

Intelligent pattern profiling

Trifacta (@trifacta)
Karthik Sethuraman
Sean Kandel



How is your data structured?

How is your data structured?

Is all your data structured identically?

How is your data structured?

Is all your data structured identically?

How to map input structure(s) to desired output?

How is your data structured?

Is all your data structured identically?

How to map input structure(s) to desired output?

ABC	IP	⌚	DATE	ABC	TZ	ABC	METHOD	ABC	PATH
2,451 Categories		Oct 25 - Nov 14		1 Category		1 Category		103 Categories	
668.667.44.3		25/Oct/2011:07:38:30		-0500		GET		/download/download	
13.386.648.380		25/Oct/2011:17:06:00		-0500		GET		/download/download	
06.670.03.40		26/Oct/2011:13:24:00		-0500		GET		/product/demos/pr	
18.656.618.46		26/Oct/2011:17:15:30		-0500		GET		/download/download	
14.688.663.667		26/Oct/2011:21:02:30		-0500		GET		/news	
13.07.338.684		26/Oct/2011:21:02:30		-0500		GET		/download	
14.688.663.667		26/Oct/2011:21:02:30		-0500		GET		/news	
688.615.03.332		26/Oct/2011:21:02:30		-0500		GET		/product/product1	
688.615.03.332		26/Oct/2011:21:02:32		-0500		GET		/product/product1	
688.615.03.332		26/Oct/2011:21:02:34		-0500		GET		/products/demos	
13.07.338.684		26/Oct/2011:21:02:37		-0500		GET		/download	
55.3.658.53		26/Oct/2011:21:06:30		-0500		GET		/buy	
55.3.658.53		26/Oct/2011:21:06:56		-0500		GET		/buy	
14.323.74.653		26/Oct/2011:21:07:00		-0500		GET		/demo	
14.323.74.653		26/Oct/2011:21:08:00		-0500		GET		/demo	
14.323.74.653		26/Oct/2011:21:09:00		-0500		GET		/demo	
14.323.74.653		26/Oct/2011:21:10:03		-0500		GET		/demo	



6 Keys

```
{ "IP": "668.667.44.3", "DATE": "25/Oct/2011:07:38:30", "TZ": "-0500", "METHOD": "GET", "PATH": "/download/download3.zip", "St >  
{ "IP": "13.386.648.380", "DATE": "25/Oct/2011:17:06:00", "TZ": "-0500", "METHOD": "GET", "PATH": "/download/download6.zip", " >  
{ "IP": "06.670.03.40", "DATE": "26/Oct/2011:13:24:00", "TZ": "-0500", "METHOD": "GET", "PATH": "/product/demos/product2", "St >  
{ "IP": "18.656.618.46", "DATE": "26/Oct/2011:17:15:30", "TZ": "-0500", "METHOD": "GET", "PATH": "/download/download4.zip", "S >  
{ "IP": "14.688.663.667", "DATE": "26/Oct/2011:21:02:30", "TZ": "-0500", "METHOD": "GET", "PATH": "/news", "Status": "200"}  
{ "IP": "13.07.338.684", "DATE": "26/Oct/2011:21:02:30", "TZ": "-0500", "METHOD": "GET", "PATH": "/download", "Status": "200"}  
{ "IP": "14.688.663.667", "DATE": "26/Oct/2011:21:02:30", "TZ": "-0500", "METHOD": "GET", "PATH": "/news", "Status": "200"}>
```

```
{  
  "IP": "688.615.03.332",  
  "DATE": "26/Oct/2011:21:02:30",  
  "TZ": "-0500",  
  "METHOD": "GET",  
  "PATH": "/product/product1",  
  "Status": "200"  
}
```



How is your data structured?

Is all your data structured identically?

How to map input structure(s) to desired output?

Relational data often has multi-structured columns

ABC name ▾	ABC address ▾	ABC phone ▾
2 Categories	2 Categories	2 Categories
Jane · Doe	1601 · W. · Broadway, · Anaheim, · CA, · 92108	123-456-7890
Doe, · John	132 · Sansom · Street, · Philadelphia, · PA	1 · (123) · 456-7890

Log data contains many types of events



How is your data structured?

Is all your data structured identically?

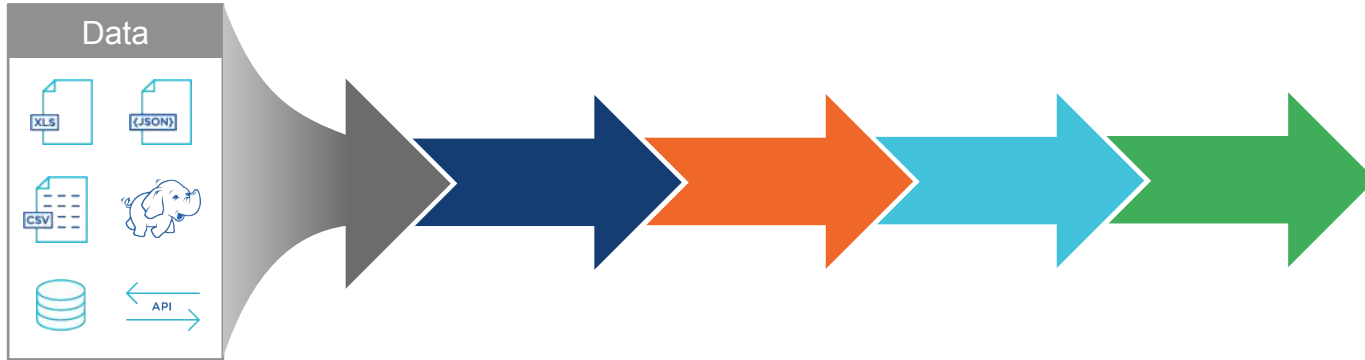
How to map input structure(s) to desired output?

Analysis and visualization
tools generally require
tabular and homogenous data

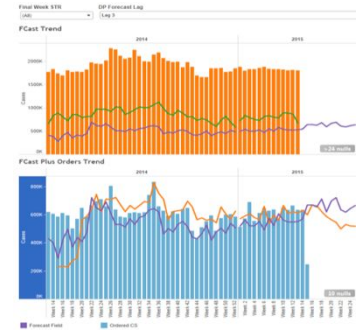
Data is increasingly
non-tabular or heterogeneous

The challenge

How do we make this more efficient?



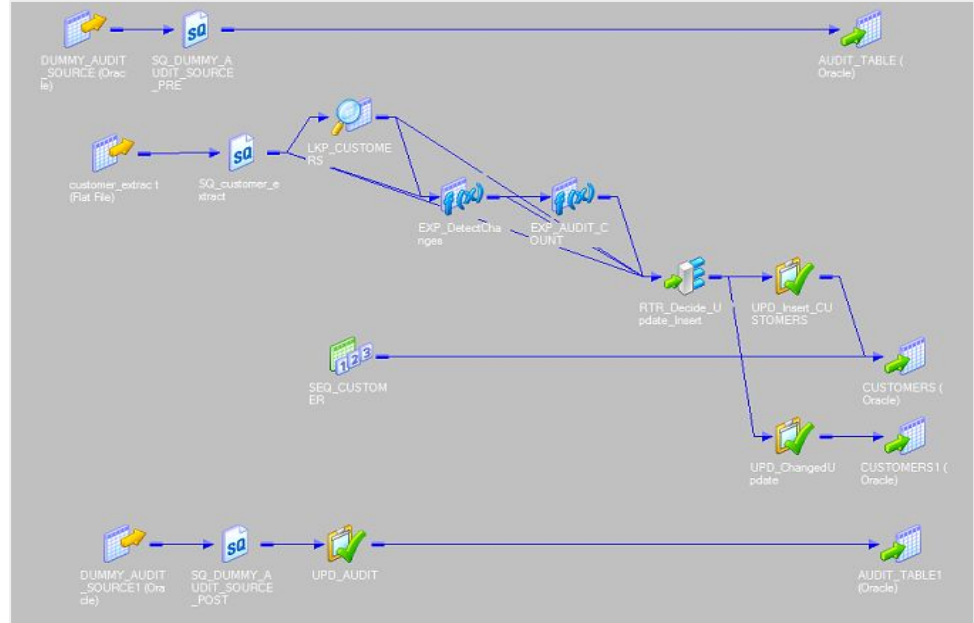
To drive more value here!



Conventional approaches inhibit self-service

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
4 requests = pd.read_csv('.../data/311-service-requests.csv')
5 requests['Incident Zip'].unique()
6 na_values = ['NO CLUE', 'N/A', '0']
7 requests = pd.read_csv('.../data/311-service-requests.csv',
8 na_values=na_values, dtype={'Incident Zip': str})
9
10 requests['Incident Zip'].unique()
11
12
13
14 rows_with_dashes = requests['Incident Zip'].str.contains('-').fillna(False)
15 len(requests[rows_with_dashes])
16
17
18 requests[rows_with_dashes]
19
20
21 long_zip_codes = requests['Incident Zip'].str.len() > 5
22 requests['Incident Zip'][long_zip_codes].unique()
23
24
25 requests['Incident Zip'] = requests['Incident Zip'].str.slice(0, 5)
26
27
28 requests[requests['Incident Zip'] == '00000']
29
30
31 zero_zips = requests['Incident Zip'] == '00000'
32 requests.loc[zero_zips, 'Incident Zip'] = np.nan
33
34
35 unique_zips = requests['Incident Zip'].unique()
36 unique_zips.sort()
37 unique_zips
38
39 zips = requests['Incident Zip']
40 is_close = zips.str.startswith('0')
41 is_far = ~(is_close) & zips.notnull()
42
43
44 zips[is_far]
45
46 requests[is_far][['Incident Zip', 'Descriptor', 'City']].sort('Incident Zip')
47
48
49 requests['City'].str.upper().value_counts()
50
51
52 na_values = ['NO CLUE', 'N/A', '0']
53 requests = pd.read_csv('.../data/311-service-requests.csv',
54 na_values=na_values,
55 dtype={'Incident Zip': str})
56
57
58 def fix_zip_codes(zips):
59     # Truncate everything to length 5
60     zips = zips.str.slice(0, 5)
61     # Set 00000 zip codes to nan
62
```

Hand-Coding



Search / ETL

Conventional approaches inhibit self-service

Users must understand complex regular expressions.

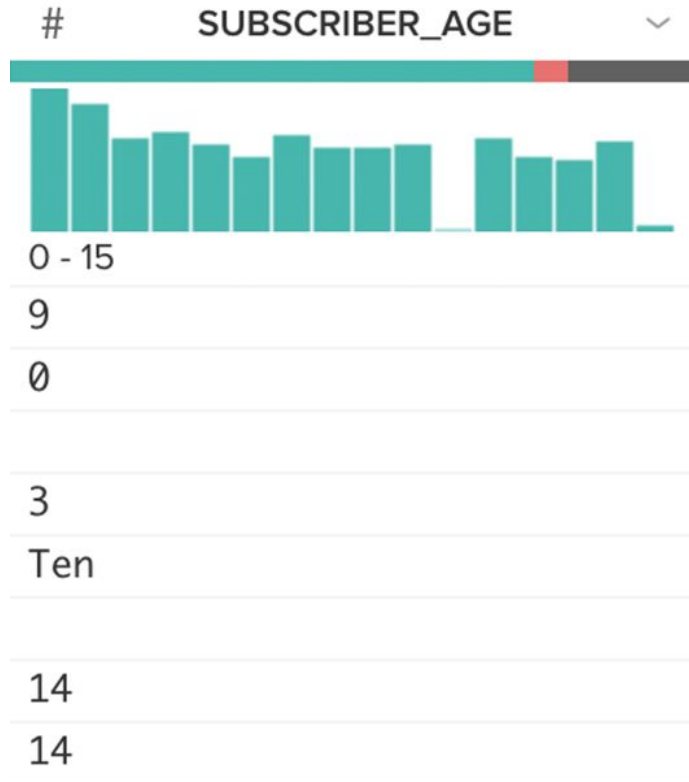
```
MOC | SFR/621630263 | /  
MTC | ORG1/638590539 | /  
SMS-MT | SMSC/600000000 | BOY/658510643  
SMS-MO | SMSC/600000000 | SFR/634989093  
SMS-MO | SMSC/600000000 | ORG1/608564604
```



```
[A-Z]+(\-[A-Z]+)?\ | [A-Z0-9]+/[0-9]{9}/([A-Z]+/[0-9]+)?
```

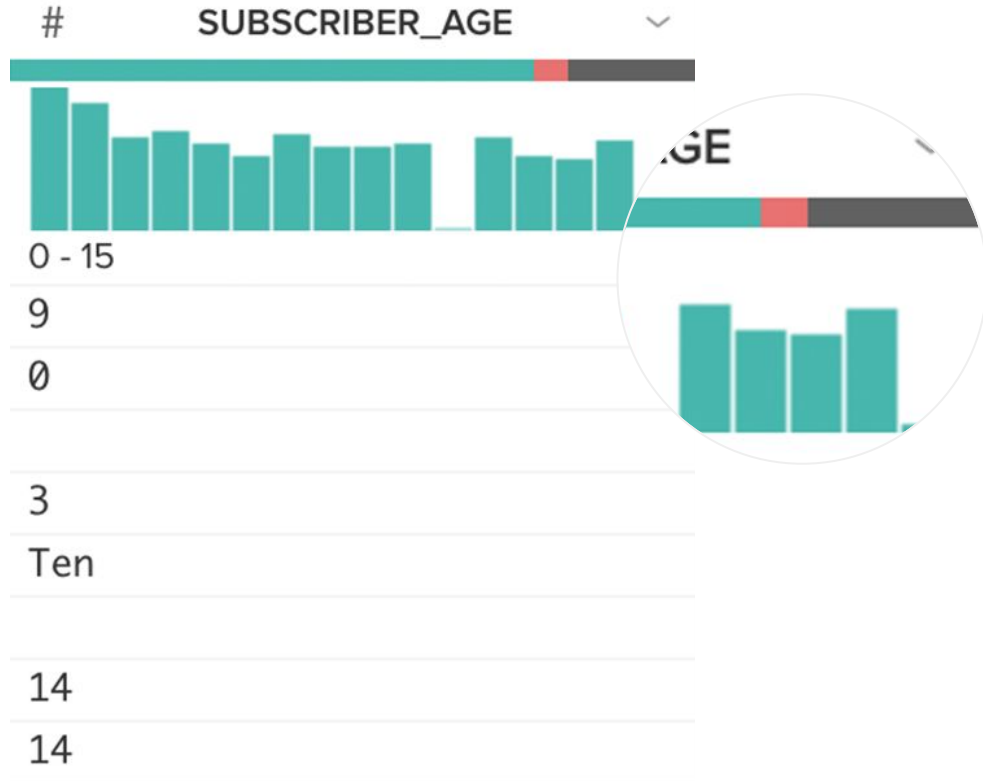
Pattern profiling combines
automatic pattern discovery
with interactive visualizations

Understanding data through profiling



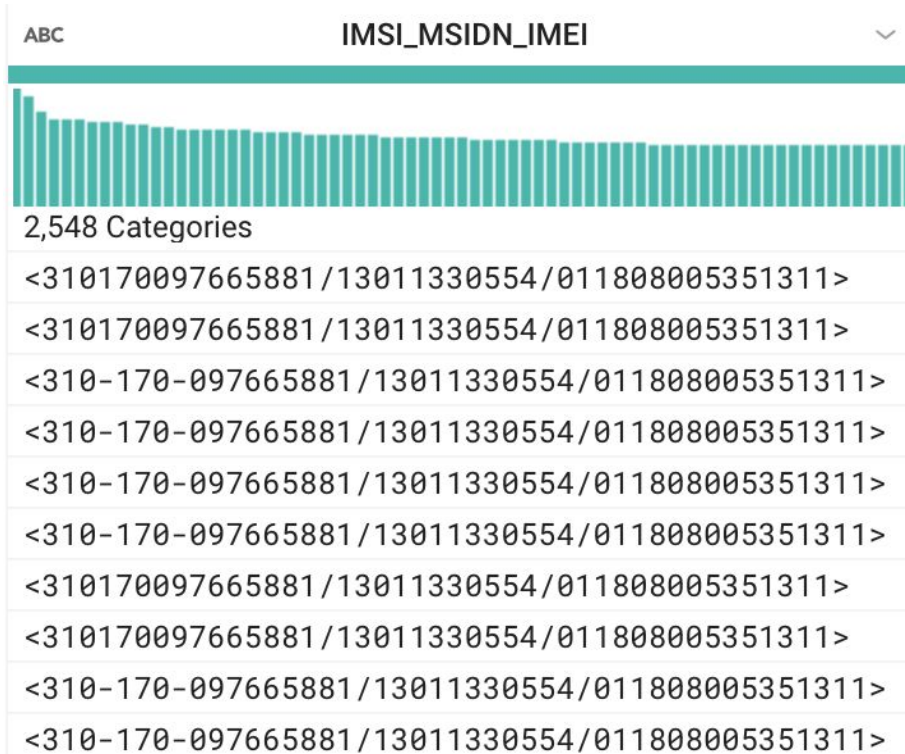
For **ordinal data** (numbers, datetimes, ...), profiles can compactly and efficiently convey **data distribution**.

Understanding data through profiling



For **well typed data**, profiles can indicate the prevalence of **valid**, **invalid**, and **null** values.

The nasty case of text



Categorical profiles fall back to **top k values**.

Data quality can only differentiate between null and non-null values.

Understanding data through patterns

`{num}(3){num}(3)-{num}(4)`

(852)519-0903

`{num}(3){num}(7)`

(717)4412597

`{num}(6)-{num}(4)`

582616-6989

`{num}(10)`

8629016974

Other patterns

Through **pattern based representations** of text, we can see **common** and **anomalous** patterns and drill down to specific records more easily.

Pattern profiling

What?

- Automatically detects and displays formatting patterns within a column
- Visually summarizes content of a column into common and anomalous patterns

The screenshot shows a data tool interface with a sidebar on the left containing column filters for 'column1', 'First_Name', 'Last_Name', 'Country', 'State_Province', 'Lead_Owner', 'Phone', 'Email', 'Title', 'Company_Account', 'Annual_Revenue', 'Lead_Source', and 'RegistrationTime'. The main area displays a 'Phone' column with a pattern detection panel overlaid. The panel shows detected patterns and their corresponding data values:

- Pattern: `digit 3 - digit 3 - digit 4`
Data: 468491716, 12192085679, 60507361
- Pattern: `digit 1 . digit 5 upper + digit 2`
Data: 4.42089E+11, 3.53876E+11

The interface also includes a 'Transform Builder' section at the bottom left with a 'Choose a transformation' dropdown set to 'TransformFormat()'. The top right of the window shows 'Run Job' and 'Column Details' buttons.

Pattern profiling

Why?

- Users can quickly understand and correct discrepancies within each column
- Provides starting point for users to identify and select subsets of records to transform

The screenshot shows a data profiling tool interface. The main window displays a 'Phone' column with a pattern profile. The profile shows a distribution of phone numbers across different patterns. A lightbulb icon indicates a suggested transformation.

Phone

Overview Patterns

All patterns

- digit 1
- digit 3 - digit 3
- digit 5 - digit 5
- (digit 3) - digit 1

468491716
12192085679
60507361

digit 3 - digit 3 - digit 4

817-740-9000
949-727-6421
617-329-1060
425-643-4408
703-123-1234

digit 1 . digit 5 upper + digit 2

4.42089E+11
3.53876E+11

Pattern profiling

How?

- Cluster records into meaningful sub-groups
- Users interact with sub-groups and example records
- Predict transformations to apply across the data

The screenshot shows a data tool interface for a file named 'sfdc_report.csv'. The 'Phone' column is selected, and the 'Patterns' tab is active. The interface displays a pattern bar and a list of identified patterns. A detailed view of a pattern is shown, including example records and suggested transformations.

Patterns identified:

- digit 1 - 49 records (12.4%)
- digit 3 - digit 3 - digit 4 - 49 records (12.4%)
- digit 1 - digit 5 - 49 records (12.4%)

Example records for the 'digit 3 - digit 3 - digit 4' pattern:

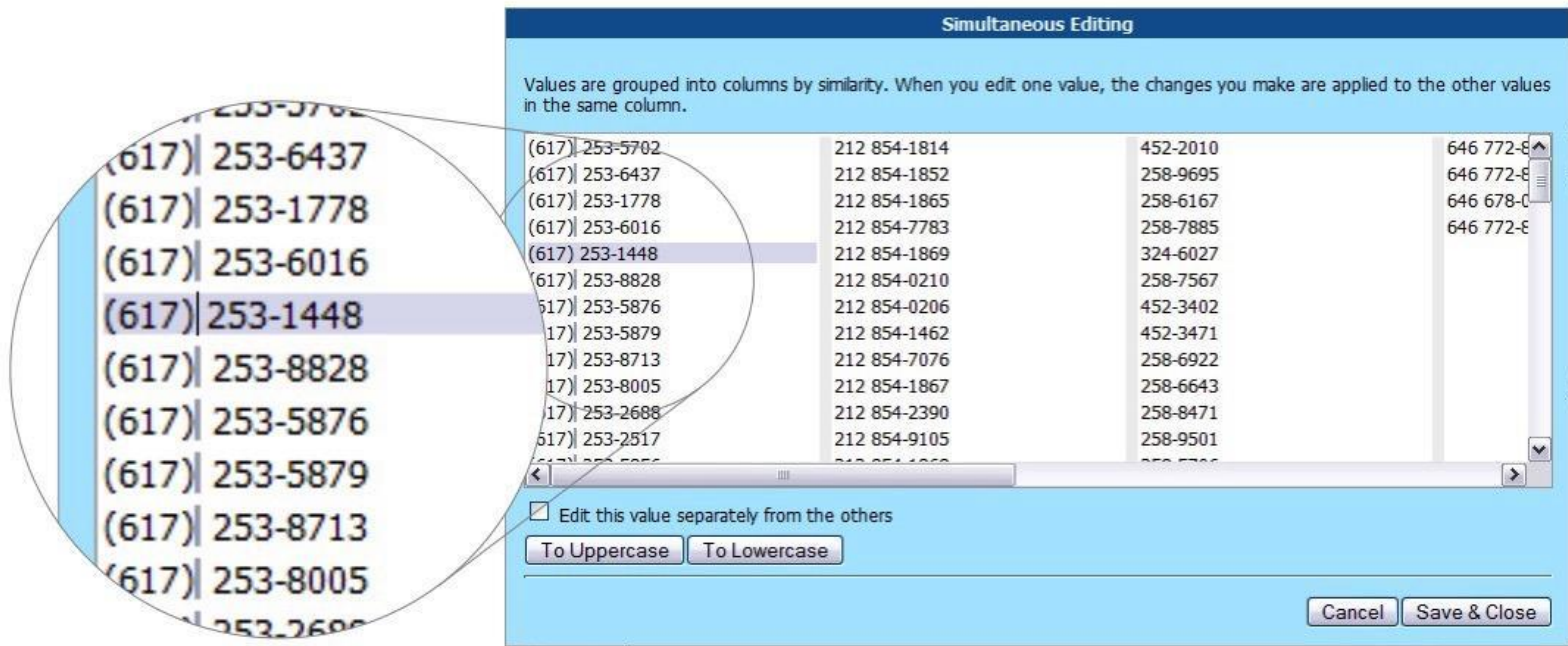
- 468491716
- 12192085679
- 60507361
- 817-740-9000
- 949-727-6421
- 617-329-1060
- 425-643-4408
- 703-123-1234

Suggested transformations for the 'digit 3 - digit 3 - digit 4' pattern:

- digit 1 . digit 5 upper + digit 2

Example records for the suggested transformation:

- 4.42089E+11
- 3.53876E+11

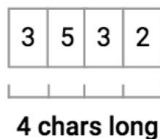


Hyunh, Miller, and Karger, ISWC 2007.
Potluck: Data Mash-Up Tool for Casual Users.

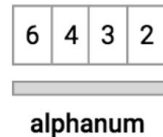
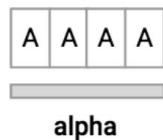
Demo

What is in a pattern?

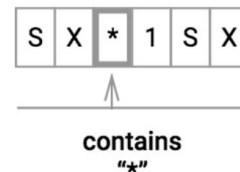
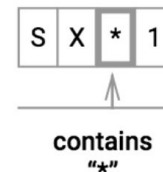
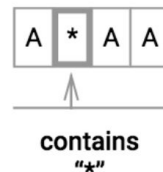
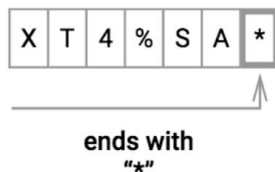
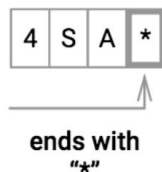
Length



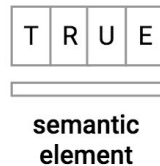
Token



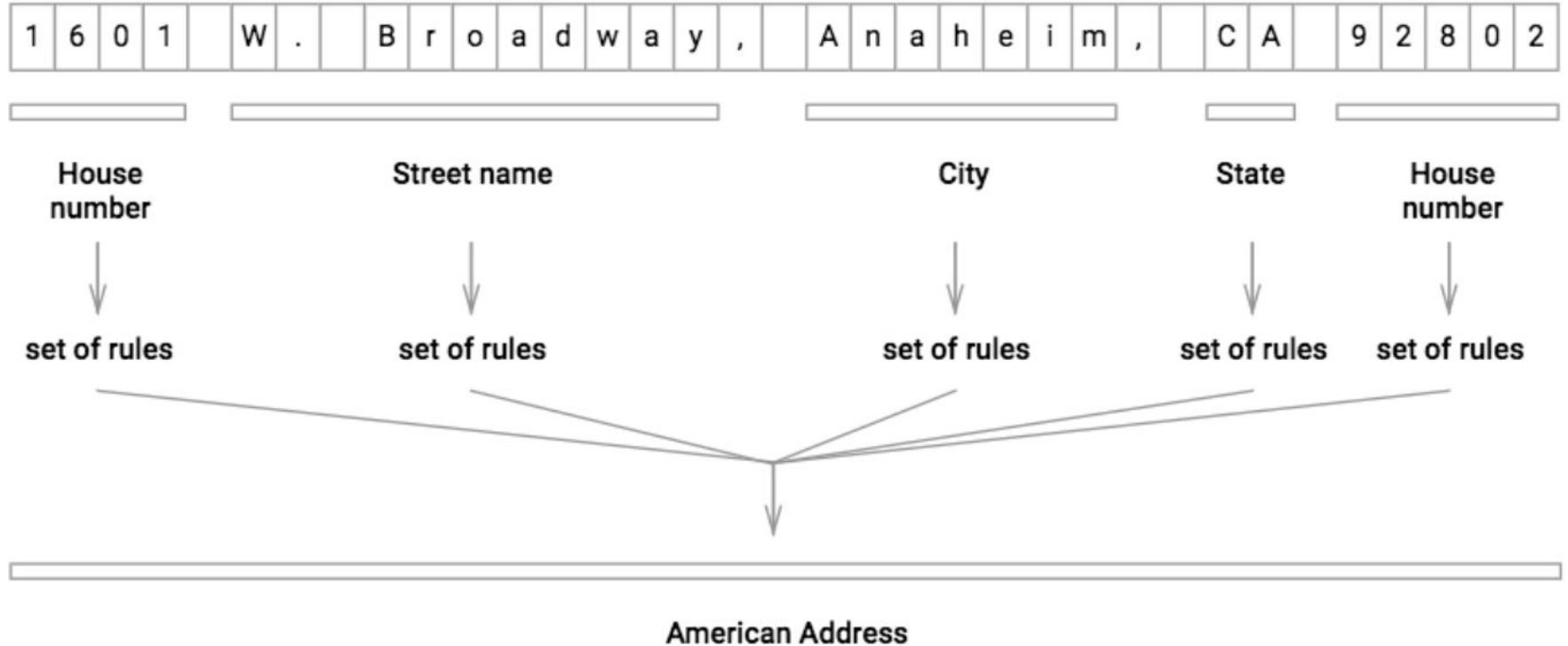
Position



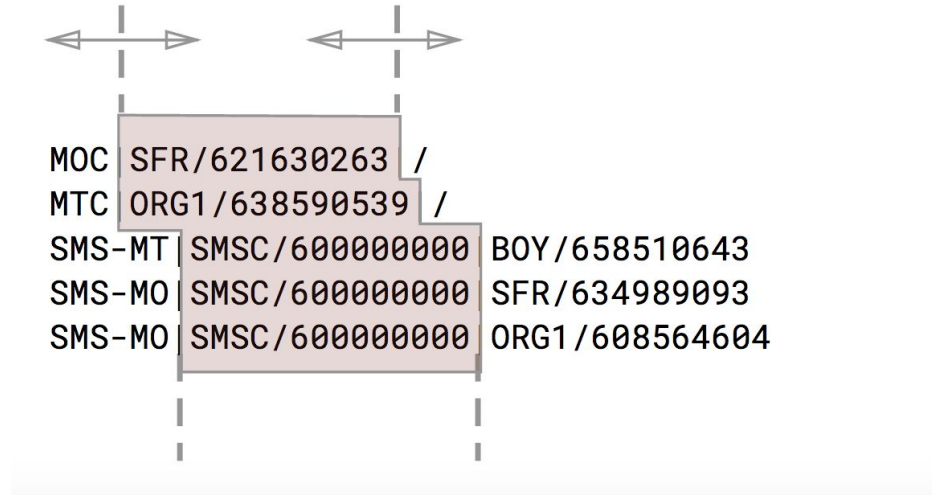
Semantic
meaning



What is in a pattern?



Capturing semi-structure with patterns

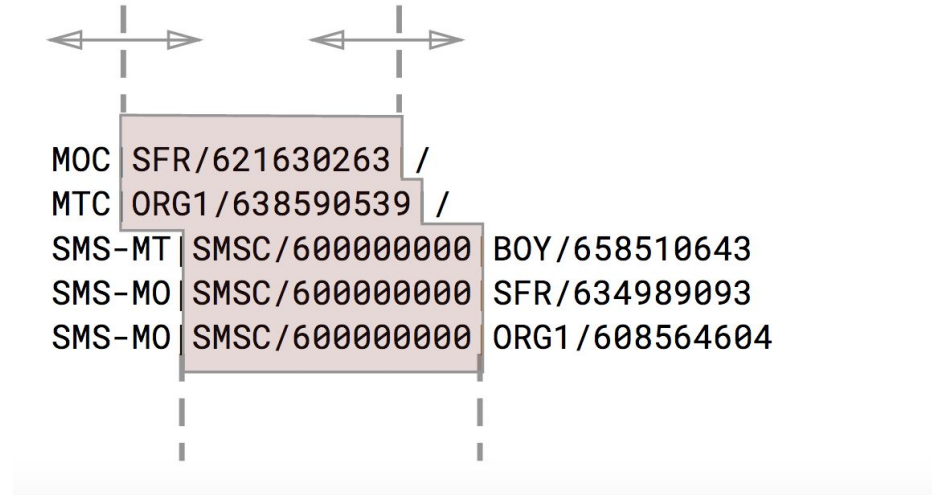


`[A-Z]+(\-[A-Z]+)?\ | [A-Z0-9]+/[0-9]{9}/([A-Z]+/[0-9]+)?` (all **five** rows)

`[A-Z]+|[A-Z0-9]+/[0-9]{9}/` (first **two** rows)

`[A-Z]+\-[A-Z]+|[A-Z0-9]+/[0-9]{9}/[A-Z]+/[0-9]+` (last **three** rows)

Capturing semi-structure with patterns

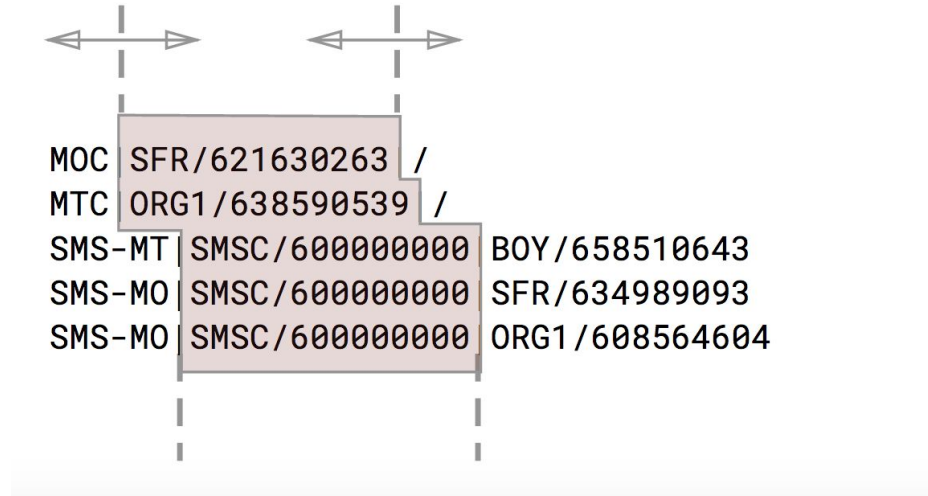


...<Product Code>...

[A-Z]+|<Product Code>/

[A-Z]+\-[A-Z]+|<Product Code>/[A-Z]+/[0-9]+

Making sense of hierarchical pattern structures



...<Product Code>... (union of **two** structures)

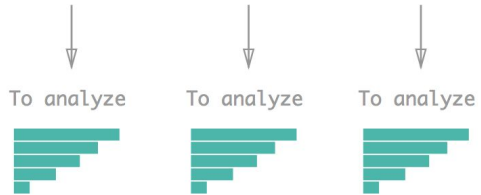
[A-Z]+|<Product Code>/

[A-Z]+\-[A-Z]+|<Product Code>/[A-Z]+/[0-9]+

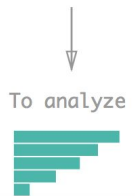
Making sense of hierarchical pattern structures

```
<604711647/208100942278779/44928067108241>  
<604523376/208102203151835/44828688676508>  
<600225657/208102531594906/44926909793892>  
<600262490/208113678616485/35328600036554>  
<603028486/208106424063363/35504800190986>  
<604508753/208118357396563/44919238527861>
```

--- {600}{num}(6) || {603}{num}(6) || {604}{num}(6)



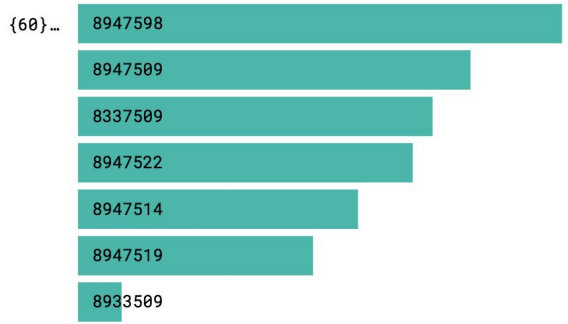
or
--- {60}{num}(7)



Abstract —————> Concrete

Cluster more **concrete**, lower-level patterns into more **abstract**, higher-level representations to give overviews of data.

Help the user transition from the higher-level **token** and **wild-card** representations to the **literal** data they encounter.



Examples as links between abstract and concrete

digit 4 / digit 2 / digit 2 · time

digit 4 / digit 2 / digit 2 · digit 2 : digit 2 : digit 2

2011/03/15·04:25:26
2011/08/15·09:36:42
2014/11/05·20:17:30
2010/11/10·13:34:11
2013/04/30·18:50:20

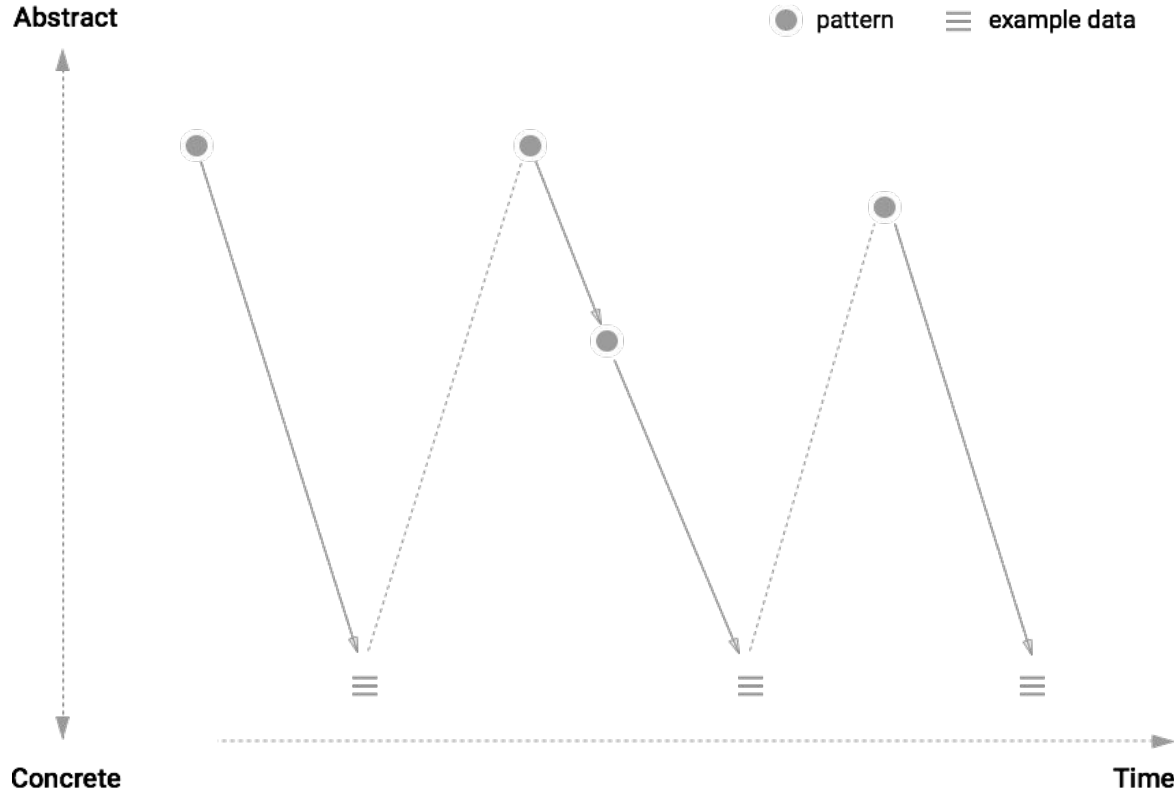
Abstract



Concrete

Example data can help users traverse the hierarchy of patterns and understand their context.

Examples as links between abstract and concrete



Inspiration from PADS

PADS (processing arbitrary data streams) is a data description language.

Format descriptions written in PADS are then used to process and structure data.

For more see:

* Fisher and Gruber 2003, PADS: Processing Arbitrary Data Streams.

* Fisher *et al.* 2008, From Dirt to Shovels: Fully Automatic Tool Generation from Ad Hoc Data.

* Xi *et al.* 2009, Ad Hoc Data and the Token Ambiguity Problem.

$c ::= a \mid i \mid s$ (constants)
 x (variables)
 $p ::= c \mid x$ (parameters)

Base types $b ::=$

Pint	(generic, unrefined integer)
PintRanged	(integer with min/max values)
Pint32	(32-bit integer)
Pint64	(64-bit integer)
PintConst	(constant integer)
Pfloat	(floating point number)
Palpha	(alpha-numeric string)
Pstring	(string; terminating character)
PstringFW	(string; fixed width)
PstringConst	(constant string)
Pother	(punctuation character)
ComplexB	(complex base type defined by regexp; e.g. date, time, etc.)
Pvoid	(parses no characters; fails immediately)
Pempty	(parses no characters; succeeds immediately)

Types $T ::=$

$b(p_1, \dots, p_k)$	(parameterized base type)
$x:b(p_1, \dots, p_k)$	(parameterized base type; underlying value named x)
struct $\{T_1; \dots T_k;\}$	(fixed sequence of items)
array $\{T;\}$	(array with unbounded repetitions)
arrayFW $\{T;\}[p]$	(array; fixed length)
arrayST $\{T;\}[sep,term]$	(array; separator and terminator)
union $\{T_1; \dots T_k;\}$	(alternatives)
enum $\{c_1; \dots c_k;\}$	(enumeration of constants)
$x:enum \{c_1; \dots c_k;\}$	(enumeration of constants; underlying value named x)
option $\{T;\}$	(type T or nothing)
switch x of $\{c_1 \Rightarrow T_1; \dots c_k \Rightarrow T_k;\}$	(dependent choice)

Representations of parsed data $d ::=$

c	(constant)
$in_i(d)$	(injection into the i^{th} alternative of a union)
(d_1, \dots, d_k)	(sequence of data items)

Figure 1. Components of the IR (Fisher *et al.* 2008).

From dirt to shovels

Aims to **fully automate** PADS format generation from input data.

Does so by **tokenizing** the input data, **discovering structure** across tokenized records, and **refining** until minimal spec reached.

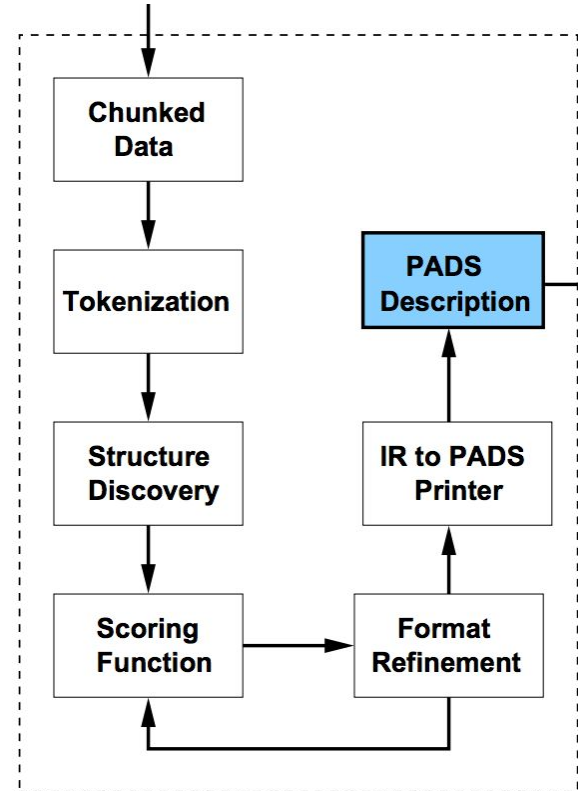


Figure 4. Architecture of the automatic tool-generation engine (Fisher *et al.* 2008).

Tokenization and lexing

Wildcard
{any}

Semantic Tokens
{ip-address}
{time}
{hashtag}
{email}

Character Tokens
{alpha-numeric}
{alpha}
{upper}
{lower}
{digit}
{delim}

Literal Tokens
{A-Za-z}
{0-9}
{/}
{-}
{_}
{|}
{,}
{ }
{<}
{>}

Specify a token tree going from most general (wildcard) to most specific (literal). Where possible, include parent child relationships.

Tokenization and lexing

192.168.2.255:GET

193.168.3.344:PUT

193.145.13.45:POST

{digit}{3}{delim}{digit}{3}{delim}{digit}{delim}{digit}{3}

{:}{upper}{3}

{digit}{3}{delim}{digit}{3}{delim}{digit}{2}{delim}{digit}{2}

{:}{upper}{4}

Pattern aggregation

Given a **pair of patterns** describing the data, determine how to **combine** and **generalize**.

Pattern aggregation

Given a **pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Structure discovery informs aggregation

Determine for a set of tokens S and data D which tokens are union tokens and which tokens are struct tokens.

Structure discovery informs aggregation

Determine for a set of tokens S and data D which tokens are union tokens and which tokens are struct tokens.

Given tokens $\{\mathbf{a}\}, \{\mathbf{b}\}$, a union relationship implies $\{\mathbf{a}\} | \{\mathbf{b}\}$ and a struct relationship implies $(\{\mathbf{a}\} \{\mathbf{b}\})$.

Structure discovery informs aggregation

Determine for a set of tokens S and data D which tokens are union tokens and which tokens are struct tokens.

```
for each token T in S:  
  define histogram H[T] = distribution of number of matches of T over D  
  
for each histogram H1 in H:  
  for each histogram H2 in H:  
    define E[H1][H2] = symmetric_relative_entropy(H1, H2)  
  
define clusters C = agglomerative clustering with distance metric E
```

For more on structure discovery see:

Fisher *et al.* 2008, From Dirt to Shovels: Fully Automatic Tool Generation from Ad Hoc Data.

Structure discovery informs aggregation

Determine for a set of tokens S and data D which tokens are union tokens and which tokens are struct tokens.

for each cluster $C1$ in C :

if histograms [... $C1$] have high coverage and narrow distribution:

$C1$ is struct

if histograms [... $C1$] have lower coverage and wider distribution:

$C1$ is union

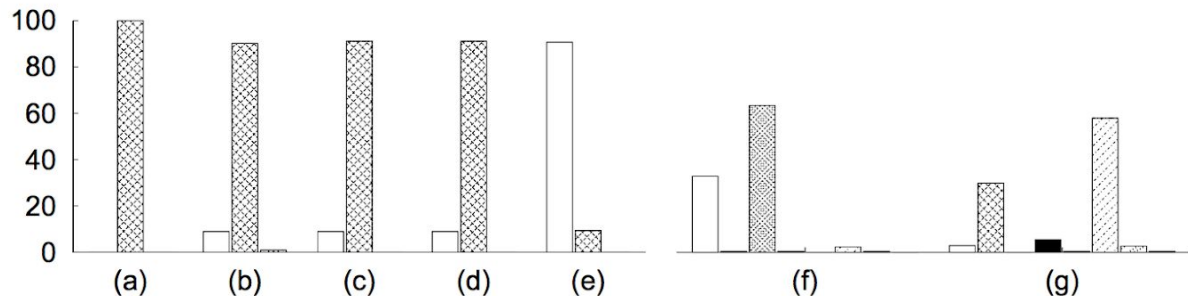


Figure 6. (a)-(e) are struct tokens, (f)-(g) are union tokens (Fisher *et al.* 2008).

Pattern aggregation

Given a **pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Align tokens (longest common subsequence) taking into account union and struct tokens.

Pattern aggregation

Given a **pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Align tokens (longest common subsequence) taking into account union and struct tokens.

Use **token hierarchy** to combine non-aligned tokens.

Token hierarchy informs aggregation

$$\left\{ \begin{array}{l} \{\text{upper}\}, \{\text{lower}\}, \{\text{alpha}\}, \{\text{foo}\}, \{\text{bar}\}, \{\text{foobar}\} \\ \{\text{upper}\} | \{\text{lower}\} = \{\text{alpha}\} \\ \{\text{foo}\} | \{\text{bar}\} = \{\text{foobar}\} \end{array} \right\} \text{ IR}$$

With $\{\text{upper}\}\{\text{lower}\}$, $\{\text{foo}\}\{\text{bar}\}$ as union clusters

Align $\{\text{upper}\}\{\text{foo}\}$
 $\{\text{lower}\}\{\text{bar}\}$

Use semantic token hierarchy from IR $\{\text{upper}\} | \{\text{lower}\} = \{\text{alpha}\}$, $\{\text{foo}\} | \{\text{bar}\} = \{\text{foobar}\}$

$\{\text{upper}\}\{\text{foo}\}$
+ $\{\text{lower}\}\{\text{bar}\}$
= $\{\text{alpha}\}\{\text{foobar}\}$

Pattern aggregation

Given a **pair of patterns** describing the data, determine how to **combine** and **generalize**.

Use **structure discovery** to suggest possible struct and union tokens.

Use **token alignment** (longest common subsequence) along struct and union tokens to find shared structure.

Use **token hierarchy** to combine non-aligned tokens.

Score candidate aggregations.

Scoring candidate aggregations

```
define score(pattern):  
    n_wildcards = count([t is {any} for t in pattern])  
    n_kleene = count([t has {*} for t in pattern])  
    n_semantics = length(intersection(semantic_tokens, pattern))  
    n_basic = length(intersection(basic_tokens, pattern))  
    n_struct = length(intersection(struct_tokens, pattern))  
  
    score = -(w1)(n_wildcards) - (w2)(n_kleene) - (w3)n_semantics +  
            (w4)(n_basic) + (w5)(n_struct)  
  
    return score / length(pattern)
```

Pattern aggregation continued...

```
for each pattern {a} in patterns:
  for each pattern {b} in patterns:
    {parent}, score = combine_patterns({a}, {b})
    scores[{a}][{b}] = {{parent}, score}

sort scores desc

for each {a}, {b}, {parent}, score in scores:
  # if disjoint (neither child has found a better parent)
  if {a}, {b} in patterns:
    remove {a}, {b} from patterns
    add {parent} to patterns

# stop at the {root} pattern
repeat until only one pattern present
```

Generating interactive examples

```
digit 4 / digit 2 / digit 2 . digit 2 : digit 2 : digit 2
```

```
2011/03/15·04:25:26  
2011/08/15·09:36:42  
2014/11/05·20:17:30  
2010/11/10·13:34:11  
2013/04/30·18:50:20
```

For each set of candidate patterns, run **conditional aggregate** queries to accumulate example records.

```
digit 2 - digit 2 - digit 4 upper digit 2 : digit 2 : digit 2
```

```
07-10-2013T23:55:05  
05-16-2012T19:06:55  
10-17-2014T17:12:40  
04-26-2011T13:47:40  
04-30-2014T23:51:21
```

digit 2 - digit 2 \rightarrow ($[\text{0-9}]\{2\}$)(-)($[\text{0-9}]\{2\}$)
\$n captures the nth token group

Interact within a pattern by using **capturing groups** and matching across example records.

Extensions

Injecting supervision

Injecting Supervision

Tuning aggregation scoring through **human-in-the-loop feedback**

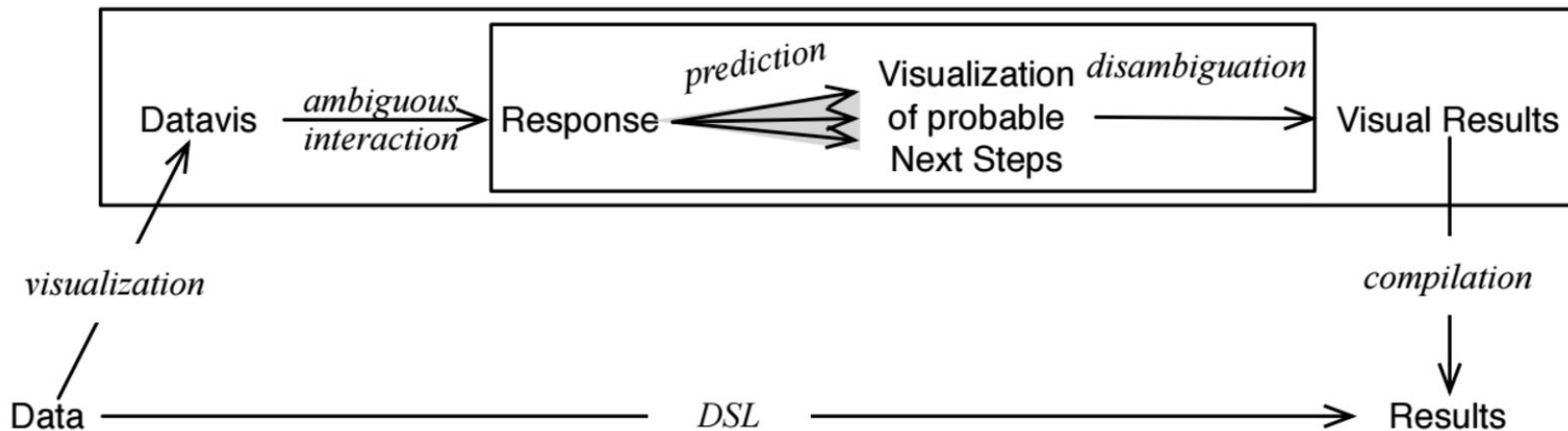


Figure 6. Predictive Interaction: The Guide / Decide Loop (Heer, Hellerstein, Kandel 2015).

Injecting Supervision

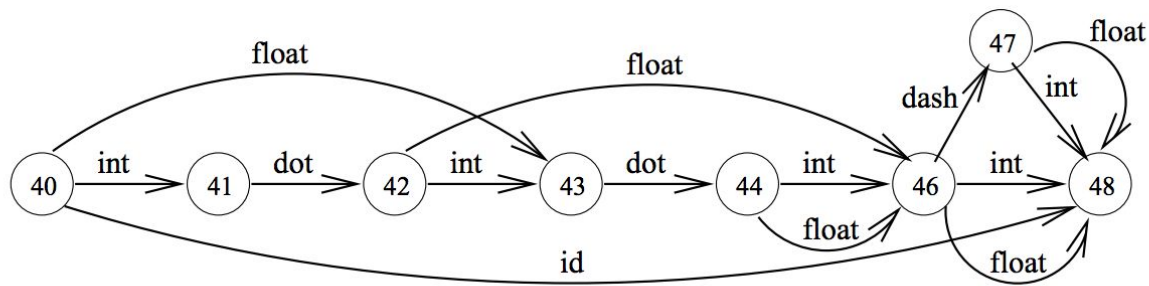
The **token ambiguity** problem

192.168.2.255

Option 1: int '.' int '.' int '.' int

Option 2: float '.' float

Option 3: ip-address



Xi *et al.* train HMMs and Hierarchical SVMs to traverse the tokenization paths, assign probabilities, pick best tokenization (2009).

Figure 2. SeqSet from parsing "2.2-13.4" (Xi *et al.* 2009).

Extensions

Injecting supervision

Improving performance

Thanks!

Michael Minar,
Athena Jiang,
Anish Doshi,
Lionel Michel,
and others @trifacta



TRIFACTA