DISV: Domain Independent Semantic Validation

Ashish Kumar[1], Bill Harris[2] and Gang Tan[1]

[1]Penn State University

[2]Galois, Inc.

May 25, 2023

Motivation

- Semantic Properties on data files are seldom validated by data receivers as they are generally limited to syntactic validation. Many such semantic properties can cause a buggy data file to crash, enter infinite recursion or have other undesired consequences.
- For example, in SVG files, objects can be attributed with 'use' and 'symbol' tags: An object with the 'use' tag can refer to a second object with a 'use' or 'symbol' tag via an href-link.

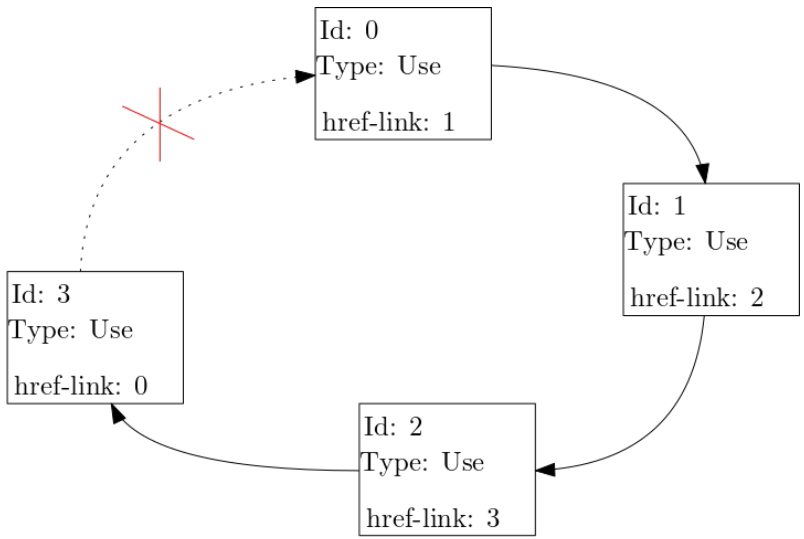

Figure: Objects from a buggy SVG data file.

Motivation 2

Then we **must validate that a 'use' reference isn't cyclic**, which may cause the data receiver to enter an infinite recursion, and therefore crash.

- **Goal:** Our goal is to design a language framework expressive enough to capture semantic properties of practical interest to validate.
- Compared to syntactic validation, validating semantic properties is more challenging, as semantic properties are often global properties - for instance, validating the use-use circularity described above requires checking cycles in the underlying use-use graph of an SVG file.
- We note that many of the global semantic properties of interest are actually graph properties, and design a framework to validate such properties.

DISV: Our Methodology

DISV has two major components:

1. Specification Grammar: It comprises of a
   (i) Graph specification specifying which objects and links from the input file form a graph
   (ii) Semantic definitions associating attributes to graph nodes and other auxiliary definitions
   (iii) Semantic properties to validate.
2. Automated Property Validation: Takes an input data file and a property specification file and outputs a boolean value indicating if the data file satisfies the property or not.

System Architecture of DISV

- Front End transforms the input file to JSON IR file.
- In the IR, objects are stored as dictionaries of key-value pairs and are given a unique ID. Links between objects are made by using the 'Ref' key of each object.

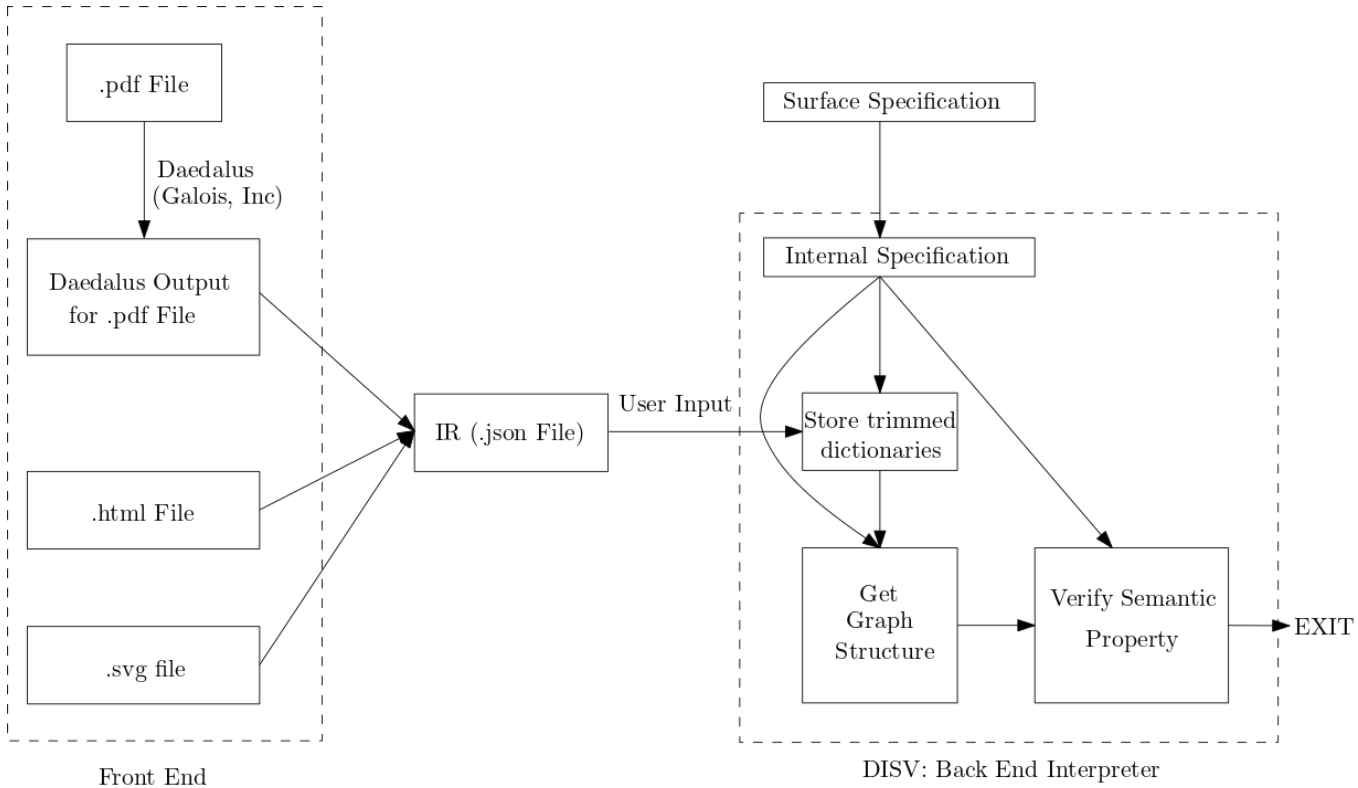

Figure: Internal Architecture of DISV

System Architecture of DISV 2

- DISV first converts the specification file to an internal form, which it uses along with the JSON IR file to store all objects in memory with only the relevant keys.
- It then uses this list of objects to construct the underlying graph, and finally recursively validates the semantic properties on this graph in a top-down order.

Specification Grammar

We illustrate our specification grammar by example using the page-tree inheritance property of PDF documents. The Specification Grammar has 3 components:

- **Graph Specification:** A PDF file may contain a set of objects which together form a so-called page tree: Each non-root node object in a page tree is either a "Page" or "Pages" type Object and the root is defined by the "Catalog" Object.

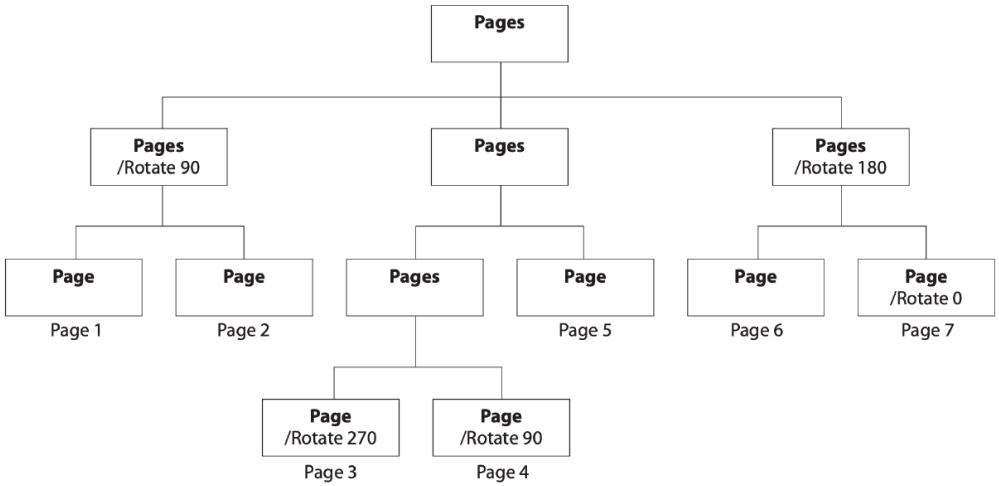

Figure: A subtree of the page tree for a pdf file

Specification Grammar 2

"Page" and "Pages" objects may have several keys like /Rotate, /Mediabox etc.

- The specification code to specify a page tree is:

```
1  Tree:
2  Root is unique d satisfying d.<"Id">
3  = d1.<"Pages"> where unique d1 satisfies
4  d1.<"Type"> = "Catalog".
5
6  forall d in Tree, d0 is Child(d) where
7  d0.<"Id"> in d.<"Kids"> and d0.<"Type">
8  in ["Page", "Pages"].
9
```

Figure: Page Tree Specification

Specification Grammar 3

- **Semantic Definition Specification:** Semantic Definitions are used to define attributes[1] and/or quantifier sets[2].
- For example, "Resources" and "MediaBox", are two keys of "Page" and "Pages" objects designated as both “Required” and “Inheritable” by the PDF specification.
- This means that for each leaf object in the page tree, the "Resources" and "Mediabox" key must either be defined by the leaf object or by one of its ancestors.

[1] synthesized or inherited.

[2] node or edge quantifier sets.

Specification Grammar 4

- Hence, we define an inherited boolean attribute for each key for each node which stores if the key is defined by some ancestor of the current node or not. The specification is as follows:

```
Semantic Definitions:
forall d in Root, d.<is_Resources_defined>
= iskeydefined(d, "Resources").

forall d in Root, d.<is_MediaBox_defined> =
iskeydefined(d, "MediaBox").

forall d1 in Tree, forall d2 in Child(d1),
d2.<is_Resources_defined> =
d1.<is_Resources_defined> or
iskeydefined(d2, "Resources").

forall d1 in Tree, forall d2 in Child(d1),
d2.<is_MediaBox_defined> =
d1.<is_MediaBox_defined> or
iskeydefined(d2, "MediaBox").
```

Figure: Page Tree Attribute Specification

Specification Grammar 5

- **Semantic Property Specification:** We can then state the property to be validated as a first order logic formula over the quantifier sets defined in the semantic definitions, and/or use the predefined attributes.
- For the above example, we want to verify that the boolean attribute assigned to each leaf node holds the value "True".

```
Semantic Property:
forall d in Leaves, d.<is_Resources_defined>
= True.

forall d in Leaves, d.<is_MediaBox_defined>
= True.
```

Figure: Semantic Property Specification

Specification File Example

Putting it all together, the spec file for the Page Tree Inheritance Property looks like:

```
Tree:
Root is unique d satisfying d.<"Id">
= d1.<"Pages"> where unique d1 satisfies
d1.<"Type"> = "Catalog".

forall d in Tree, d0 is Child(d) where
d0.<"Id"> in d.<"Kids"> and d0.<"Type">
in ["Page", "Pages"].

Semantic Definitions:
forall d in Root, d.<is_Resources_defined>
= iskeydefined(d, "Resources").

forall d in Root, d.<is_MediaBox_defined> =
iskeydefined(d, "MediaBox").

forall d1 in Tree, forall d2 in Child(d1),
d2.<is_Resources_defined> =
d1.<is_Resources_defined> or
iskeydefined(d2, "Resources").

forall d1 in Tree, forall d2 in Child(d1),
d2.<is_MediaBox_defined> =
d1.<is_MediaBox_defined> or
iskeydefined(d2, "MediaBox").

Semantic Property:
forall d in Leaves, d.<is_Resources_defined>
= True.

forall d in Leaves, d.<is_MediaBox_defined>
= True.
```

Figure: Page Tree Inheritance Property Specification

Automated Property Validation

Let

$$n \leftarrow \text{No. of graph nodes}$$
$$e \leftarrow \text{No. of graph edges}$$
$$D \leftarrow \text{No. of IR dictionaries}$$
$$q \leftarrow \text{Max Quantifier Depth of semantic properties}$$

Then our property validator has 3 components:

1. **Compute Graph**: Compute the graph structure from the list of objects in $O(nD)$ time.
2. **Compute Attributes**: Assuming that there are atmost constant number of properties per graph node, we can compute the attribute values for all objects of the graph in $O(n)$ time.
3. **Property Checking**: We first (i) compute the node or edge set over which the semantic properties quantify over in $O(n^q + n^2)$ time and then (ii) validate these properties in $O(n^q + e^q)$ time by using a 'quantifier map' which maps quantified variables to the current instantiated values.

We implement DISV in $\sim$ 4600 LOC in C++.

Case Studies

We validate 9 expressive semantic properties on data files from 3 different data formats (SVG, HTML and PDF), which we tabulate below:

Property Name	LOC	Graph Type	Semantic Definition Type	Quantifier Type
PDF page-tree inheritance	32	Tree	Inherited Attributes	Universal
PDF faulty permissions attack	38	Graph	Node + Edge Quantifiers	Universal
HTML head element referenced at most once	20	Tree	Node + Edge Quantifiers	Universal
HTML unique refstrings	15	Tree	Synthesized Attributes	Universal
HTML nesting order of table elements	17	Tree	Node Quantifiers	Universal + Existential
HTML paragraphs cannot be nested	13	Forest	–	Universal
SVG 'Title' is first child	17	Ordered Tree	Node Quantifers	Universal
SVG references in 'defs'	14	Tree	Edge Quantifiers	Universal
SVG no use-use circularity	15	Tree	Node + Edge Quantifiers	Universal

Table: Summary of Properties Validated.

Security Applications: PDF Attack Detection

- Our specification grammar is expressible enough to validate the No Faulty Permission Level attack property in PDF documents, which we describe below.
- A certified PDF defines permissions that allow certain changes to the file. Certifiers choose between 3 different permission levels : $P1 \overset{\text{stricter}}{>} P2 \overset{\text{stricter}}{>} P3$.
  - P1: No modifications allowed.
  - P2: Only filling out form fields and digitally signing is allowed.
  - P3: All incremental updates are allowed.
- If an attacker forces a disallowed modification to a certified document, the certification breaks.
- Recent studies[3] demonstrated attacks that allowed modifying a certified PDF file with changes not within the permission level, and avoided detection by some PDF readers.
- The no faulty permission level attack property specifies that only those changes are allowed in a pdf file which are permissible by it's permission levels.

[3] Shadow Attacks: Hiding and Replacing Content in Signed PDFs. by Mainka et al.

Validation Time Results

Time is $O(n^q + e^q + n^2 + nD)$, as demonstrated by experiments.

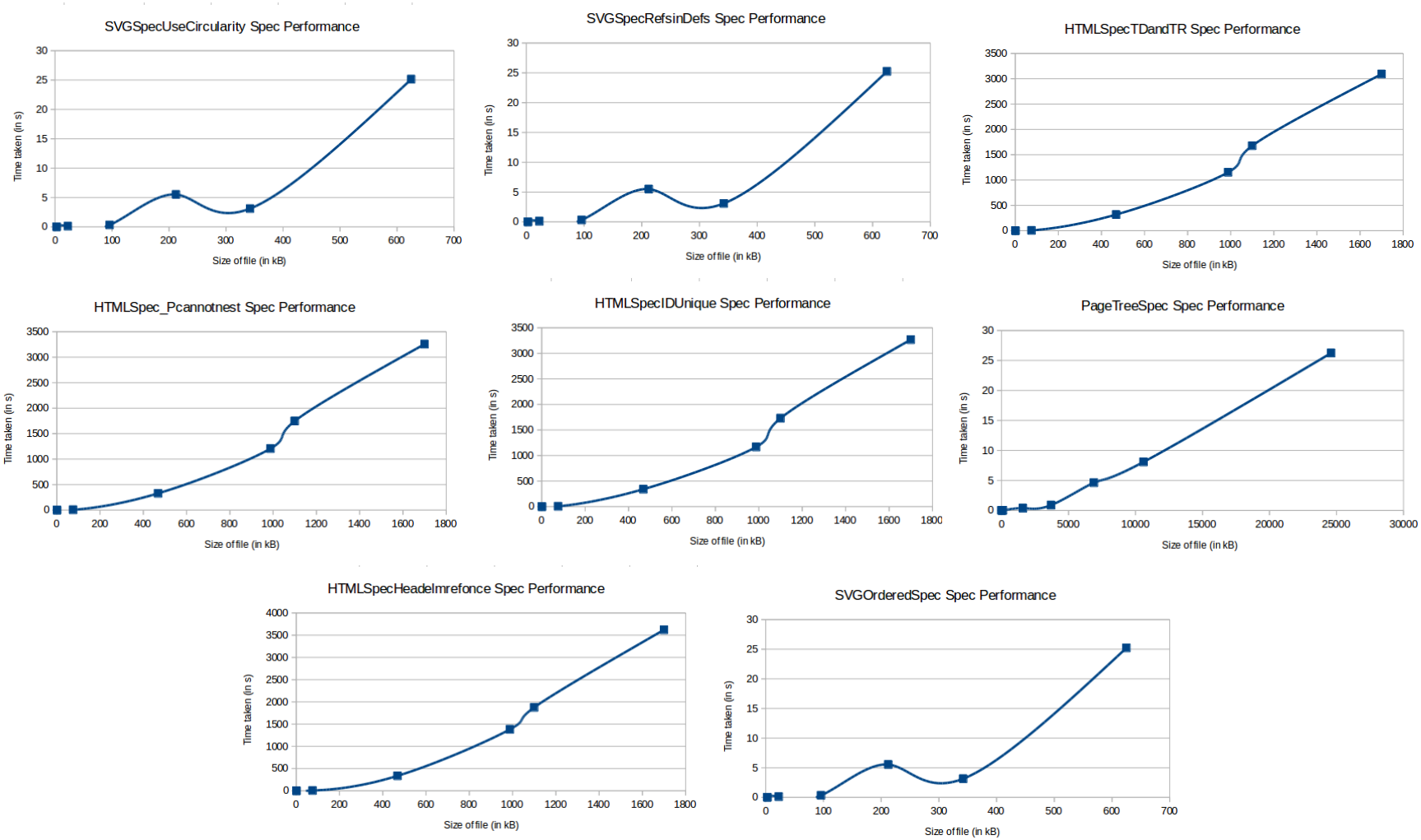

Figure: Time Performance Plots for 8 case studies

Future Directions

We can extend this research in the following future directions:

- Use the DISV model in the field of security to detect security attacks such as EAA and SSA attacks on PDF documents, which break the integrity of a PDF documents.
- Extend the graph logic specification used by DISV to allow for second order graph logic properties (MSO1 or MSO2).

Thank You.

Questions?