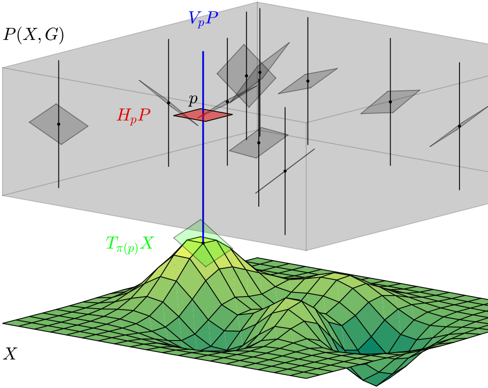
The geometry of syntax and semantics for directed file transformations

Steve Huntsman [1]
Michael Robinson [2]

[1]FAST Labs / Cyber Technology

[2]American University

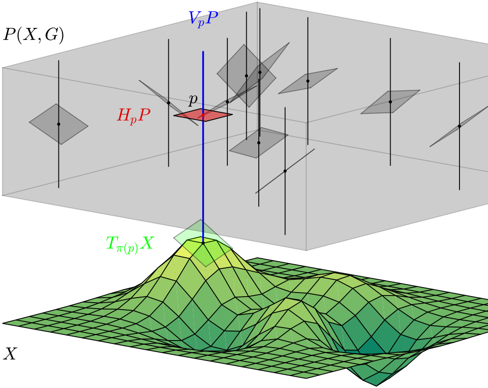
21 May 2020



BAE SYSTEMS

`string.h` must be used carefully to prevent buffer overflo

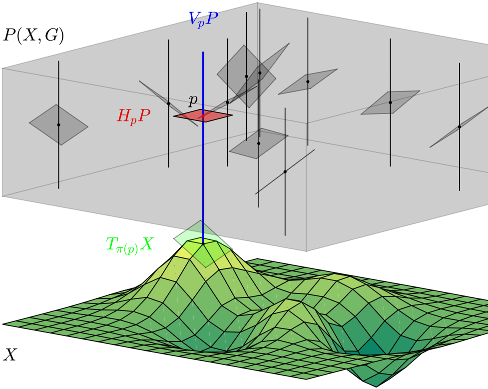
- X = strings of ASCII `NULL`s and printable characters
- G = cyclic shifts on individual characters
- Goal: remove `NULL`s and punctuation; make lowercase
- This example is discussed in the paper



BAE SYSTEMS

Transform files to achieve language-theoretical security

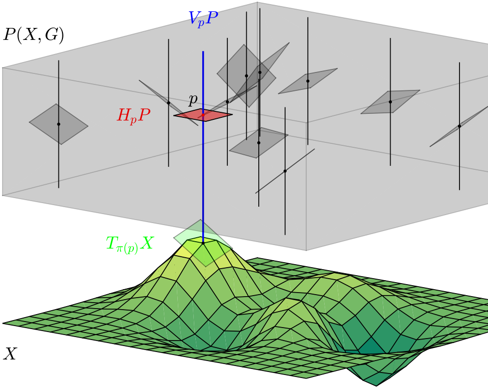
- X = space of files in some fixed format (e.g., PDF)
- G = various **invertible** transformations
- Goal: eliminate nondeterministic syntax
- Input ambiguity = vulnerability



BAE SYSTEMS

Patch binary code to secure critical legacy systems

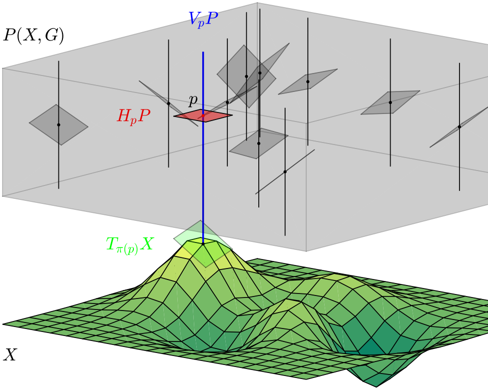
- X = space of disassembled binary code
- G = "sugar-neutral" **lifts**, translations, etc
- Goal: parsimoniously patch a known vulnerability
- Compiler/build options, dependencies make this hard



BAE SYSTEMS

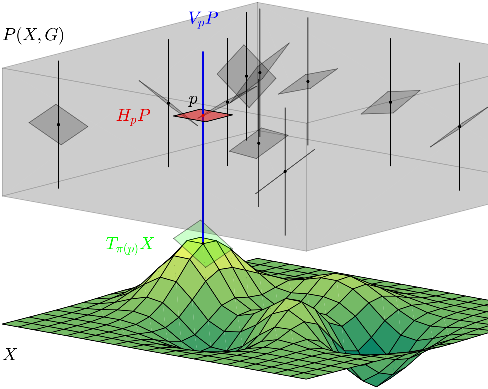
Principal bundles model syntax and semantics

- X = space of documents
- G = group of invertible transformations



Principal bundles model syntax and semantics

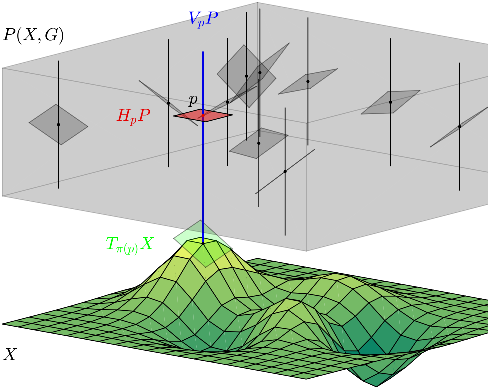
- X = space of documents
- G = group of invertible transformations
- Think of X like a manifold and get something akin to a *principal bundle* $P(X, G)$



BAE SYSTEMS

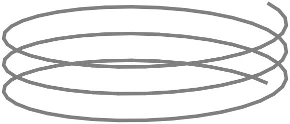
Principal bundles model syntax and semantics

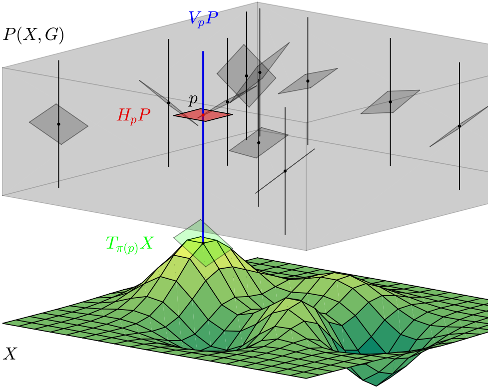
- X = space of documents
- G = group of invertible transformations
- Think of X like a manifold and get something akin to a *principal bundle $P(X, G)$*
    - Locally looks like $X \times G$
    - G acts on P nicely



Principal bundles model syntax and semantics

- X = space of documents
- G = group of invertible transformations
- Think of X like a manifold and get something akin to a *principal bundle* $P(X, G)$
  - Locally looks like $X \times G$
  - G acts on P nicely
- E.g., $X = S^1$ (time of day); $G = \mathbb{Z}$ (epoch); $P = \mathbb{R}$ (as a helix above X)

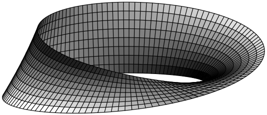


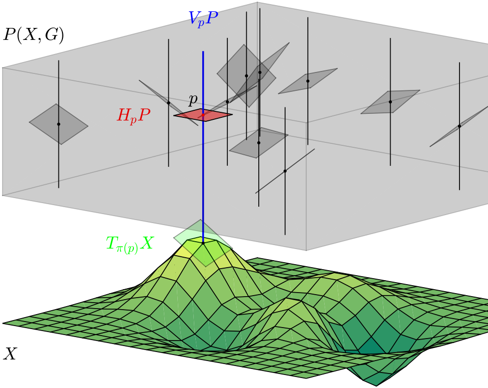


BAE SYSTEMS

Principal bundles model syntax and semantics

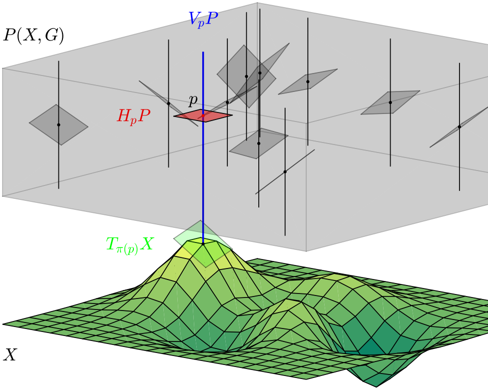
- X = space of documents
- G = group of invertible transformations
- Think of X like a manifold and get something akin to a *principal bundle* $P(X, G)$
  - Locally looks like $X \times G$
  - G acts on P nicely
- E.g., $X = S^1$; $G = (0, 1)$ w/ $x \boxplus y := f(f^{-1}(x) + f^{-1}(y))$ for invertible $f : \mathbb{R} \to (0, 1)$





Principal bundles model syntax and semantics

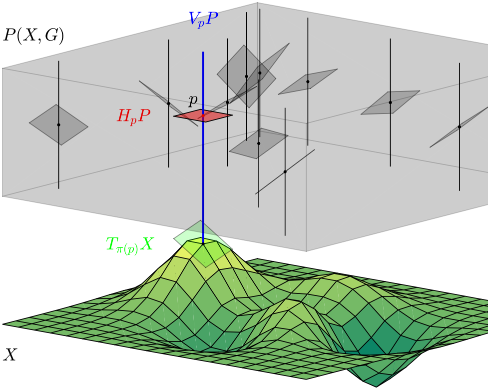
- X = space of documents
- G = group of invertible transformations
- Think of X like a manifold and get something akin to a *principal bundle* $P(X, G)$
  - Locally looks like $X \times G$
  - G acts on P nicely
- E.g., *Hopf fibration* $S^1 \to S^3 \to S^2$



BAE SYSTEMS

Connections model geometry directing transformations

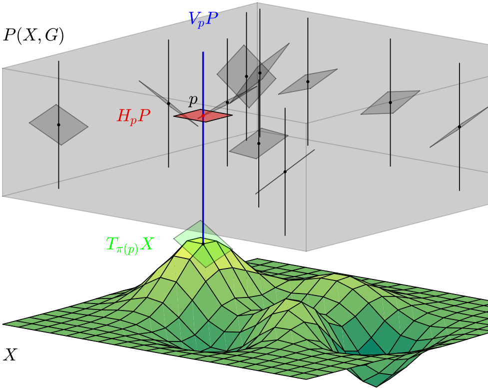
- Principal bundles are a natural arena for geometry realized through a *connection*



BAE SYSTEMS

Connections model geometry directing transformations

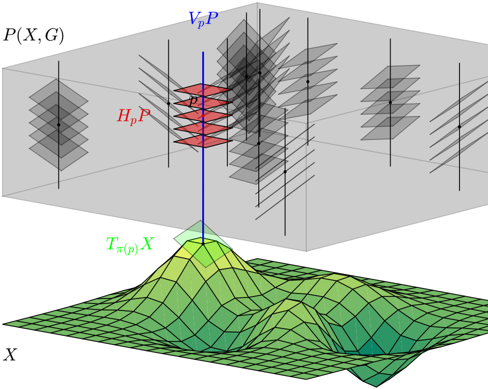
- Principal bundles are a natural arena for geometry realized through a *connection*
  - I.e., a "vertical" and "horizontal" direct sum decomposition of tangent spaces ...



BAE SYSTEMS

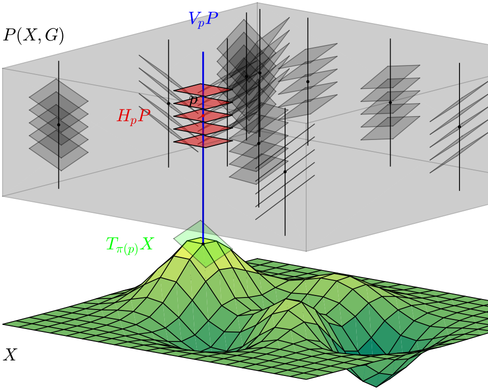
Connections model geometry directing transformations

- Principal bundles are a natural arena for geometry realized through a *connection*
  - I.e., a "vertical" and "horizontal" direct sum decomposition of tangent spaces ...
  - ...that is *equivariant* under group action



Connections model geometry directing transformations

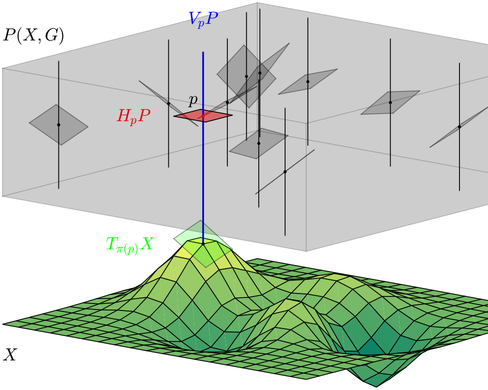
- Principal bundles are a natural arena for geometry realized through a *connection*
  - I.e., a "vertical" and "horizontal" direct sum decomposition of tangent spaces ...
  - ...that is *equivariant* under group action
- Connects local product geometries via *parallel transport*



BAE SYSTEMS

Syntactic transformations must be invertible

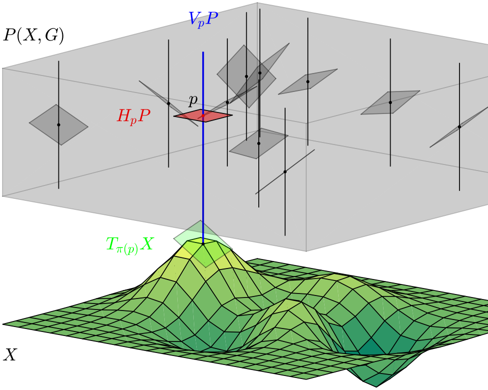
- This **requirement** of the mathematical model is really a **hint** about how to perform file transformations



BAE SYSTEMS

Syntactic transformations must be invertible

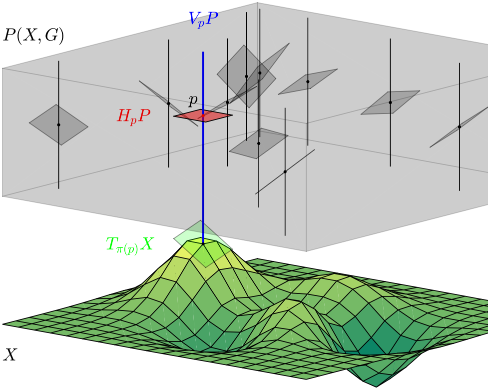
- This **requirement** of the mathematical model is really a **hint** about how to perform file transformations
- Record (or in reverse, delete) details of atomic transformations in ancillae



Syntactic transformations must be invertible

- This **requirement** of the mathematical model is really a **hint** about how to perform file transformations
- Record (or in reverse, delete) details of atomic transformations in ancillae

objend

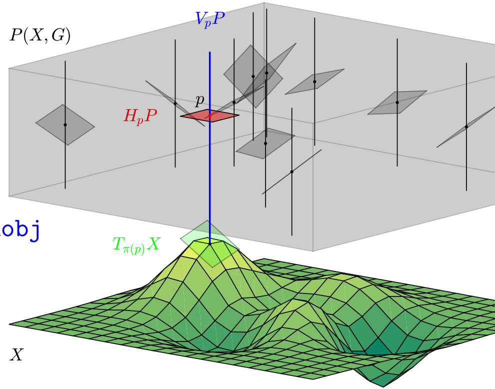


BAE SYSTEMS

Syntactic transformations must be invertible

- This **requirement** of the mathematical model is really a **hint** about how to perform file transformations
- Record (or in reverse, delete) details of atomic transformations in ancillae

```
objend
    ⇒ objend % objend -> endobj
```

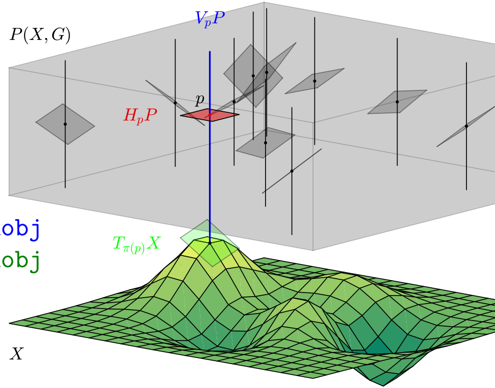


BAE SYSTEMS

Syntactic transformations must be invertible

- This **requirement** of the mathematical model is really a **hint** about how to perform file transformations
- Record (or in reverse, delete) details of atomic transformations in ancillae

```
objend
   ⇒ objend % objend -> endobj
   ⇒ endobj % objend -> endobj
```

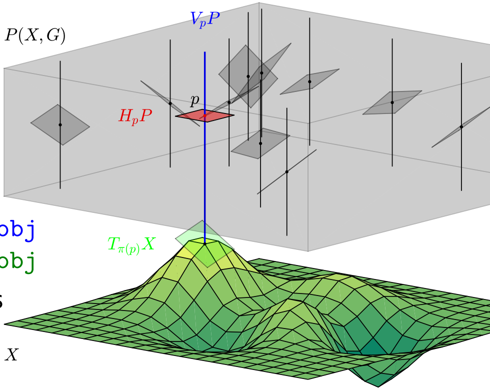


Syntactic transformations must be invertible

- This **requirement** of the mathematical model is really a **hint** about how to perform file transformations
- Record (or in reverse, delete) details of atomic transformations in ancillae

```
objend
   ⇒ objend % objend -> endobj
   ⇒ endobj % objend -> endobj
```

- *Sugar-neutral*: transformations should handle sugar, but not introduce or eliminate it



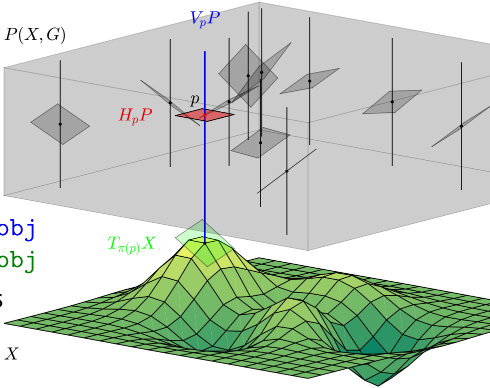
BAE SYSTEMS

Syntactic transformations must be invertible

- This **requirement** of the mathematical model is really a **hint** about how to perform file transformations
- Record (or in reverse, delete) details of atomic transformations in ancillae

```
objend
   ⇒ objend % objend -> endobj
   ⇒ endobj % objend -> endobj
```

- *Sugar-neutral*: transformations should handle sugar, but not introduce or eliminate it
  - Suggests using *normal forms*



BAE SYSTEMS

Normal forms simplify and disambiguate

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

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

(From Lacomis *et al.*)

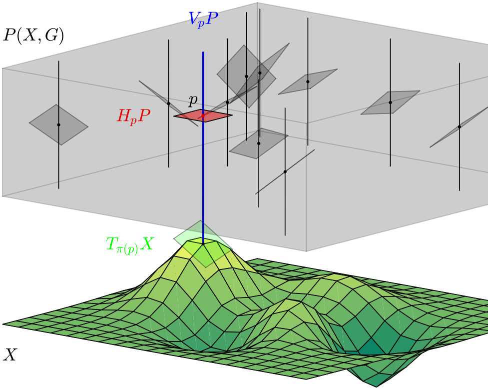
```
      jmp @5
@4:
      jmp @9
@8:
      jne @19
      jmp @10
@19:
      jmp @14
@13:@14:
      jg  @13
@9:@10:
      jge @20
      jmp @8
@5:@20:
      jge @21
      jmp @4
@21:
```

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

(From Zhang and D'Hollander)

Concrete syntax trees parameterize a principal bundle

- G corresponds to semantics-preserving CST transformations
- Equivalence class of CSTs corresponding to a given AST has group-theoretical and language security significance and indicates format redundancy
    - E.g., `xref` table in PDF (which nobody trusts)



BAE SYSTEMS

Dynamic concretization semantically enriches an AST

[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. [1]

- Annotating (with, e.g., types) and cross-linking an AST gives a semantically rich *derived graph*
- To understand a file, parse it . . .

[1] [Cousot and Cousot], replacing "program" and "variable" with "file" and "object," respectively.

Dynamic concretization semantically enriches an AST

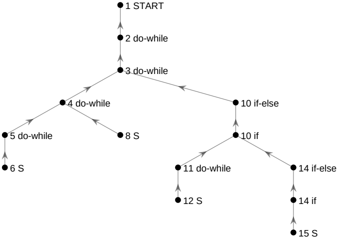
[Files] can be considered as an abstraction of their semantics. For example the syntax of [files] records the existence of [objects] and maybe their type but not [the trace of a parser or renderer], as defined by the semantics. [1]

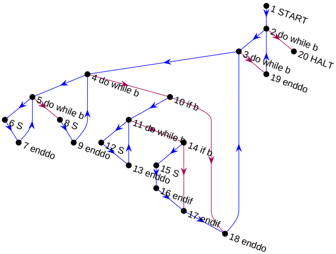
- Annotating (with, e.g., types) and cross-linking an AST gives a semantically rich *derived graph*
- To understand a file, parse it . . .
- . . . to understand it more, render/compile it

[1] [Cousot and Cousot], replacing "program" and "variable" with "file" and "object," respectively.

To transform syntax trees, transform derived graphs

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

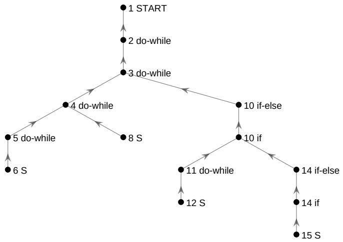


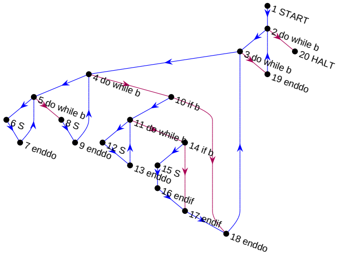


- Compilers parse source code into abstract syntax tree, then cross-link into control flow graph

To transform syntax trees, transform derived graphs

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



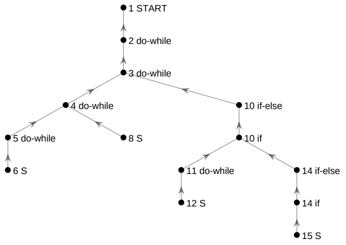


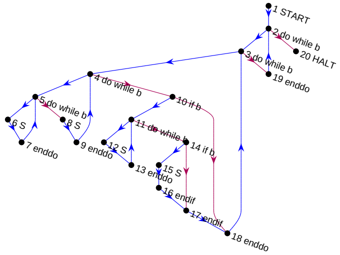
- Compilers parse source code into abstract syntax tree, then cross-link into control flow graph
- PDF analogue: indirect object cross-references

BAE SYSTEMS

To transform syntax trees, transform derived graphs

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



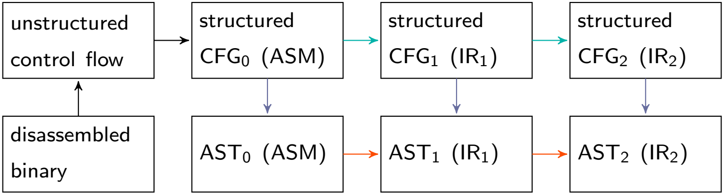


- Compilers parse source code into abstract syntax tree, then cross-link into control flow graph
- PDF analogue: indirect object cross-references
- Transform the derived graph to transform ASTs

BAE SYSTEMS

To transform syntax trees, transform derived graphs

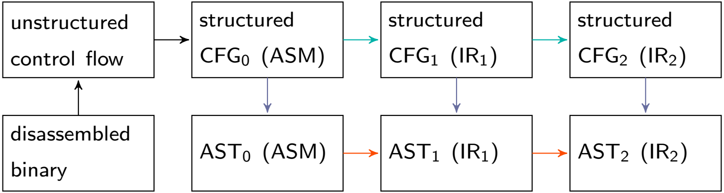
- Compositionally transform derived graphs
  - Restructure; decompose/locally perturb
- Invertibly reduce derived graphs back to syntax trees
  - Local AST dissimilarities suffice for geometry
    - E.g., elimination of nondeterministic syntax elements
    - E.g., local similarity to some reference file



BAE SYSTEMS

To transform syntax trees, transform derived graphs

- Compositionally transform derived graphs
  - Restructure; decompose/locally perturb
- Invertibly reduce derived graphs back to syntax trees
  - Local AST dissimilarities suffice for geometry
    - E.g., elimination of nondeterministic syntax elements
    - E.g., local similarity to some reference file
- This approach inherits compositionality of derived graphs and can be viewed through the lens of a category of lenses



BAE SYSTEMS

Dissimilarities on file artifacts yield geometry

- Attributed tree dissimilarities for CSTs
  - Edit distance exploits compositionality
  - Kernels are fast

Dissimilarities on file artifacts yield geometry

- Attributed tree dissimilarities for CSTs
  - Edit distance exploits compositionality
  - Kernels are fast
- Sequence dissimilarities for traces
  - Any parser IR (token sequence, CST, AST, etc.) defines a *section* associated to a set of execution traces
  - Software errors $\Rightarrow$ section is typically local, but ideally global

Dissimilarities on file artifacts yield geometry

- Attributed tree dissimilarities for CSTs
  - Edit distance exploits compositionality
  - Kernels are fast
- Sequence dissimilarities for traces
  - Any parser IR (token sequence, CST, AST, etc.) defines a *section* associated to a set of execution traces
  - Software errors $\Rightarrow$ section is typically local, but ideally global
- Order metric for ontologies
  - Use w/ topological differential testing for *de facto* syntax $\Rightarrow X$

Dissimilarities on file artifacts yield geometry

- Attributed tree dissimilarities for CSTs
    - Edit distance exploits compositionality
    - Kernels are fast
- Sequence dissimilarities for traces
    - Any parser IR (token sequence, CST, AST, etc.) defines a *section* associated to a set of execution traces
    - Software errors $\Rightarrow$ section is typically local, but ideally global
- Order metric for ontologies
    - Use w/ topological differential testing for *de facto* syntax $\Rightarrow X$
- Wasserstein metric on functor from a small category to **Set**
    - E.g., small category = two parallel morphisms between two objects $\Rightarrow$ functor = quiver
    - Convex relaxation of Hausdorff-style metric $\Rightarrow$ linear program
    - Attributed/labeled structures not covered by this at present

In general, consider fibrations endowed with geometry

- A *fibration* is a generalization of a fiber bundle that retains desirable homotopy properties
  - Homotopy-equivalent fibers
  - *Homotopy lifting property*: if $f, \tilde{f}_0$ make the outer square commute, there exists $\tilde{f}$ making the entire diagram commute
- Key feature: a path in X can be uniquely lifted to a path in P

$$
\begin{tikzcd}
Y \arrow[r, "\tilde{f}_0"] \arrow[d, hook, "id \times \{0\}"'] & P \arrow[d, "\pi"] \\
Y \times [0,1] \arrow[r, "f"'] \arrow[ur, dashed, "\tilde{f}"] & X
\end{tikzcd}
$$

In general, consider fibrations endowed with geometry

- A *fibration* is a generalization of a fiber bundle that retains desirable homotopy properties
  - Homotopy-equivalent fibers
  - *Homotopy lifting property*: if $f, \tilde{f}_0$ make the outer square commute, there exists $\tilde{f}$ making the entire diagram commute
- Key feature: a path in X can be uniquely lifted to a path in P
- Homotopy type theory: dependent types are fibrations

$$
\begin{tikzcd}
Y \arrow[r, "\tilde{f}_0"] \arrow[d, hook, "id \times \{0\}"'] & P \arrow[d, "\pi"] \\
Y \times [0,1] \arrow[r, "f"'] \arrow[ur, dashed, "\tilde{f}"] & X
\end{tikzcd}
$$

In general, consider fibrations endowed with geometry

- A *fibration* is a generalization of a fiber bundle that retains desirable homotopy properties
  - Homotopy-equivalent fibers
  - *Homotopy lifting property*: if $f, \tilde{f}_0$ make the outer square commute, there exists $\tilde{f}$ making the entire diagram commute
- Key feature: a path in X can be uniquely lifted to a path in P
- Homotopy type theory: dependent types are fibrations
- Avoid invertibility requirement via monoidal fibrations?
  - Maybe, but trading simplicity for generality is not a good start

$$\begin{tikzcd}
Y \arrow[r, "\tilde{f}_0"] \arrow[d, hook, "id \times \{0\}"'] & P \arrow[d, "\pi"] \\
Y \times [0,1] \arrow[r, "f"'] \arrow[ur, dashed, "\tilde{f}"] & X
\end{tikzcd}$$

BAE SYSTEMS

Lenses can help with complex file transformations

- Simple structural dependencies such as cross-references can be handled using a derived graph
- Complex structural dependencies such as checksums can obstruct *ad hoc* transformations to a normal form

Lenses can help with complex file transformations

- Simple structural dependencies such as cross-references can be handled using a derived graph
- Complex structural dependencies such as checksums can obstruct *ad hoc* transformations to a normal form
- The notion of a *lens* 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, suggesting an approach for safely transforming files

Generalized lenses are Grothedieck fibrations

- A generalized lens category can be defined in terms of a category $\mathcal{C}$ and a functor $F : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$
- This recipe turns out to yield a *Grothendieck fibration* or *fibered category*
  - Generalized "total space" of a bundle

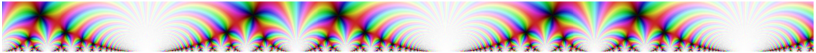
Generalized lenses are Grothedieck fibrations

- A generalized lens category can be defined in terms of a category $\mathcal{C}$ and a functor $F : \mathcal{C}^{\mathrm{op}} \to \mathbf{Cat}$
- This recipe turns out to yield a *Grothendieck fibration* or *fibered category*
  - Generalized "total space" of a bundle
- Many of the cases motivating the definition of this generalized lens category correspond specifically to bundles
  - Bimorphic lenses can be interpreted as trivial bundles (i.e., the total space is a Cartesian product)

BAE SYSTEMS

Semantics is a *modulus* (complete isomorphism invariant)

> *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 commute with translations].* [2]



- *Moduli spaces* or *stacks* describe the algebraic invariants associated to *categories fibered in groupoids*
  - For the moduli stack of elliptic curves the appropriate (coarse, i.e., automorphism-forgetting) modulus is the j-invariant
  - *Modular forms* are sections of line bundles on this stack
- The role of "total space" is played by a Grothendieck fibration

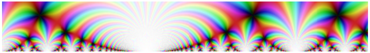
[2] [E and Zhou]

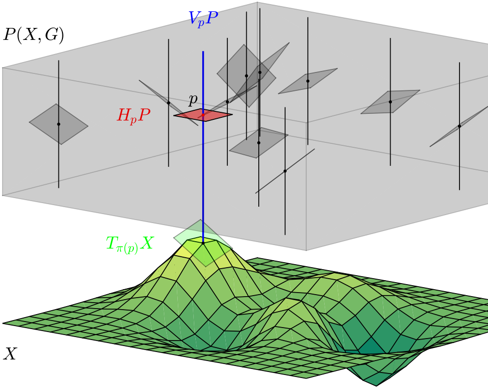
Thanks

BAE Systems FAST Labs @ https://bit.ly/2X2UwcP

steve.huntsman @ baesystems.com

paper @ https://arxiv.org/abs/2001.04952





BAE SYSTEMS

Syntax : semantics :: algebra : [arena for] geometry

The duality between syntax and semantics is really a manifestation of that between algebra and geometry. [3]

[3] [Awodey and Forssell]

Syntax : semantics :: algebra : [arena for] geometry

- LHS = categorical logic
  - Simply typed lambda calculus : Cartesian closed category
  - First-order logic : hyperdoctrine
  - Dependent type theory : locally Cartesian closed category
  - Homotopy type theory : elementary $(\infty, 1)$ topos

The duality between syntax and semantics is really a manifestation of that between algebra and geometry. [3]

[3] [Awodey and Forssell]

Syntax : semantics :: algebra : [arena for] geometry

- LHS = categorical logic
  - Simply typed lambda calculus : Cartesian closed category
  - First-order logic : hyperdoctrine
  - Dependent type theory : locally Cartesian closed category
  - Homotopy type theory : elementary $(\infty, 1)$ topos
- RHS = Isbell/sheaf/spectral duality; noncommutative topology
  - Boolean algebra : Stone space
  - Commutative C*-algebra : compact Hausdorff space
  - Commutative ring : affine scheme
  - **Crossed product C*-algebra : principal bundle**

The duality between syntax and semantics is really a manifestation of that between algebra and geometry. [3]

[3] [Awodey and Forssell]

BAE SYSTEMS

Syntax : semantics :: algebra : [arena for] geometry

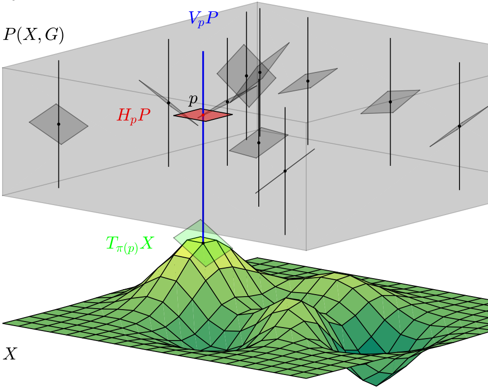
- LHS = categorical logic
  - Simply typed lambda calculus : Cartesian closed category
  - First-order logic : hyperdoctrine
  - Dependent type theory : locally Cartesian closed category
  - Homotopy type theory : elementary $(\infty, 1)$ topos
- RHS = Isbell/sheaf/spectral duality; noncommutative topology
  - Boolean algebra : Stone space
  - Commutative C*-algebra : compact Hausdorff space
  - Commutative ring : affine scheme
  - **Crossed product C*-algebra : principal bundle**
- No actual geometry yet, just spaces as substrates

The duality between syntax and semantics is really a manifestation of that between algebra and geometry. [3]

[3][Awodey and Forssell]

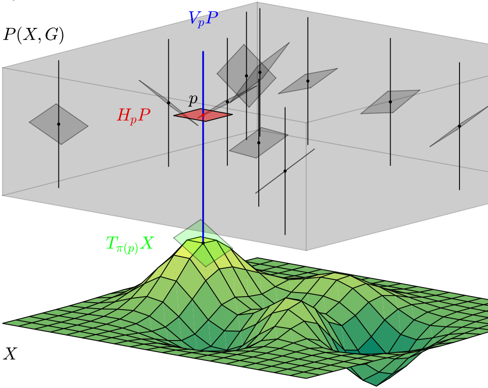
Bundle geometry connects algebra across a base space

- Syntax transformations form a group (or groupoid)
- Semantically distinct (reps of) files form a "base space"



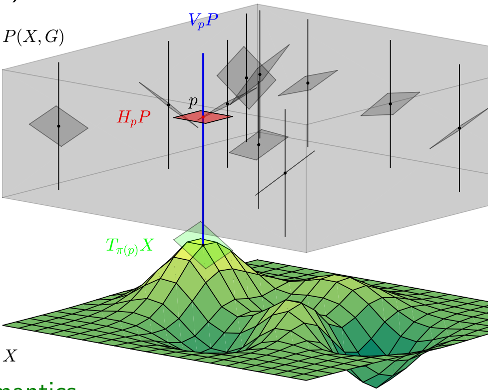
Bundle geometry connects algebra across a base space

- Syntax transformations form a group (or groupoid)
- Semantically distinct (reps of) files form a "base space"
    - *Homotopy hypothesis*: spaces = ∞-groupoids
    - Geometric interpretation for base vs algebraic interpretation for paths



Bundle geometry connects algebra across a base space

- Syntax transformations form a group (or groupoid)
- Semantically distinct (reps of) files form a "base space"
  - *Homotopy hypothesis*: spaces = ∞-groupoids
  - Geometric interpretation for base vs algebraic interpretation for paths
- Unifying constructs: bundles and fibrations
  - Goal-directed file transformation imbues a notion of geometry connecting syntax and semantics



BAE SYSTEMS