Visualizations

Other forms of mathematical visualizations become really difficult to parse with large inputs---do not scale nicely