The geometry of syntax and semantics for directed file transformations

Steve Huntsman
BAE Systems FAST Labs
Arlington, Virginia, USA
steve.huntsman@baesystems.com

Michael Robinson
American University
Washington, DC, USA
michaelr@american.edu

Abstract—We introduce a conceptual framework that associates syntax and semantics with vertical and horizontal directions in principal bundles and related constructions. This notion of geometry corresponds to a mechanism for performing goal-directed file transformations such as "eliminate unsafe syntax" and suggests various engineering practices.

Index Terms—bundle, fibration, lens, language-theoretical security

I. INTRODUCTION

There is a long tradition of considering syntax and semantics as dual, e.g., with various type theories and sorts of categories respectively inhabiting these roles [1]. Meanwhile, there is an even longer tradition of considering algebra and geometry as (Isbell, "sheafily," or "spectrally") dual, as manifested in the notions of Stone duality between Boolean algebras and Stone spaces; or Gel'fand duality between commutative C*-algebras and locally compact Hausdorff spaces; or duality between commutative rings and affine schemes [2]. [1] However, it can be argued that "the duality between syntax and semantics is really a manifestation of that between algebra and geometry" [5].

Here, we introduce a conceptual framework that simultaneously embraces and extends this perspective: syntax transformations are viewed as forming a group (or more generally, a groupoid), and semantically distinct (representations of) files form a "base space." [2] These constructs are unified in the structure of a bundle (or more generally, a fibration), and a goal-directed transformation of files imbues this structure with a notion of geometry that connects syntax and semantics.

Perhaps the most contentious part of this conceptual framework is the notion that syntax transformations are (or ought to be) invertible. On one hand, the requirement of vertical invertibility is *imposed* by mathematics once one commits to the model of a principal bundle or, more generally, a category fibered in groupoids (see §II-A or §III-B, respectively) for providing an arena where geometry can direct file transformations (or conversely, where directed file transformations can be considered as defining a geometry). [3] On the other hand, the requirement is *justified* by performing transformations with ancillary memory: i.e., annotating any transformations with inline comments or external ancillae that are invertible (into nothingness) by (de)construction. [4]

For example, consider a PDF file [12]. As §2 of [13] points out, the program `pdftk` will produce a valid PDF from a malformed file with an abbreviated header and missing data about both the length of the page content stream and the cross-reference table. At the same time, the program will manipulate abstractly irrelevant details of concrete syntax such as whitespace. Insofar as this process would be instantiated in our framework, and notwithstanding the fact that the original malformed and valid PDF files are presented in the same syntactic representation, the overall transformation itself should be considered as the composition of one explicitly invertible "vertical"/"syntactic" and one not explicitly invertible "horizontal"/"semantic" transformation. The former transformation merely inserts comments detailing the malformations (including tags that detail the comments' provenance and hence facilitate their removal or "inversion"), whereas the latter transformation actually manipulates the header, inserts the missing data, and performs some collateral manipulation of concrete syntax.

A simpler and completely explicit example in the same vein would be a PDF file with invalid terminal object delimiters, e.g. `objend` instead of `endobj`. [5] Here, the transformation

[1]Of particular interest for our present considerations is the duality between crossed product C*-algebras and principal bundles [3], and more generally dualities that embody noncommutative geometry [4].

[2]The *homotopy hypothesis/theorem* that "spaces are ∞-groupoids" [6] can and should be viewed in this context as giving a geometrical interpretation for this base space, rather than an algebraic interpretation for paths on this base space.

[3]The models we consider might admit further generalization to circumvent this invertibility requirement via monoidal fibrations [7]. However, the relative simplicity of the considerations of §II could (and probably should) be taken as a hint to take the invertibility requirement seriously. Indeed, exposing the constraints imposed by a mathematical model is the primary benefit of an exercise such as ours.

[4]This is essentially the reverse of the tactic underlying *logically reversible computation*, for which see, e.g. [8]–[11]. In reversible computation, one accepts ancillary inputs that are transformed into "garbage": here, we accept ancillary outputs that can be reverse-transformed into nothingness.

[5]We thank P. Wyatt for noting that this delimiter is invalid, though it can be seen on occasion "in the wild."

```
int i;
for (i=0; i<10; i++)
{
  z+=i;
}
```

```
int n=0;
while (n<10) {
    x+=n;
    n++;
}
```

(a) A `for` loop.   (b) A `while` loop.

Fig. 1: (Adapted from [14].) Two semantically identical C loops. The more versatile `while` construct is better suited for a normal form of C (and for a bootstrapping compiler).

```
        jmp @5
@4:
        jmp @9
@8:
        jne @19
        jmp @10
@19:
        jmp @14
@13:@14:
        jg  @13
@9:@10:
        jge @20
        jmp @8
@5:@20:
        jge @21
        jmp @4
@21:
```

```
START; S
do while b
  S
  do while b
    if b
      S
      do while b
        S
      enddo
    endif
    S
  enddo
  S
enddo; HALT
```

(b) Algorithmically restructuring control flow from (a) to turn backward branches into `while` loops. Each S is its own statement/subroutine; each b is its own Boolean predicate.

(a) Branches and labels from assembler listing of Gauss-Jordan elimination code.

Fig. 2: (Adapted from [15].) Decompiling into normal form.

sequence would be something like

$$
\begin{aligned}
\texttt{objend} &\Rightarrow^{vert/}_{synt} && \texttt{objend \% objend -> endobj} \\
&\Rightarrow^{horz/}_{sema} && \texttt{endobj \% objend -> endobj}
\end{aligned}
$$

which actually specifies how to perform the inverse syntactic transformation. (Recall that comments in PDFs are initialized by `%` and terminated by end-of-line markers outside of strings or inside of content streams (see §7.2.4 of [12]).)

For lack of a better term, we call this sort of bookkeeping *sugar-neutral*: i.e., we view syntactic sugar such as variable forms for a token in a file or a `case` or `elseif` statement in source code as something that should be accomodated and potentially preserved in a file at the *outset* of a transformation, but that should not be introduced *during* a transformation. Taking (de)compilation as an example and insisting on *normal forms* as in Figures 1 and 2, we can in principle carry along the concrete syntax of a file through quite complex transformations to detail how to transform between semantically equivalent files with different concrete syntax in any representation (see §II-B3).

II. BUNDLES

A. Principal bundles

The geometry of *principal bundles* [16], [17] turns out to provide a useful conceptual framework for reasoning about and manipulating syntax and semantics. Given a "horizontal base space" X corresponding to some particular lossless representation of a language/file format (e.g., strings/words, concrete syntax trees [CSTs], etc.) and a "vertical" group G of invertible syntactic transformations, we consider an object akin to a *connection* in a *principal bundle* $P(X, G)$ as depicted in Figure 3.
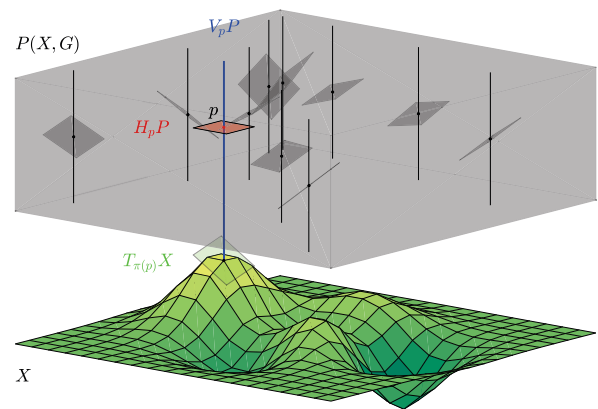

Fig. 3: A principal G-bundle $P(X, G)$ provides a natural arena for geometry realized through a *connection*, i.e., a smooth direct sum decomposition $T_pP = V_pP \oplus H_pP$ of tangent spaces into "vertical" and "horizontal" components that is *equivariant* under an action of G. In the figure, $\pi : P \to X$ is the bundle projection map.

In the present context, we can think of a connection as a recipe for directing file transformations in terms of vertical syntactic transformations and horizontal semantic transformations, or in a complementary sense to connect the spaces of lifts of nearby files. Since syntactic file transformations can be thought of as abstractions of semantic file transformations [18], the connection (in particular, its equivariance property) informs the distinction between syntax and semantics.

In other words, the common notion of lifting a file to a different representation is consistent with the usage of the term "lift" in mathematical use: a directed semantic modification corresponds to the terminal point of a (presumably fairly "short") path on the base space X that can be lifted to a path in the bundle $P(X, G)$ using a connection. In this differential-geometrical analogy, the goal of semantic file transformations is manifested by side information (e.g., format specifications, corpora, programs that nominally accept such files as inputs, etc.) that define a geometry, and a directed file transformation

itself corresponds to *parallel transport* along a vector field. [6]

An explicit objective function that gives rise to the geometry can be specified in practice by, e.g., a dissimilarity measure between the original and perturbed files [21], or between traces or indeed any other suitable artifacts (see §III-C). In this analogy, "infinitesimal" transformations are composed, and the transformation approach of §II-B draws from this conceptual framework.

Crucially, adding or removing sugar in syntactic transformations affects the group structure required to define a notional connection (though as we discuss later, groupoids can suffice for algebraic purposes and a connection *per se* is unnecessary since geometry can be constructed via alternative means). We avoid this by choosing representative semantics-preserving transformations based on *normal forms*. This reflects the observation of [22] that "sugaring is a compilation process." For programming languages, such normal forms can be enforced by, e.g., restructuring code so that backward branches become `while` loops [15] or in reference to a particular configuration of some machine learning algorithm such as [14] (i.e., we fix a trained implementation once and for all).

Under this analogy, in the file transformation process, points in $P(X, G)$ correspond to CSTs and G corresponds to semantics-preserving (invertible) transformations on CSTs. The equivalence class of CSTs that correspond to a given abstract syntax tree (AST) carries both group-theoretical and language-security theoretical significance, as it indicates redundancy in a format.

B. File transformations

In order to indicate the substantive nature of the analogy in §II-A, we proceed to sketch some examples of directed file transformations in §II-B1, detail the first of these in §II-B2, and in §II-B3 outline a generic framework for implementing directed file transformations that separates concerns between syntax and semantics.

1) Examples of directed file transformations: [7]

- Toy example 1 (detailed and elaborated upon in §II-B2):
  - $X = \{0, 32, \ldots, 126\}^*$ (i.e., ASCII strings of `NULL`s and printable characters) endowed with edit distance;
  - G = cyclic shifts on individual characters;
  - Goal: remove `NULL`s and punctuation and make lowercase.

- Toy example 2:
  - $X = \{97, \ldots, 122\}^*$ (i.e., lowercase alphabetical ASCII strings) endowed with edit distance;
  - $G = $ `rot13` (or $G = 97 + 13 \cdot \mathbb{Z}/2\mathbb{Z}$);

---

[6]It bears mentioning that the relevant mathematics can be adapted to the cases where X and/or G are discrete: these are respectively exemplified by lattice gauge theory [19] and/or discrete gauge groups in physics [20]. The essential idea is obvious: such discrete spaces are embedded in continuous ones that they approximate.

[7]There may be relatively simple and useful examples obtainable by, e.g., manipulating casts/encodings of simple datatypes.

  - Goal: minimize the number of character edits plus the Hamming distance between the eventual result and the `rot13` of its reversal. [8]

- **Language-theoretical security** [23]:
  - $X = $ files in a fixed format endowed with a distance on, e.g. ASTs (see §III-C1);
  - $G = $ sugar-neutral bidirectional transformations;
  - Goal: eliminate syntax that does not conform to reasonably specified deterministic grammar (possibly including syntax features such as [24], [25]).

- Feature elimination in C:
  - $X = $ C source files with distance defined on ASTs;
  - $G = $ sugar-neutral translations (between, e.g., source, LLVM, etc.);
  - Goal: parsimoniously eliminate a particular type of syntactic sugar or other language feature.

- Binary patching:
  - $X = $ disassembled binaries (for distance, see §II-B);
  - $G = $ sugar-neutral lifts/translations/etc.;
  - Goal: parsimoniously patch a known vulnerability.

2) Detail of toy example 1: Consider the alphabet $\mathcal{A} := \{0, 32, \ldots, 126\}$ corresponding to ASCII `NULL` and printable characters, and let $X := \oplus_{j=1}^{\infty} A_{(j)}$ be the categorical direct sum of copies of $\mathcal{A}$, i.e., the infinite sequences over $\mathcal{A}$ with only finitely many nonzero entries. The string or word w corresponding to an element of X is defined here simply by removing any (trailing) zeros and mapping the numbers to ASCII characters. We endow X with the Levenshtein distance d_X: i.e., edit distance with unit-cost insertions/deletions/substitutions. Since $|\mathcal{A}| = 96$, let $\mathcal{G} := \mathbb{Z}/96\mathbb{Z}$ act on $\mathcal{A}$ via cyclic unit shifts, and define $G := \prod_{j=1}^{\infty} \mathcal{G}_{(j)}$. Finally, define $P := X \times G$ to be the trivial "principal G-bundle" over X.

The goal is to remove non-trailing `NULL`s and punctuation, and to make the input lowercase. This goal can be achieved by different combinations of atomic horizontal and vertical steps. Here, an atomic horizontal step means replacing $x \in X$ with $x' \in X$ with $d_X(x, x') = 1$, and an atomic vertical step means applying $g_k \in G$ of the form $g_k := \oplus_{j=1}^{\infty} \delta_{jk}$, where δ_{jk} equals the identity in $\mathcal{G}$ unless $j = k$, in which case it equals the unit shift in $\mathcal{G}$ that sends $0 \mapsto 32, 32 \mapsto 33, \ldots, 125 \mapsto 126, 126 \mapsto 0$.

It is easy to see that the optimal solution (in terms of number of steps) is to delete and insert characters. This is because `A ...Z` and `a ...z` respectively correspond to the `uint8`s $(65, \ldots, 90)$ and $(97, \ldots, 122)$, and the cost of making a character lowerspace by group actions is therefore $97 - 65 = 32 = 122 - 90$, whereas the cost is only 2 to lower case via a deletion followed by an insertion.

But suppose we change the notion of an atomic vertical step to $g_k' := 31 \oplus_{j=1}^{\infty} \delta_{jk}$. Because 31 is coprime to 96, these atomic steps generate a group isomorphic to G, which

---

[8]Note that strings such as `gnat`, `tang`, `robe`, `serf`, `thug`, etc. are left unaffected by this goal.

provides a measure of justification for this change. We have that $31\delta_{jk}$ sends $0 \mapsto 61, 32 \mapsto 62, \ldots, 95 \mapsto 126, 96 \mapsto 0, 97 \mapsto 32, \ldots, 126 \mapsto 60$. Now consider the initial string $\alpha = \texttt{ABCD}$: deletion and insertion to arrive at the goal $\omega = \texttt{abcd}$ requires a cost of 8, whereas applying $g_1', \ldots, g_4'$ yields $\beta = \texttt{`abc}$ at a cost of 4. Deleting the leading $\texttt{`}$ and inserting a trailing $\texttt{d}$ incurs an extra cost of 2, for a total cost of $6 < 8$. That is, the deletion/insertion strategy is not always optimal anymore, though it usually still will be.

Observe that there is a "flat connection" on P, which is to say that the trivial factorization of $P = X \times G$ into horizontal and vertical spaces is G-equivariant. That is, for all $x, y \in X$ there exists $g \in \pi^{-1}(x) \cong G$ such that $gx = y$. [9] In other words, the trivial factorization into horizontal and vertical spaces throughout P should be taken as compatible with exchanging vertical displacements into horizontal ones:

$$\underbrace{d_G(x, gx)}_{\text{vertical displacement}} = \underbrace{d_X(x, y)}_{\text{horizontal displacement}}.$$

To encode the idea of a collection of related files, we can use a *local section* of P, which is a map $s : Y \to P$, where $Y \subseteq X$ is open (for the implicitly assumed discrete topology, this is a trivial restriction) and such that $s(y)_X := \pi(s(y)) = y$ for all $y \in Y$. Moreover, goal-directed transformations are generally (lifts of) *paths* $t : \mathbb{Z} \to Y$. A path is is a *parallel path* if $u(n)_G t(n) = t(n+1)$, for all $n \in \mathbb{Z}$, where here $u := s \circ t$ and the subscript $_G$ denotes the projection from the trivial bundle $P = X \times G$ onto its second factor. Notice that this allows the geometry of X to be lifted into G (or P) via the equation

$$d_G(u(n)_G, id_G) := d_X(t(n), t(n+1)).$$

With the definition $L(t) := \sum_{n=-\infty}^{\infty} d_X(t(n), t(n+1))$ for *path length*, we obtain the following

PROPOSITION. If t is a parallel path, then $L(t) = \sum_{n=-\infty}^{\infty} d_G(u(n)_G, id_G)$, i.e., the horizontal and vertical distances along a path are equal if the path is parallel. $\square$

Armed with this, we can study problems of finding minimal-length paths subject to constraints on the path in X, e.g., specified initial and terminal points, or more saliently the requirement to stay within a specified subspace of X. As examples of the latter sort of constraint, consider a forbidden subsequence such as (the $\texttt{uint8}$) representation of $\texttt{EOF}$ or a "buffer" constraint on the maximum number of nonzero entries of points in X. Such a restriction to some $Y \subseteq X$ entails the presence of more globally parallel sections and fewer locally parallel sections than on X, and this can be measured via

relative sheaf cohomology, e.g. by comparing the cohomology to the space of parallel sections on X. [10] [11]

3) A framework for implementations: Language representatives can be transformed by lifting/parsing them into semantics (i.e., CSTs/ASTs), transforming the semantics at a higher degree of abstraction, and projecting/unparsing them as in [18], which points out that

> [files] can be considered as an abstraction of their semantics. For example the syntax of [files] records the existence of [objects] and maybe their type but not [the trace of a parser or renderer], as defined by the semantics. [12]

In general, an AST will be subject to additional processing in order to reason over the syntax, frequently by producing an augmented AST and identifying vertices in some way to produce a suitably annotated digraph that we call a *derived syntax graph* (DSG). [13] For example, the AST produced by a PDF parser will have nodes for indirect objects and their corresponding cross references (i.e., the byte offsets in the $\texttt{xref}$ table) as well as some additional relevant information in the PDF trailer, and these data are subsequently associated to each other in a DSG, even if implicitly [13].

As a more generally familiar example, a compiler will use the AST of a computer program to produce a DSG in the guise of a control flow graph (Figure 4). While parsing an input to a CST is invertible (if it is actually performed), parsing an input directly to an AST (or transforming a CST to an AST) is obviously very far from invertible. However, the transformation from an AST to a DSG is generally (or with only minor annotations, can be made) invertible.

Our present considerations suggest a general principle by which to separate concerns in file transformations that is algorithmically favorable: compositionally [14] manipulate an appropriate DSG, then unparse it into its corresponding AST, where geometrical considerations can be more naturally *and efficiently* accounted for. For example, a DSG may be restructured, decomposed, and locally perturbed, and only the corresponding local ASTs need be compared for geometrical purposes (see §III-C) such as determining convergence to a

---

[9]For example, if $x = \texttt{ABCD}$ and $y = \texttt{ACD}$, then the corresponding $g = (0, 1, 1, -100)$ in ASCII numbering or $g = (0, 1, 1, -37)$ in "internal" numbering.

[10]In the event that G is abelian, relative sheaf cohomology is comparatively straightforward: in the more general nonabelian case, one can either panic or contemplate much more abstract techniques such as outlined in §7 of [26] (see also [27]).

[11]Equivalences of files in X induce additional highly nontrivial topological structure on the resulting quotient space, which can in turn induce curvature in *any* connection (via Chern-Weil theory, for which see, e.g. [28], [29] and whose most basic incarnation is the Gauss-Bonnet theorem that relates the integral of the Gaussian curvature over a closed orientable surface to the Euler characteristic).

[12]Here we have replaced the words "program" and "variable" with "file" and "object," respectively.

[13]Note that dependencies between (versus within) files of the same format suggest that we focus attention on properties relative to certain subspaces of the base space, with all the topological baggage that implies. It would be unusual except in very simple cases for the semantic base space to be topologically trivial. In the same vein, the strongest topological condition seems likely to be broadly applicable to a bundle in the present context is that its universal cover is homotopy equivalent to a trivial bundle [30].

[14]I.e., category-theoretically, which strongly constrains the form of a DSG.

```
 1  START
 2  do while b
 3    do while b
 4      do while b
 5        do while b
 6          S
 7        enddo
 8        S
 9      enddo
10      if b
11        do while b
12          S
13        enddo
14        if b
15          S
16        endif
17      endif
18    enddo
19  enddo
20  HALT
```
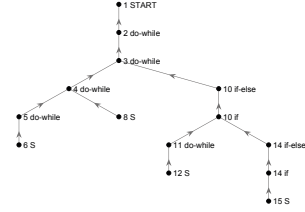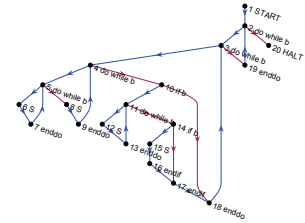
(b) The corresponding AST.

(a) A toy program skeleton. Each S is its own statement or subroutine; each b is its own Boolean predicate.

(c) The corresponding control flow graph. Branches are colored according to whether or not their corresponding b evals to $\top$ or $\bot$.

Fig. 4: When suitably annotated with concrete details, semantically richer representations of a file are better suited for performing transformations on the semantics itself. The result can be unparsed and/or syntactically transformed as appropriate. Here, the control flow graph is a semantic augmentation of the AST that includes cross-references, i.e., a DSG.

desired state, e.g. the elimination of nondeterministic syntax elements [23] or local similarity to some reference file. Such an approach inherits any compositional properties of the DSG and can be viewed through the lens of a category of lenses [31], [32] (cf. §III-A).

As a concrete example, a compiler might demand that a control flow graph be defined and decomposed as in [33], [34], restructured using the algorithm of [15] (cf. [35], [36]), and then recursively manipulate (e.g., deoptimize) subroutines only if/as needed. Extending this example, the structured program theorem [37], [38] implies that we can choose a normal form for the structured control flow in which only if-else and while (or optionally also case/switch) constructs are used (see also Fig. 2). [15]

III. FIBRATIONS

A *fibration* [40], [41] is a generalization of a fiber bundle that retains desirable homotopy properties, i.e., homotopy-equivalent fibers and the *homotopy lifting property* which states that for any f and $\tilde{f}_0$ that make the outer square of the following diagram commute, there is a map $\tilde{f}$ making the entire diagram commute:
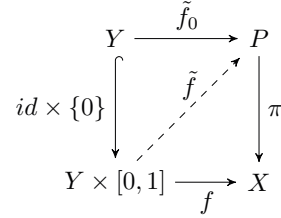
As a consequence, a path in X can be uniquely lifted to a path in P. For example, in the context of homotopy type theory, dependent types are fibrations [6]. A more general and abstract notion of fibration is provided by the theory of model categories and homotopical algebra [42].

A. Lenses as bundles and fibrations

Transforming files into a normal form has been considered as a mechanism to produce simpler, unambiguous (i.e., not polyglot [43] or schizophrenic [44]) files. However, complex structural dependencies such as checksums can obstruct *ad hoc* solutions along these lines. The notion of a *lens* [31] provides a principled, compositional solution that permits modifications to a file to be automatically transported to its putative normal form. Lenses have been synthesized at small scale from specifications and translation examples [45], [46], suggesting an approach for safely transforming files [47].

It turns out that this lens-oriented approach can be fruitfully viewed from our perspective: indeed, a generalized lens category can be defined in terms of a category $\mathcal{C}$ and a functor $F : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$ [32]. This recipe turns out to yield a *Grothendieck fibration* or *fibered category*, which can be thought of a generalized "total space" of a bundle (cf. §III-B). [16] Indeed, many of the cases motivating the definition of this generalized lens category correspond specifically to bundles, and in particular bimorphic lenses can be interpreted as trivial bundles (i.e., the total space is a Cartesian product) [50].

B. Moduli spaces

As [51] points out,

> A mathematically attractive definition of semantics is that it is the invariant after translation. If we view translation as operators between different [representations], the fact that semantics is preserved after translation means that the generators for different [representations] are all similar to one another [i.e., generators for representations commute with the corresponding translations].

In other words, semantics is a *modulus* (i.e., a complete isomorphism invariant) in the sense of algebraic geometry, wherein *moduli spaces* or *stacks* describe the algebraic invariants associated to *categories fibered in groupoids* [52], [17] and

---

[15]These considerations apply equally well to decompilation: for example the immediately preceding control flow constructs can be recognized from the graph structure of the structured control flow graph alone [39], thereby separating more mechanical issues of control flow and modularity from undecidable problems such as variable name and type inference.

[16]There are bundles (and similar objects) whose points are themselves bundles (and related objects), e.g., bundles of connections [48], moduli spaces of bundles [49], etc.

[17]For the moduli stack of elliptic curves [53], the appropriate (coarse, i.e., automorphism-forgetting) modulus is the j-invariant, which sends "the" modular curve $X(1)$ to the affine line; modular forms are sections of line bundles on this stack.

wherein the role of "total space" is played by a Grothendieck fibration [54].

In the event that such "generators" and translations can be instantiated as linear operators, the spectra of the generators ought to be *a priori* identical and yield "semantic fingerprints." [55] exploits this to perform high-performance unsupervised translation between natural language corpora. The essential step is to construct a Markov chain from statistics of the spacings between word pairs in a document, though other techniques (e.g., cross-correlations of tokens or words) might also be used in similar ways.

C. Geometry of program artifacts

Transformations on dynamic program artifacts (e.g., ASTs, traces, error ontologies etc.) define relevant groupoids, and dissimilarity measures between these artifacts define relevant geometries on fibrations and their ilk. Here, we outline various (classes) of examples in this vein.

1) ASTs: As suggested in §II-B, ASTs are well-suited for performing goal-directed transformations on files using a dissimiliarity measure (or outright metric) as an explicit objective function.

For instance, edit distances for ASTs would be considerably less computationally expensive than edit distances for DSGs. [18] Also in this particular vein, tree edit distance is appealing due to the compositional structure of dynamic programs [62] that compute it: i.e., edit distances are recursively computed from edit distances of substructures. Moreover, node annotations/labels can be taken into account in a way that separates their concerns from the tree structure by considering dissimilarities on *attributed trees*. There are several potential avenues to producing a suitable and generic dissimilarity in this vein, e.g. combining known ordered tree isomorphism algorithms with the polytime approach of [63] for attributed rooted labeled unordered trees. Another avenue is to use kernels for attributed trees [64], [65]. [19]

2) Traces: An execution trace of a parser is a more dynamic and architecture-specific (i.e., operational [66]) representation of semantics than an AST. Considering traces as paths on the control flow graph of a program, one might coarse-grain subroutines [34] or roughly equivalently, use the dynamic sequence of function calls to get a suitably high-level notion of trace to define a relevant notion of algebra on individual fibers and a geometry relating fibers. A particularly useful class of dissimilarities on traces can be constructed using [67].

In other words, each intermediate representation (token sequence, CST, AST, etc.) that a parser constructs defines a section in a fibration associated to a set of execution traces.

---

[18]Tree edit distances are still quite expensive: the best known algorithm for edit distance on rooted labeled ordered trees requires cubic time [56], and a subcubic algorithm is unlikely to exist [57], though there are reasonably good practical algorithms [58], [59] and approximations [60], [61].

[19]It might be sufficient or even practically necessary to use the trivial metric on node annotations defined by $d(x,y) := 1$ for $x \neq y$, and $d(x,x) := 0$ for all x. Besides completing the specification of a dissimilarity measure, it seems likely that this particular case would admit tailored improvements relative to generic node annotation metrics.

Due to software errors, this section is typically local, but ideally global.

3) Ontologies: An order metric [68] can be applied across multiple instantiations of parser (or more generically, program) errors. This has the advantage that we can perform topological differential testing [69] in concert with an error ontology to define a nice notion of *de facto* syntax and get a reasonable notion of the "base space" X.

4) Generalized Wasserstein metrics: A particularly interesting and general avenue for defining *bona fide* metrics using category theory is suggested by [70], which shows how to define a generalized Wasserstein metric on functors from a given small category to **Set**. If for example we take the small category to be given by two parallel morphisms between two objects, such functors are quivers/multidigraphs.

With a suitable functor, the Wasserstein metric applies to attributed quivers, and may admit specialization to be more narrowly tailored for a metric dissimilarity of a domain-specific form. One advantage of this approach is that the metric is a convex relaxation of a Hausdorff-style metric that admits computation via a linear program. Thus the algorithmic effort can be concentrated in the selection of an appropriate category and specialization of the linear program formulation to support fast evaluation.

IV. REMARKS

As §II-B2 shows, even slightly nontrivial examples are intractable to explicitly analyze with this framework, but this is not its *raison d'être*. Rather, this framework is intended to provide a conceptual basis for engineering transformation frameworks as in §II-B3. Insisting on analogies or formal identifications with bundle-like objects endowed with geometry can inform the design and implementation of goal-directed file transformations.

REFERENCES

[1] nLab authors, "Relation between type theory and category theory." http://ncatlab.org/, Jan. 2020.

[2] nLab authors, "Isbell duality." http://ncatlab.org/, Jan. 2020.

[3] J. Phillips and I. Raeburn, "Crossed products by locally unitary automorphism groups and principal bundles," *Journal of Operator Theory*, pp. 215–241, 1984.

[4] A. Connes, *Noncommutative Geometry*. Academic Press, 1994.

[5] S. Awodey and H. Forssell, "First-order logical duality," *Annals of Pure and Applied Logic*, vol. 164, no. 3, pp. 319 – 348, 2013.

[6] Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*. Univalent Foundations, 2013.

[7] J. Moeller and C. Vasilakopoulou, "Monoidal Grothendieck construction," *arXiv preprint arXiv:1809.00727*, 2018.

[8] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[9] T. Toffoli, "Reversible computing," in *International Colloquium on Automata, Languages, and Programming*, pp. 632–644, Springer, 1980.

[10] R. Kaarsgaard, *The Logic of Reversible Computing: Theory and Practice*. PhD thesis, University of Copenhagen, 2017.

[11] K. Morita, *Theory of Reversible Computing*. Springer, 2017.

[12] "Document management – Portable document format – Part 2: PDF 2.0," standard, International Organization for Standardization, Geneva, Switzerland, July 2017.

[13] J. Whitington, *PDF Explained*. O'Reilly, 2011.

[14] J. Lacomis, P. Yin, E. J. Schwartz, M. Allamanis, C. L. Goues, G. Neubig, and B. Vasilescu, "DIRE: a neural approach to decompiled identifier naming," *arXiv preprint arXiv:1909.09029*, 2019.

[15] F. Zhang and E. H. D'Hollander, "Using hammock graphs to structure programs," *IEEE Transactions on Software Engineering*, vol. 30, no. 4, pp. 231–245, 2004.

[16] S. Kobayashi and K. Nomizu, *Foundations of Differential Geometry*, vol. 1. Wiley, 1963.

[17] K. S. Lam, *Topics in Contemporary Mathematical Physics*. World Scientific, 2 ed., 2015.

[18] P. Cousot and R. Cousot, "Systematic design of program transformation frameworks by abstract interpretation," in *ACM SIGPLAN Notices*, vol. 37, pp. 178–190, ACM, 2002.

[19] H. J. Rothe, *Lattice Gauge Theories: An Introduction*. World Scientific, 3 ed., 2005.

[20] A. Dimakis and F. Muller-Hoissen, "Differential calculus and gauge theory on finite sets," *Journal of Physics A: Mathematical and General*, vol. 27, no. 9, pp. 3159–3178, 1994.

[21] E. Schulte, J. Ruchti, M. Noonan, D. Ciarletta, and A. Loginov, "Evolving exact decompilation," in *Workshop on Binary Analysis Research (BAR)*, 2018.

[22] J. Pombrio and S. Krishnamurthi, "Resugaring: lifting evaluation sequences through syntactic sugar," in *ACM SIGPLAN Notices*, vol. 49, pp. 361–371, ACM, 2014.

[23] F. Momot, S. Bratus, S. M. Hallberg, and M. L. Patterson, "The seven turrets of Babel: a taxonomy of LangSec errors and how to expunge them," in *2016 IEEE Cybersecurity Development (SecDev)*, pp. 45–52, IEEE, 2016.

[24] S. Lucks, N. M. Grosch, and J. König, "Taming the length field in binary data: calc-regular languages," in *2017 IEEE Security and Privacy Workshops (SPW)*, pp. 66–79, IEEE, 2017.

[25] P. Ganty, B. Köpf, and P. Valero, "A language-theoretic view on network protocols," in *International Symposium on Automated Technology for Verification and Analysis*, pp. 363–379, Springer, 2017.

[26] J. Lurie, *Higher Topos Theory*. Princeton, 2009.

[27] nLab authors, "Cohomology." http://ncatlab.org/, Jan. 2020.

[28] J. W. Milnor and J. D. Stasheff, *Characteristic Classes*. Princeton, 1974.

[29] S. Rosenberg, *The Laplacian on a Riemannian Manifold*. Cambridge, 1997.

[30] D. Rust, "When is a fiber of a fiber bundle a retract of the total space?." https://math.stackexchange.com/a/1993162/241 (version: 2016-10-31).

[31] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, "Combinators for bidirectional tree transformations: a linguistic approach to the view-update problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 29, no. 3, p. 17, 2007.

[32] D. I. Spivak, "Generalized lens categories via functors $\mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$," *arXiv preprint arXiv:1908.02202*, 2019.

[33] R. Johnson, D. Pearson, and K. Pingali, "The program structure tree: computing control regions in linear time," in *ACM SIGPLAN Notices*, vol. 29, pp. 171–185, ACM, 1994.

[34] S. Huntsman, "The multiresolution analysis of flow graphs," in *International Workshop on Logic, Language, Information, and Computation*, pp. 323–341, Springer, 2019.

[35] K. Yakdan, S. Eschweiler, E. Gerhards-Padilla, and M. Smith, "No more gotos: decompilation using pattern-independent control-flow structuring and semantic-preserving transformations," in *Network and Distributed System Security Symposium*, Internet Society, 2015.

[36] Y. Yu, Y. Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, and J. C. S. do Prado Leite, "Reverse engineering goal models from legacy code," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, pp. 363–372, IEEE.

[37] C. Böhm and G. Jacopini, "Flow diagrams, Turing machines and languages with only two formation rules," *Communications of the ACM*, vol. 9, no. 5, pp. 366–371, 1966.

[38] D. Harel, "On folk theorems," *Communications of the ACM*, vol. 23, no. 7, pp. 379–389, 1980.

[39] M. Sharir, "Structural analysis: a new approach to flow analysis in optimizing compilers," *Computer Languages*, vol. 5, no. 3-4, pp. 141–153, 1980.

[40] R. W. Ghrist, *Elementary Applied Topology*. Createspace, 2014.

[41] A. Hatcher, *Algebraic Topology*. Cambridge, 2002.

[42] W. G. Dwyer and J. Spalinski, "Homotopy theories and model categories," in *Handbook of Algebraic Topology* (I. M. James, ed.), pp. 73–126, Elsevier, 1995.

[43] F. Raynal, G. Delugré, and D. Aumaitre, "Malicious origami in PDF," *Journal in Computer Virology*, vol. 6, no. 4, pp. 289–315, 2010.

[44] A. Albertini, "Funky files, the novella!," *PoC ∥ GTFO 0x07:6*, 2015.

[45] A. Miltner, K. Fisher, B. C. Pierce, D. Walker, and S. Zdancewic, "Synthesizing bijective lenses," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, p. 1, 2017.

[46] S. Maina, A. Miltner, K. Fisher, B. C. Pierce, D. Walker, and S. Zdancewic, "Synthesizing quotient lenses," *Proceedings of the ACM on Programming Languages*, vol. 2, no. ICFP, p. 80, 2018.

[47] W. Harris. Personal communication, 2019.

[48] S. Kobayaschi, "Theory of connections," *Annali di Matematica Pura ed Applicata*, vol. 43, no. 1, pp. 119–194, 1957.

[49] nLab authors, "Moduli space of bundles." http://ncatlab.org/, Jan. 2020.

[50] D. I. Spivak, "Lenses: applications and generalizations." http://math.ucr.edu/home/baez/ACTUCR2019/ACTUCR2019_spivak.pdf.

[51] W. E, "Machine learning: mathematical theory and scientific applications," *Notices of the American Mathematical Society*, vol. 66, no. 12, pp. 1813–1820, 2019.

[52] B. Noohi, "A quick introduction to fibered categories and topological stacks." http://www.maths.qmul.ac.uk/ noohi/papers/quick.pdf.

[53] nLab authors, "Moduli stack of elliptic curves." http://ncatlab.org/, Jan. 2020.

[54] The Stacks Project authors, "The Stacks Project." https://stacks.math.columbia.edu/tag/003S, 2020.

[55] W. E and Y. Zhou, "A mathematical model for linguistic universals," *arXiv preprint arXiv:1907.12293*, 2019.

[56] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann, "An optimal decomposition algorithm for tree edit distance," *ACM Transactions on Algorithms (TALG)*, vol. 6, no. 1, p. 2, 2009.

[57] K. Bringmann, P. Gawrychowski, S. Mozes, and O. Weimann, "Tree edit distance cannot be computed in strongly subcubic time (unless APSP can)," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1190–1206, Society for Industrial and Applied Mathematics, 2018.

[58] M. Pawlik and N. Augsten, "Efficient computation of the tree edit distance," *ACM Transactions on Database Systems (TODS)*, vol. 40, no. 1, p. 3, 2015.

[59] M. Pawlik and N. Augsten, "Tree edit distance: robust and memory-efficient," *Information Systems*, vol. 56, pp. 157–173, 2016.

[60] A. Torsello and E. R. Hancock, "Efficiently computing weighted tree edit distance using relaxation labeling," in *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pp. 438–453, Springer, 2001.

[61] B. Paaßen, C. Gallicchio, A. Micheli, and B. Hammer, "Tree edit distance learning via adaptive symbol embeddings," in *International Conference on Machine Learning*, pp. 3976–3985, 2018.

[62] O. D. Moor, "Categories, relations and dynamic programming," *Mathematical Structures in Computer Science*, vol. 4, no. 1, pp. 33–69, 1994.

[63] A. Torsello, D. Hidovic-Rowe, and M. Pelillo, "Polynomial-time metrics for attributed trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1087–1099, 2005.

[64] F. Aiolli, G. Da San Martino, and A. Sperduti, "An efficient topological distance-based tree kernel," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 5, pp. 1115–1120, 2015.

[65] P. Duessel, *Detection of Unknown Cyber Attacks Using Convolution Kernels Over Attributed Language Models*. PhD thesis, Universität Bonn, 2018.

[66] H. R. Nielson and F. Nielson, *Semantics With Applications: an Appetizer*. Springer, 2007.

[67] S. Huntsman, "De Bruijn entropy and string similarity," in *Combinatorics on Words*, Kiel Computer Science Series, 2015.

[68] C. Joslyn and E. Hogan, "Order metrics for semantic knowledge systems," in *International Conference on Hybrid Artificial Intelligence Systems*, pp. 399–409, Springer, 2010.

[69] S. Huntsman, M. Robinson, and M. Yutin, "Topological differential testing." In preparation.

[70] E. Patterson, "Hausdorff and Wasserstein metrics on graphs and other structured data," *arXiv preprint arXiv:1907.00257*, 2019.