USC

USC Viterbi
School of Engineering

Automatic Source Modeling

Jose Luis Ambite
University of Southern California

Based on slides by Mark Carman, Craig Knoblock
Jose Luis Ambite and Kristina Lerman

Source Modeling vs. Schema Matching

- Schema Matching/Mapping
  - Align schemas between data sources
  - Assumes static sources and complete access to data
  - Schema Mapping does provide queries to map between sources
- Source Modeling
  - Incrementally build models from partial data (e.g., web services, html forms, programs)
  - Model not just the fields but the source types and even the function of a source
  - Support richer source models (a la Semantic Web) that describe the contents of sources

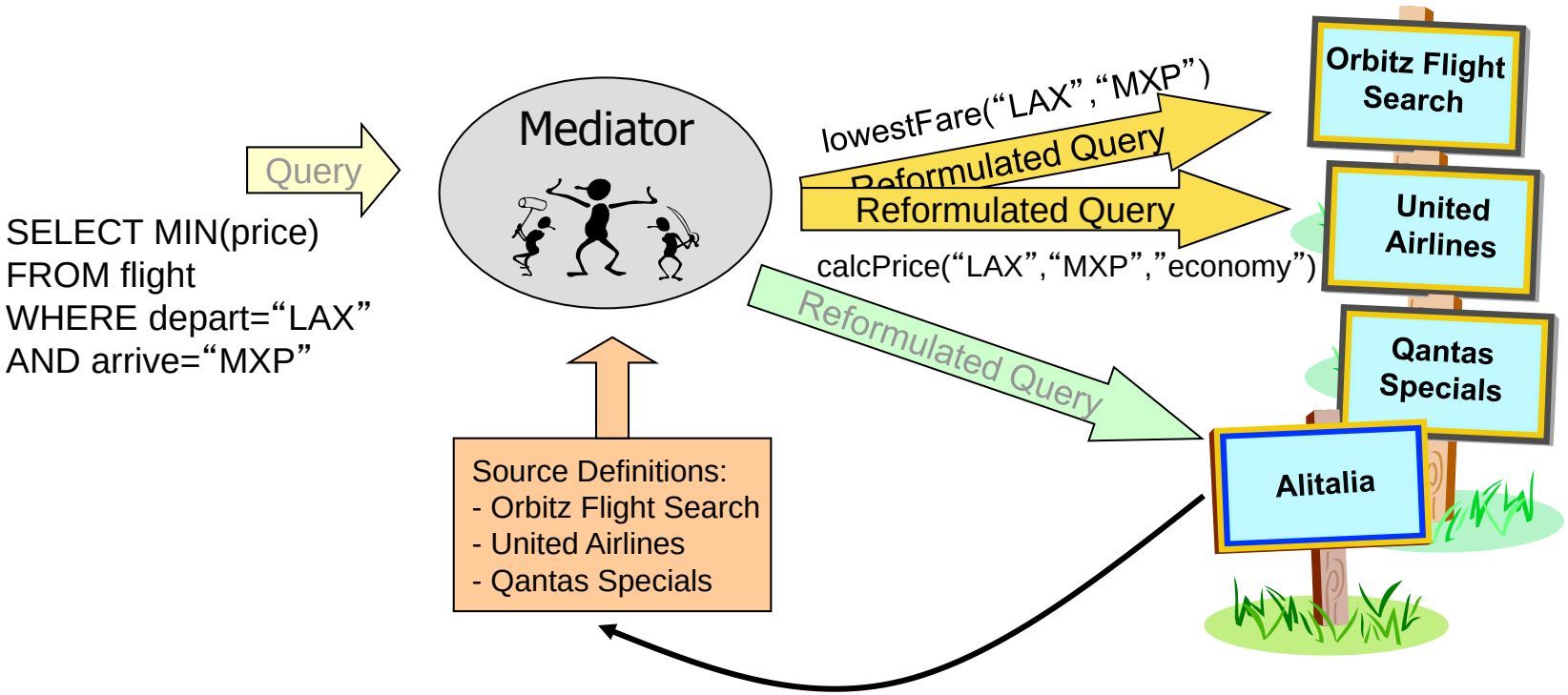
Overview

- **Learning Semantic Descriptions of Web Information Sources**
  - Integrated approach to learning semantic definitions of a source
  - Needs to invoke the source
  - Learn Datalog description of the source: LAV mapping

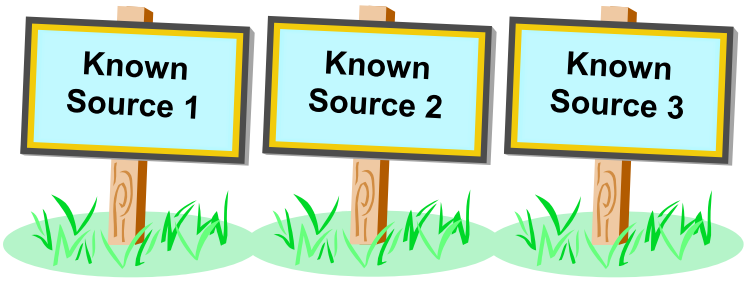
- **Automatically Constructing Semantic Web Services from Online Sources**
  - Deimos: End to end system for discovering and modeling sources
  - Uses system above to automatically discover semantic definitions of discovered sources

Mediators Require Source Definitions

- New service => no source definition!
- Can we discover a definition automatically?

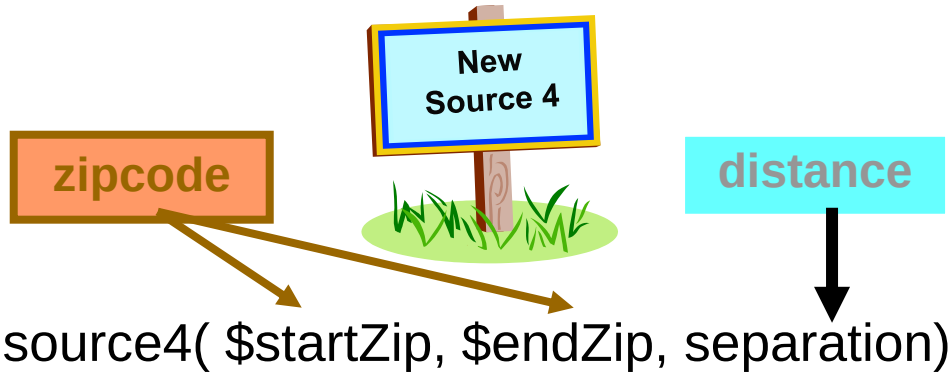


Inducing Source Definitions by Example

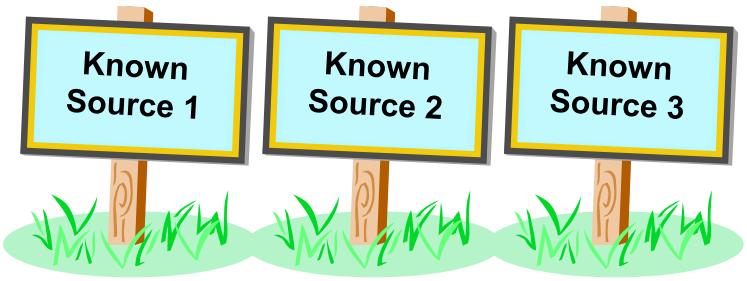



```
source1($zip, lat, long) :-
    centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
    greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
    convertKm2Mi(dist1, dist2).
```

- Step 1: classify input & output semantic types



Inducing Source Definitions - Step 2



source1($zip, lat, long) :-
centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
convertKm2Mi(dist1, dist2).

- Step 1: classify input & output semantic types
- Step 2: generate plausible definitions

source4($zip1, $zip2, dist):-
source1(zip1, lat1, long1),
source1(zip2, lat2, long2),
source2(lat1, long1, lat2, long2, dist2),
source3(dist2, dist).

source4($zip1, $zip2, dist):-
centroid(zip1, lat1, long1),
centroid(zip2, lat2, long2),
greatCircleDist(lat1, long1, lat2, long2, dist2),
convertKm2Mi(dist1, dist2).

Inducing Source Definitions – Step 3

- Step 1: classify input & output semantic types
- Step 2: generate plausible definitions
- Step 3: invoke service & compare output

```
source4($zip1, $zip2, dist):-
    source1(zip1, lat1, long1),
    source1(zip2, lat2, long2),
    source2(lat1, long1, lat2, long2, dist2),
    source3(dist2, dist).
```

```
source4($zip1, $zip2, dist):-
    centroid(zip1, lat1, long1),
    centroid(zip2, lat2, long2),
    greatCircleDist(lat1, long1, lat2, long2, dist2),
    convertKm2Mi(dist1, dist2).
```

match

$zip1	$zip2	dist *(actual)*	dist *(predicted)*
80210	90266	842.37	843.65
60601	15201	410.31	410.83
10005	35555	899.50	899.21

Searching for Definitions

- Search space of *conjunctive queries:*

  target(X) :- source1(X_1), source2(X_2), …
- For scalability don't allow negation or union
- Perform Top-Down Best-First Search

Expressive Language
Sufficient for modeling most online sources

1. First sample the New Source

Invoke ***target*** with set of random inputs;
Add empty clause to ***queue***;
while (***queue*** not empty)
v := best definition from ***queue***;
forall (***v'*** in **Expand(*v*)**)
if (**Eval(*v'*)** > **Eval(*v*)**)
insert ***v'*** into ***queue***;

2. Then perform best-first search through space of candidate definitions

Invoking the Target

Generate Input Tuples: *<zip1, dist1>*

Invoke

source5($zip1, $dist1, zip2, dist2)

Invoke source with *representative* values

- Try randomly generating input tuples:
  - Combine examples of each type
  - Use distribution if available

Randomly Combined Example Values

Input <zip1, dist1>	**Output** <zip2, dist2>
<07307, 50.94>	{<07097, 0.26>, <07030, 0.83>, <07310, 1.09>, ...}
<60632, 10874.2>	{}

Non-empty Result

Empty Result

Invoking the Target

Generate Input Tuples: *<zip1, dist1>*

Invoke

source5($zip1, $dist1, zip2, dist2)

Invoke source with *representative* values

- Try randomly generating input tuples:
  - Combine examples of each type
  - Use distribution if available
- If *only empty invocations* result
  - Try *invoking other sources* to generate input
- Continue until sufficient non-empty invocations result

Top-down Generation of Candidates

Start with empty clause & generate specialisations by

- Adding one predicate at a time from set of sources
- Checking that each definition is:
  - Not logically redundant
  - Executable (binding constraints satisfied)

source5($zip1,$dist1,zip2,dist2)

source5(_,_,_,_).

Expand

source5(zip1,_,_,_) :- **source4**(zip1,zip1,_).
source5(zip1,_,zip2,dist2) :- **source4**(zip2,zip1,dist2).
source5(_,dist1,_,dist2) :- **<**(dist2,dist1).
…

Best-first Enumeration of Candidates

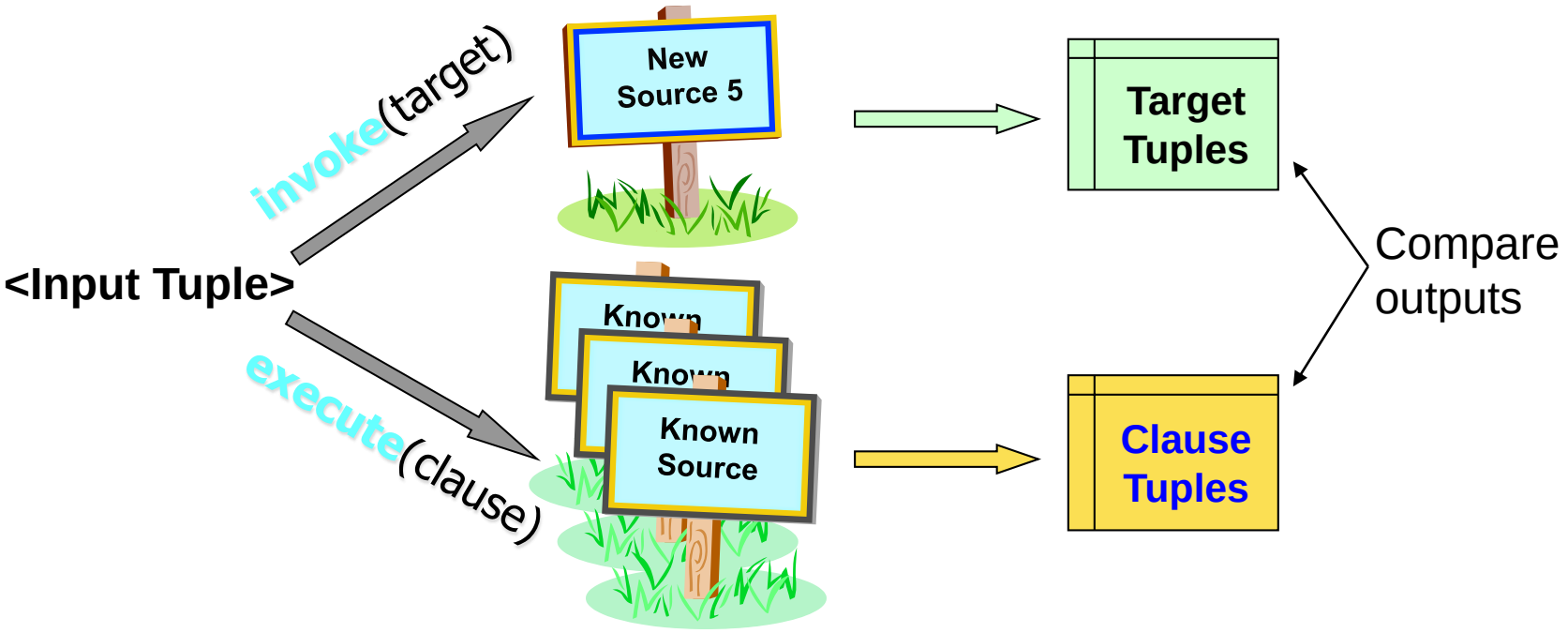
- Evaluate each clause produced
- Then expand best one found so far
- Expand high-arity predicates incrementally

source5(zip1,_,zip2,dist2) :- **source4**(zip2,zip1,dist2).

Expand

source5(zip1,dist1,zip2,dist2) :- **source4**(zip2,zip1,dist2), **source4**(zip1,zip2,dist1).
source5(zip1,dist1,zip2,dist2) :- **source4**(zip2,zip1,dist2), **<**(dist2,dist1).
…

Evaluating Candidates



- Compare output of clause with that of target.
- Average the results across different input tuples.

Evaluating Candidates II

Candidates may return multiple tuples per input

- Need measure that compares sets of tuples!

Input <$zip1, $dist1>	Target Output <zip2, dist2>	Clause Output <zip2, dist2>
<60632, 874.2>	{}	{<60629, 2.15>, <60682, 2.27>, <60623, 2.64>, ..}
<07307, 50.94>	{<07097, 0.26>, <07030, 0.83>, <07310, 1.09>, ...}	{}
<28041, 240.46>	{<28072, 1.74>, <28146, 3.41>, <28138, 3.97>,...}	{<28072, 1.74>, <28146, 3.41>}

No Overlap

No Overlap

Overlap!

Approximating Equality

Allow flexibility in values from different sources

- Numeric Types like *distance*

  10.6 km ≈ 10.54 km

  Error Bounds (eg. +/- 1%)

- Nominal Types like *company*

  Google Inc. ≈ Google Incorporated

  String Distance Metrics (e.g. JaroWinkler Score > 0.9)

- Complex Types like *date*

  Mon, 31. July 2006 ≈ 7/31/06

  Hand-written equality checking procedures.

Next …

Automatically build semantic models for data and services available on the larger Web

- Construct models of these sources that are sufficiently rich to support querying and integration
- Current focus:
  - Build models for the vast amount of structured and semi-structured data available
    - *Not just web services, but also form-based interfaces*
    - *E.g., Weather forecasts, flight status, stock quotes, currency converters, online stores, etc.*
  - Learn models for information-producing web sources and web services

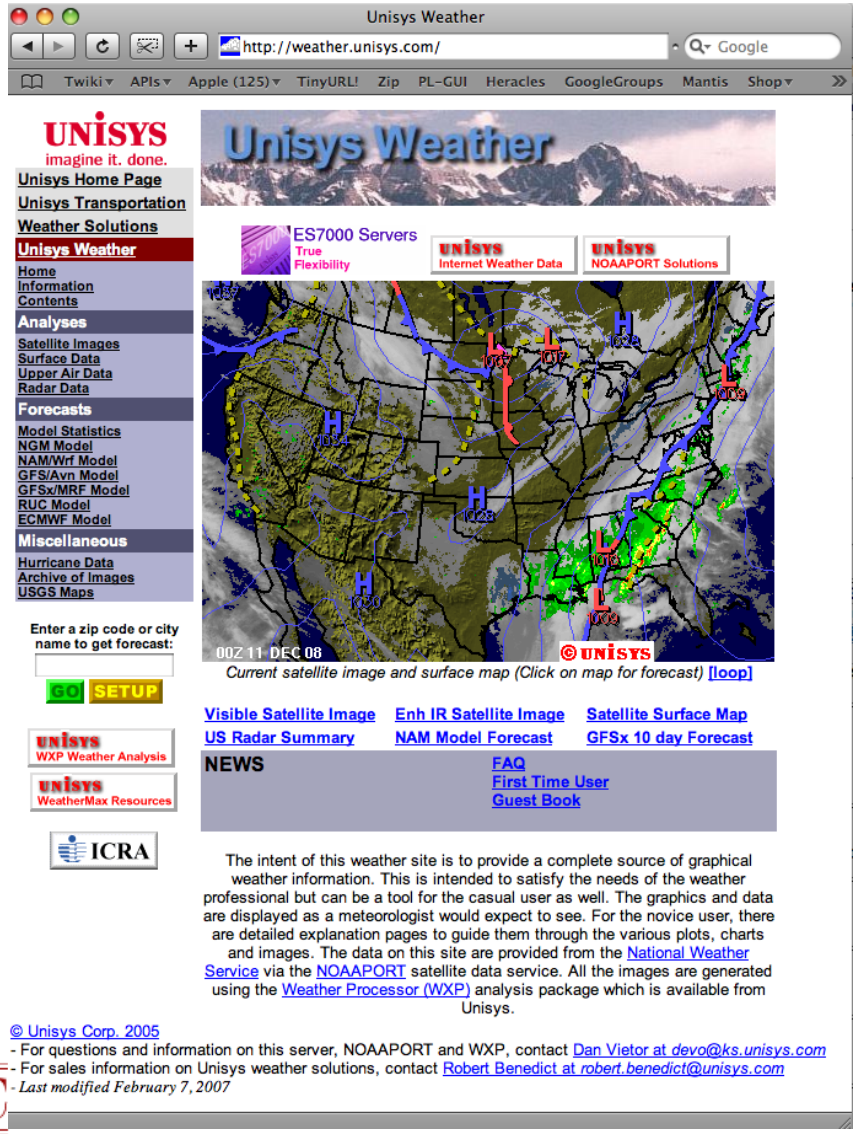
Integrated Approach

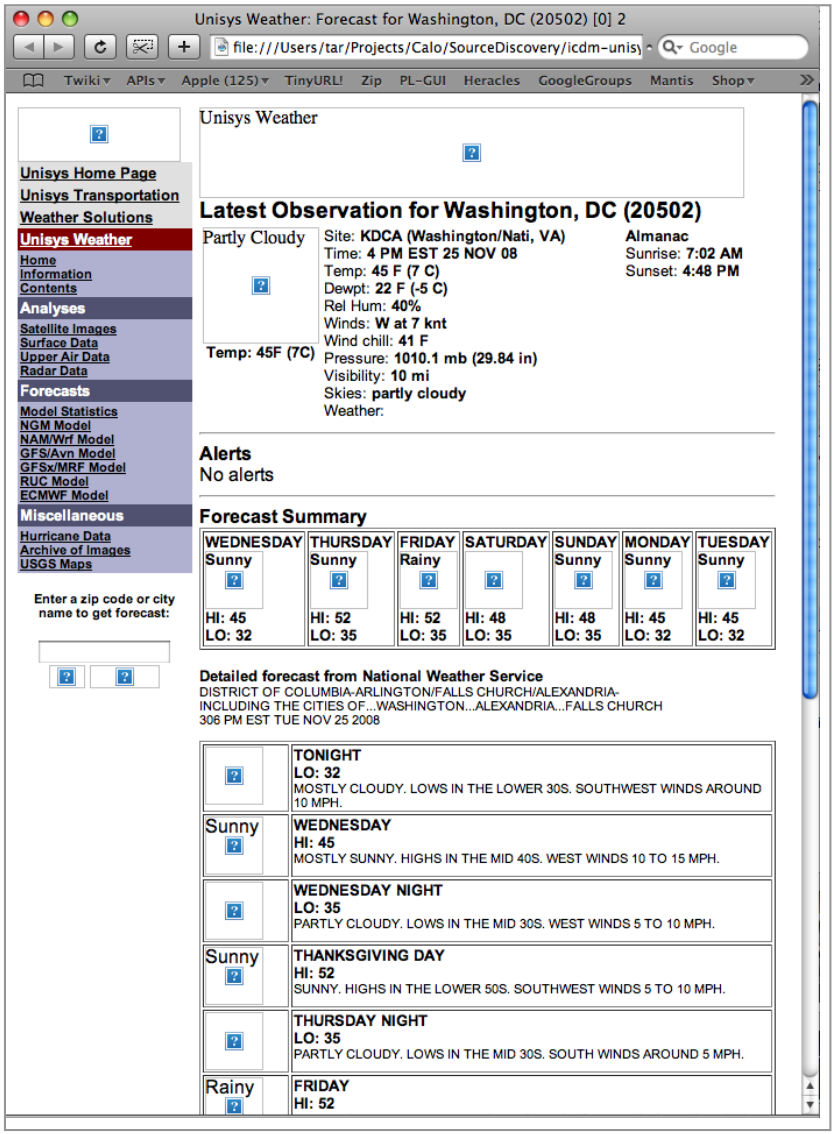
- Start with an some initial knowledge of a domain
  - Sources and semantic descriptions of those sources
- Automatically
  - Discover related sources
  - Determine how to invoke the sources
  - Learn the syntactic structure of the sources
  - Identify the semantic types of the data
  - Build semantic models of the source
  - Construct semantic web services

Seed Source

USC

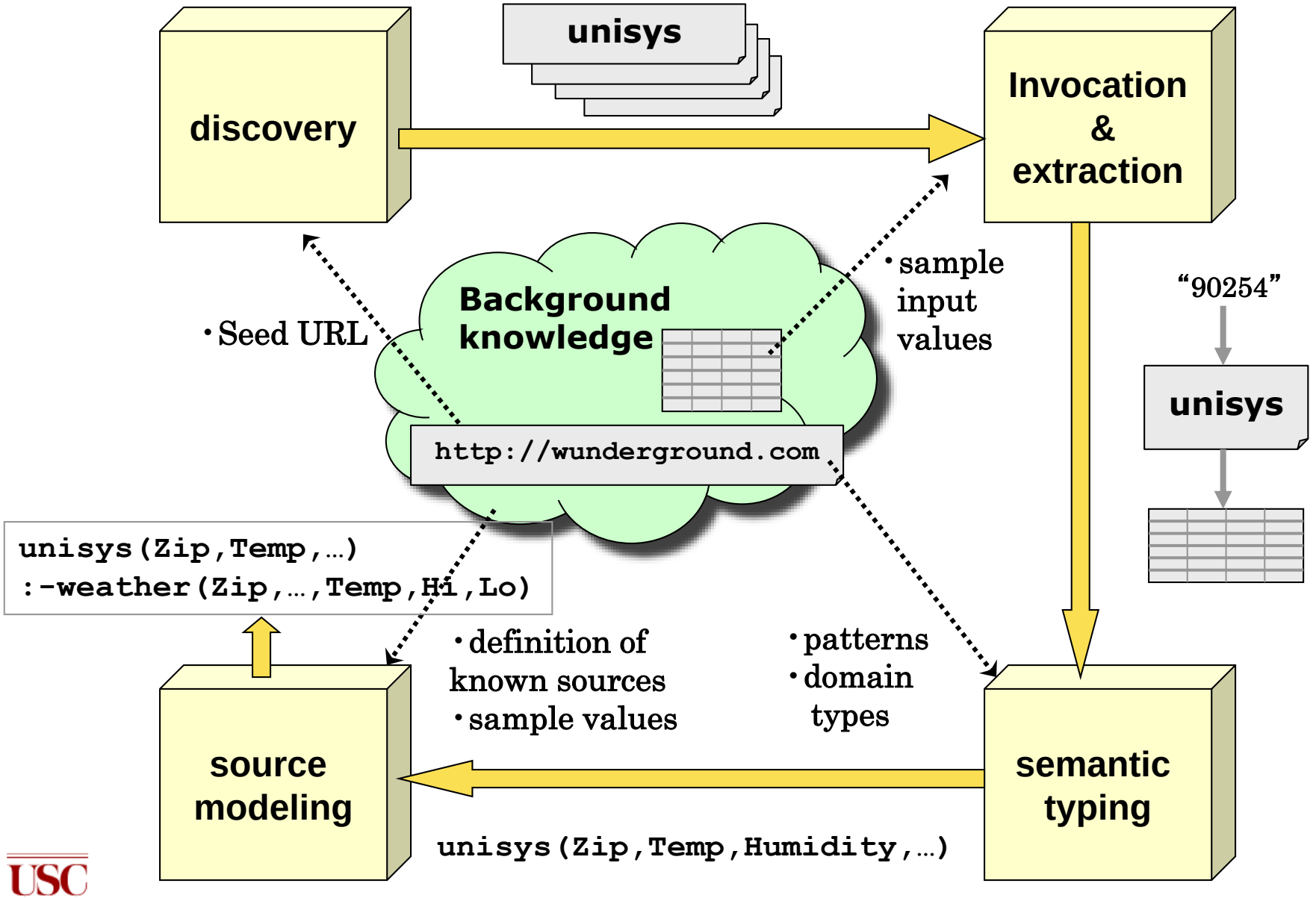
Automatically Discover and Build Semantic Web Services for Related Sources



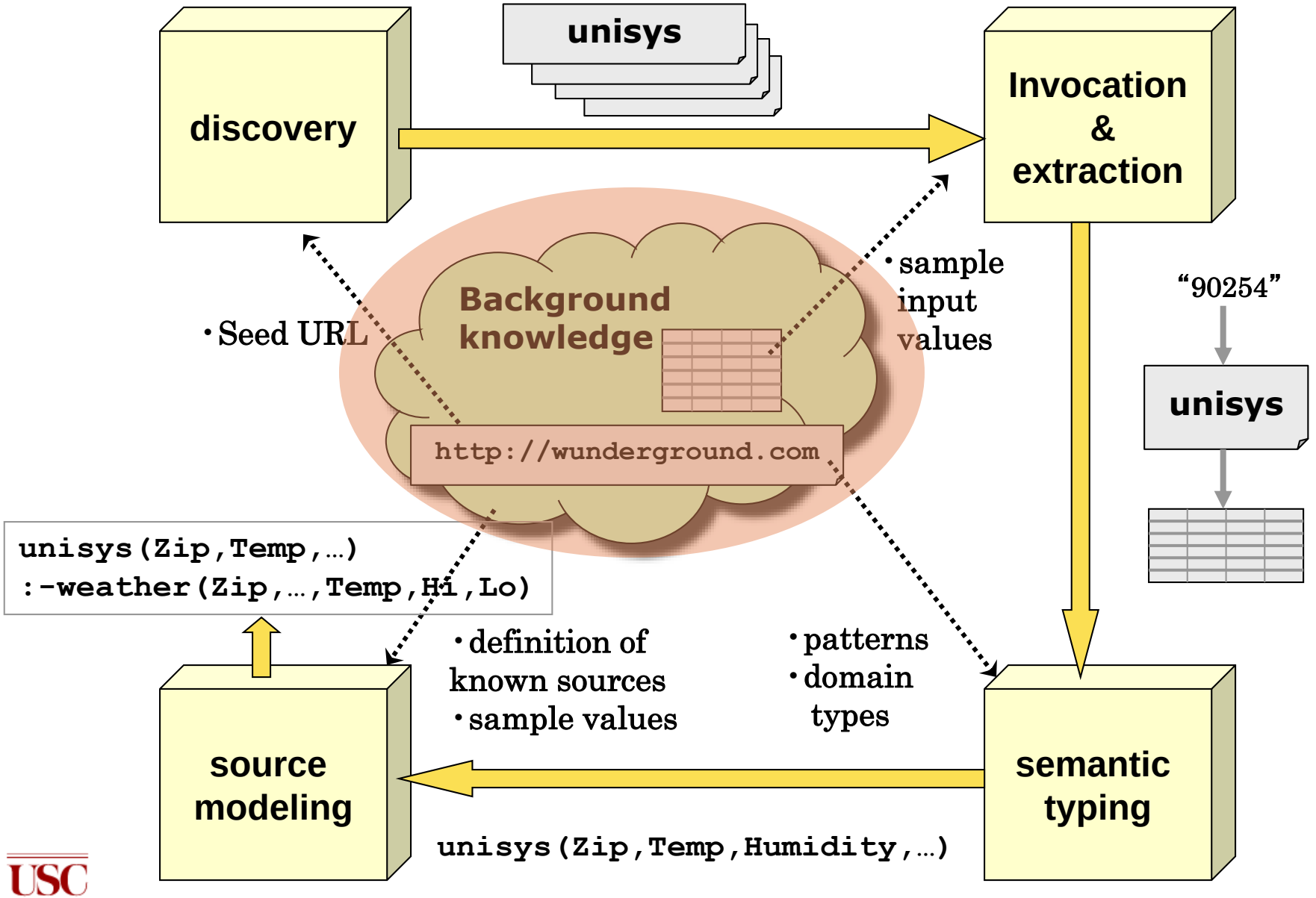


USC

Integrated Approach



Background Knowledge

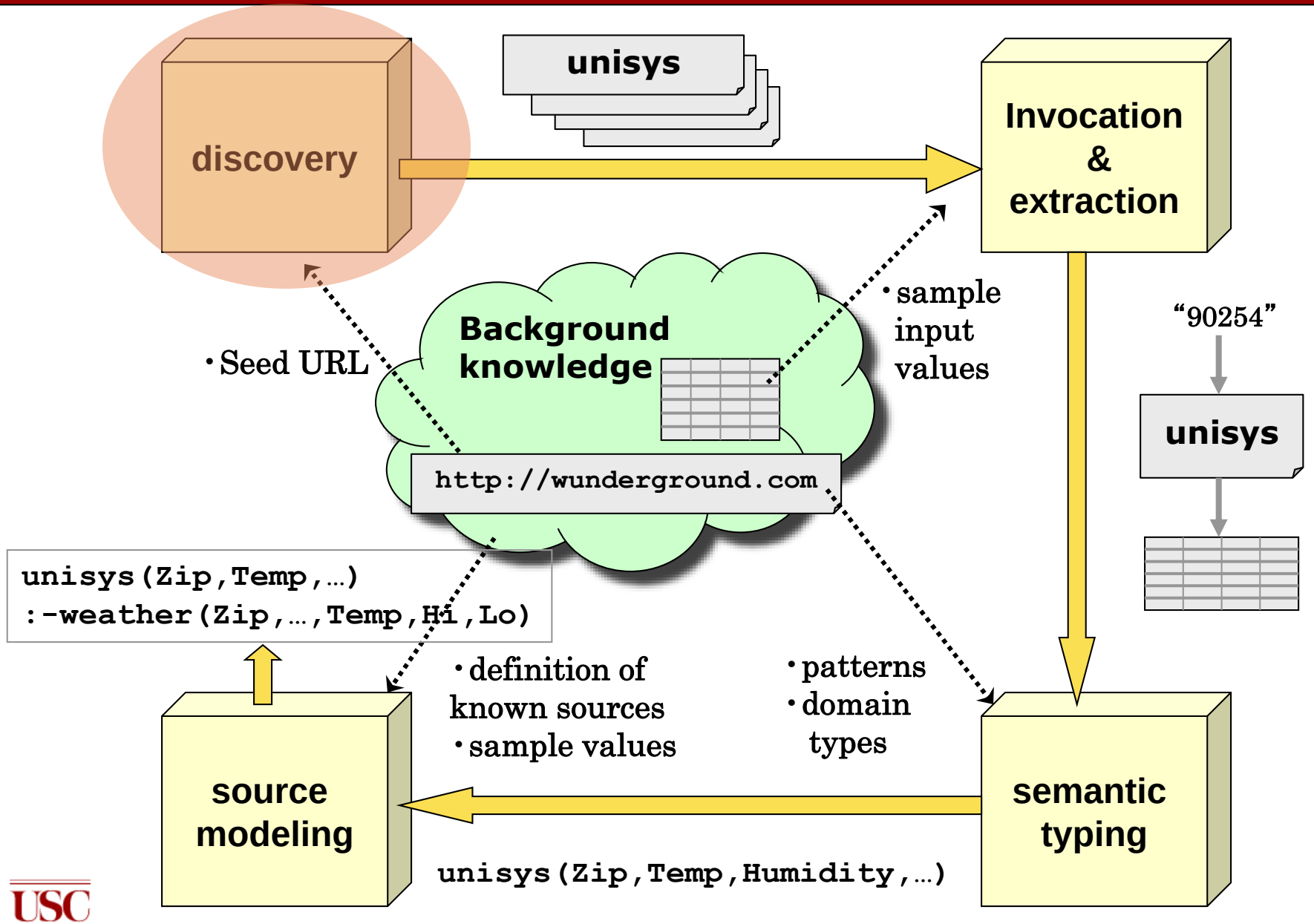


Background Knowledege

- Ontology of the inputs and outputs
  - e.g., TempF, Humidity, Zipcode;
- Sample values for each semantic type
  - e.g., “88 F” for TempF, and “90292” for Zipcode
- Domain input model
  - a weather source may accept Zipcode or City and State as input
  - Sample input values
- Known sources (seeds)
  - e.g., http://wunderground.com
- Source descriptions in Datalog or RDF
  - wunderground($Z,CS,T,F0,S0,Hu0,WS0,WD0,P0,V0,FL1,FH1,S1,FL2,FH2,S2,
    FL3,FH3,S3,FL4,FH4,S4,FL5,FH5,S5) :-
    weather(0,Z,CS,D,T,F0,_,_,S0,Hu0,P0,WS0,WD0,V0)
    weather(1,Z,CS,D,T,_,FH1,FL1,S1,_,_,_,_,_),
    weather(2,Z,CS,D,T,_,FH2,FL2,S2,_,_,_,_,_),
    weather(3,Z,CS,D,T,_,FH3,FL3,S3,_,_,_,_,_),
    weather(4,Z,CS,D,T,_,FH4,FL4,S4,_,_,_,_,_),
    weather(5,Z,CS,D,T,_,FH5,FL5,S5,_,_,_,_,_).

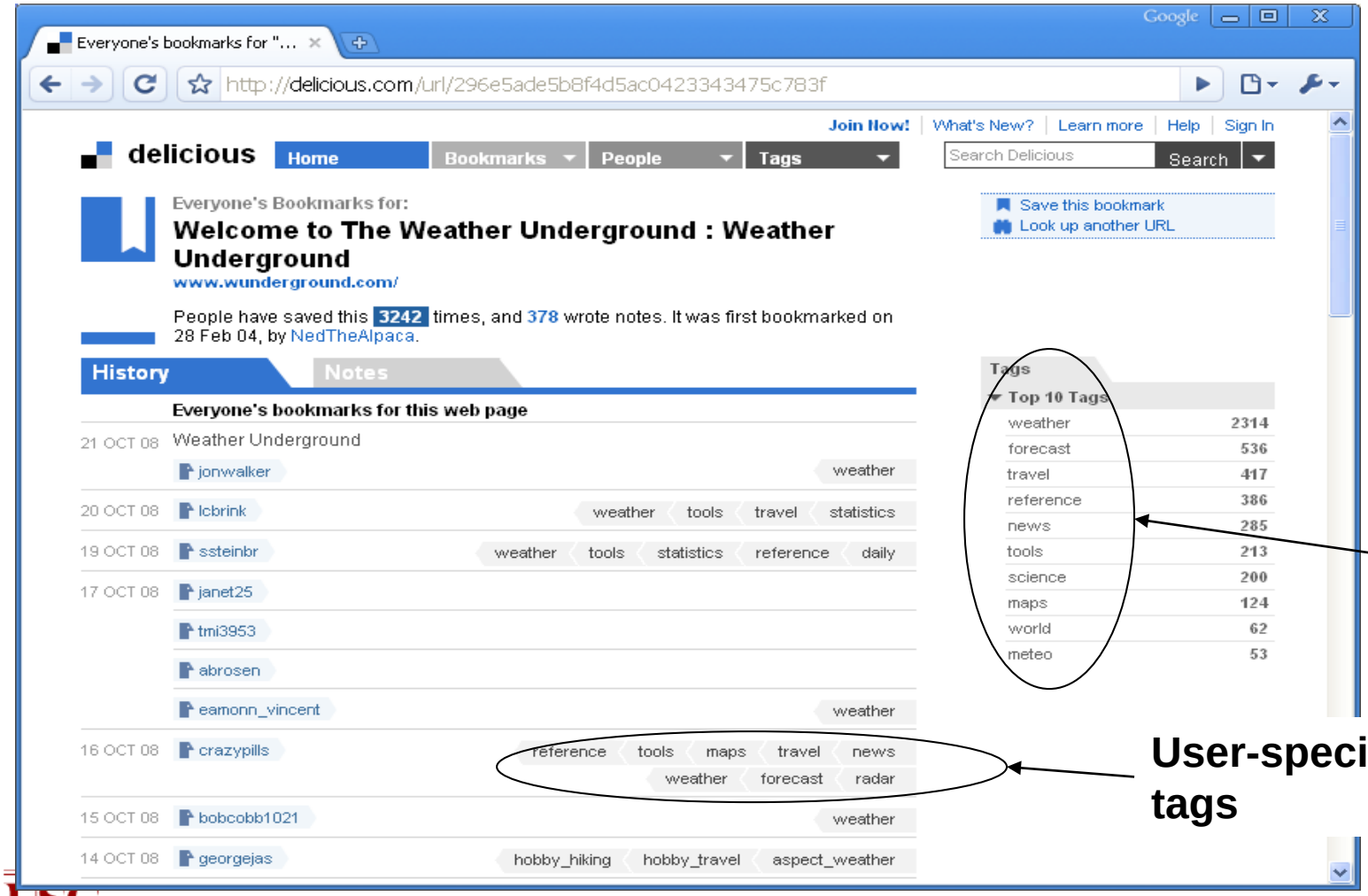
USC

Source Discovery



Source Discovery [Plangprasopchok and Lerman]

- Leverage user-generated tags on the social bookmarking site del.icio.us to discover sources similar to the seed

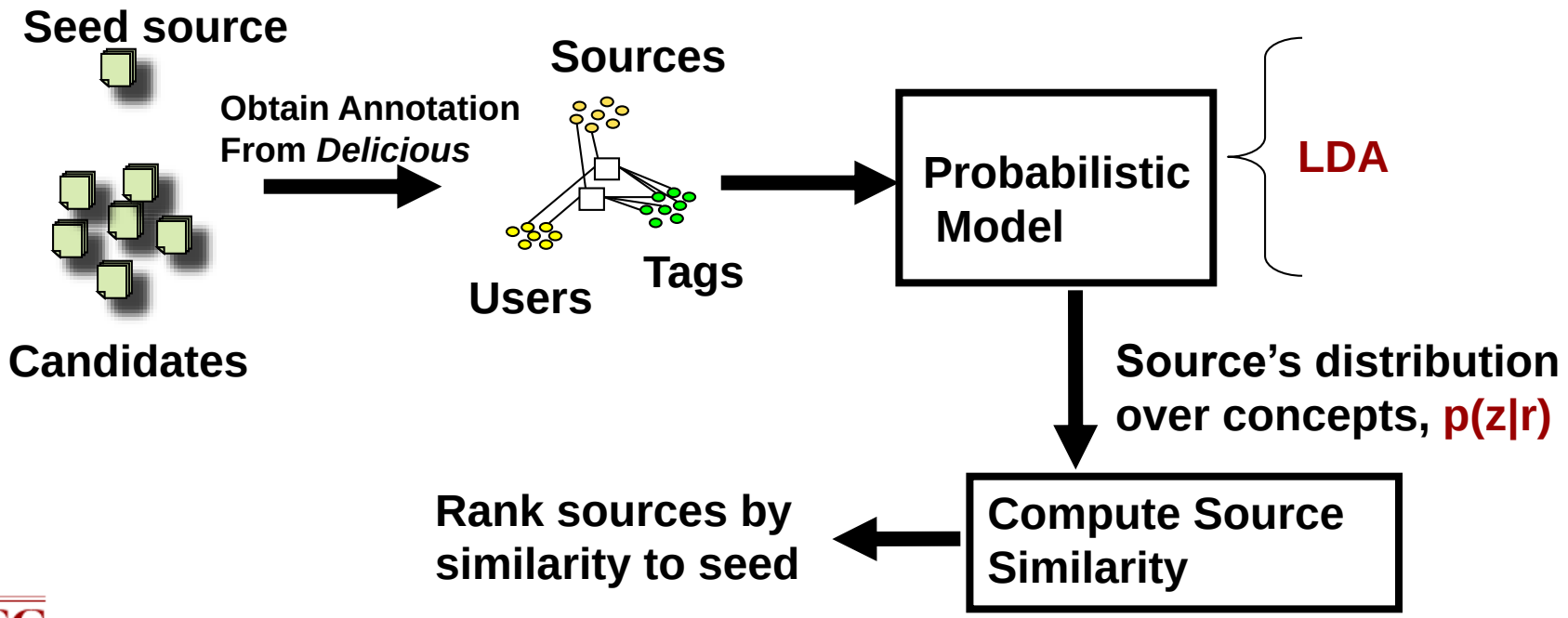


Most common tags

User-specified tags

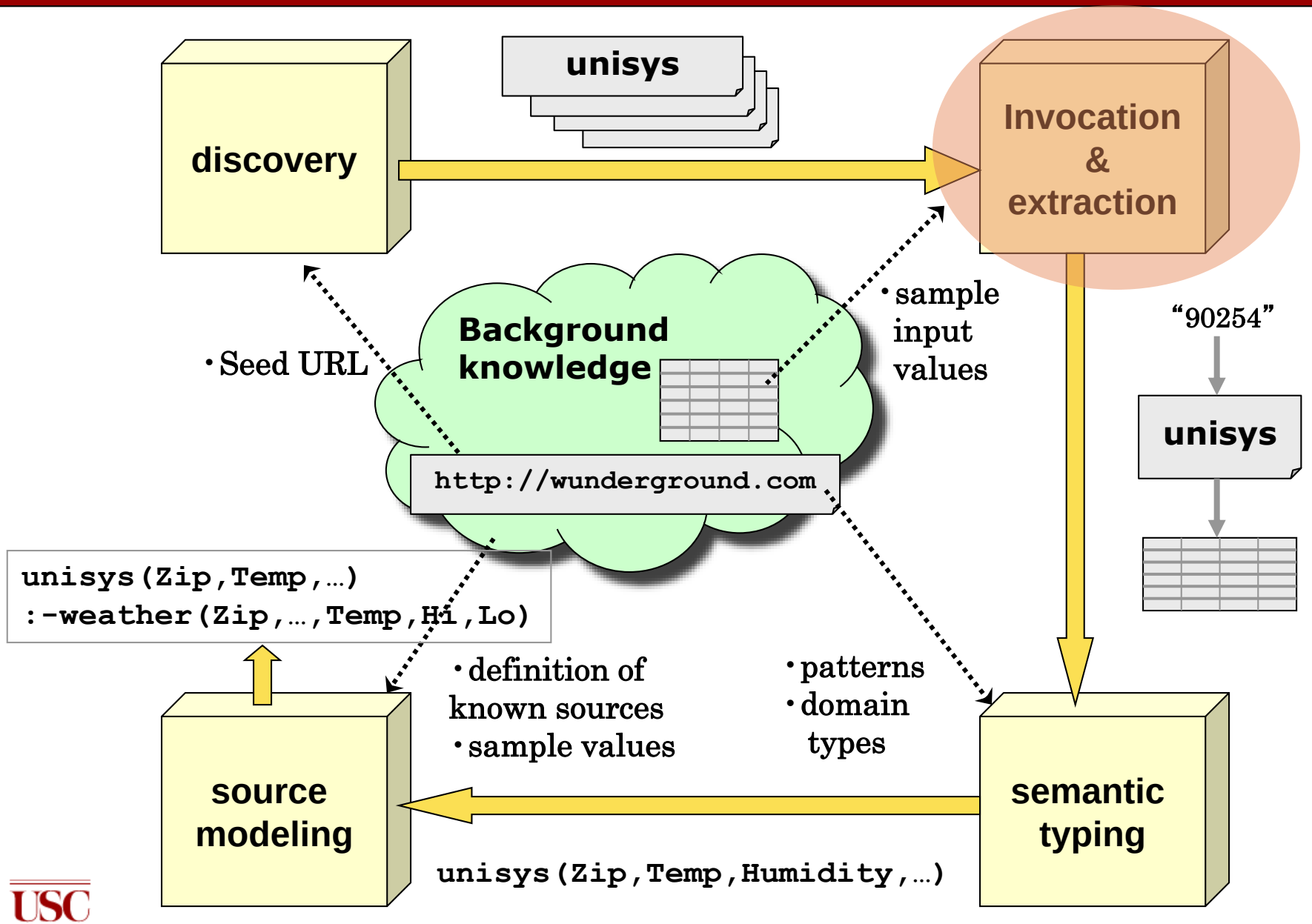
Exploiting Social Annotations for Resource Discovery

- Resource discovery task : "*given a seed source, find other most similar sources*"
  - Gather a corpus of <user, source, tag> bookmarks from del.icio.us
  - Use probabilistic modeling to find hidden topics in the corpus
  - Rank sources by similarity to the seed within topic space



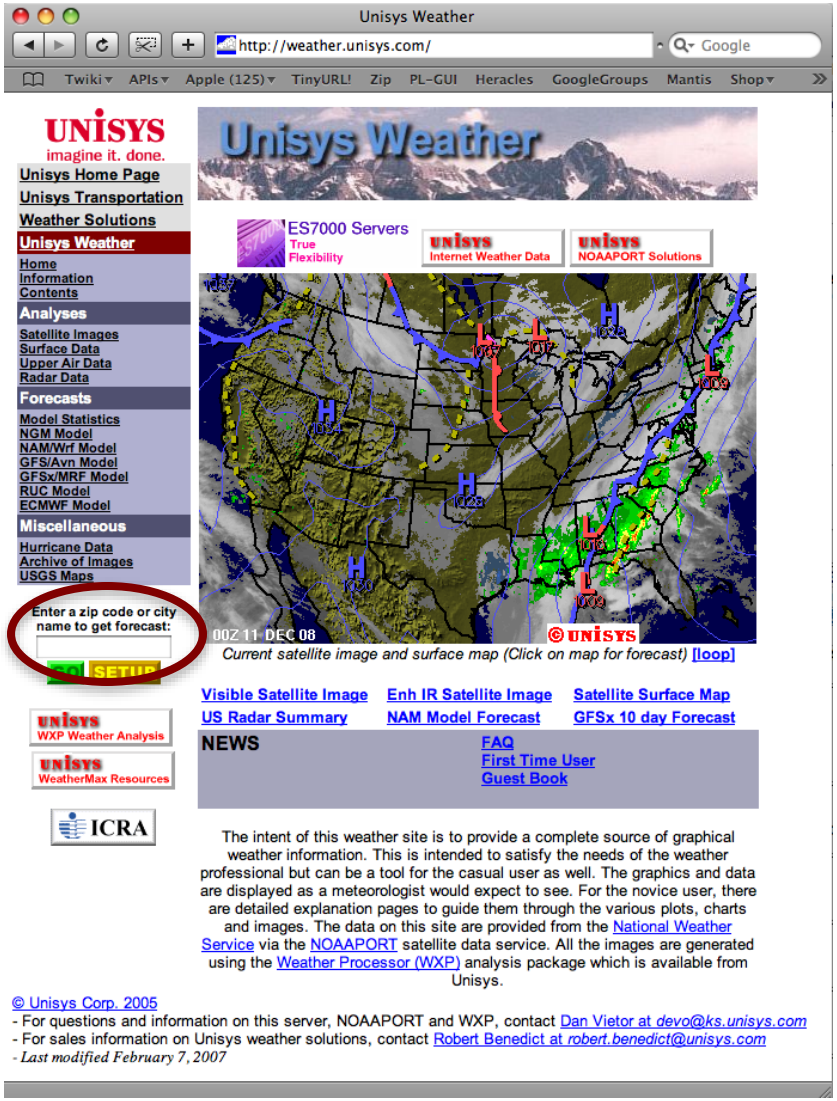
USC

Source Invocation & Extraction



Target Source Invocation

- To invoke the target source, we need to locate the form and determine the appropriate input values
  1. Locate the form
  2. Try different data type combinations as input
     - *For weather, only one input - location, which can be zipcode or city/state*
  3. Submit Form
  4. Keep successful invocations



USC

Inducing Extraction Templates

- Template: a sequence of alternating slots and stripes
  - stripes are the common substrings among all pages
  - slots are the placeholders for data
- Induction: Stripes are discovered using the Longest Common Subsequence algorithm

Sample Page 1

```
<img src="images/Sun.png" alt="Sunny"><br>
<font face="Arial, Helvetica, sans-serif">
  <small><b>Temp: 72F (22C)</b></small></font>
<font face="Arial, Helvetica, sans-serif">
  <small>Site: <b>KSMO (Santa_Monica_Mu, CA)</b><br>
       Time: <b>11 AM PST 10 DEC 08</b>
```

Sample Page 2

```
<img src="images/Clouds.png" alt="Cloudy"><br>
<font face="Arial, Helvetica, sans-serif">
  <small><b>Temp: 37F (2C)</b></small></font>
<font face="Arial, Helvetica, sans-serif">
  <small>Site: <b>KAGC (Pittsburgh/Alle, PA)</b><br>
       Time: <b>2 PM EST 10 DEC 08</b>
```

Induction

Template

```
<img src="images/✱.png" alt="✱"><br>
<font face="Arial, Helvetica, sans-serif">
  <small><b>Temp: ✱ (✱)</b></small></font>
<font face="Arial, Helvetica, sans-serif">
  <small>Site: <b>✱ (✱, ✱)</b><br>
       Time: <b>✱ 10 DEC 08</b>
```

Slot

Stripe

USC

Data Extraction with Templates [Gazen& Minton]

- To extract data: Find data in slots by locating the stripes of the template on unseen page:

Unseen Page

```
<img src="images/Sun.png" alt="Sunny"><br>
<font face="Arial, Helvetica, sans-serif">
  <small><b>Temp: 71F (21C)</b></small></font>
<font face="Arial, Helvetica, sans-serif">
  <small>Site: <b>KCQT (Los_Angeles_Dow, CA)</b><br>
       Time: <b>11 AM PST 10 DEC 08</b>
```

Induced Template

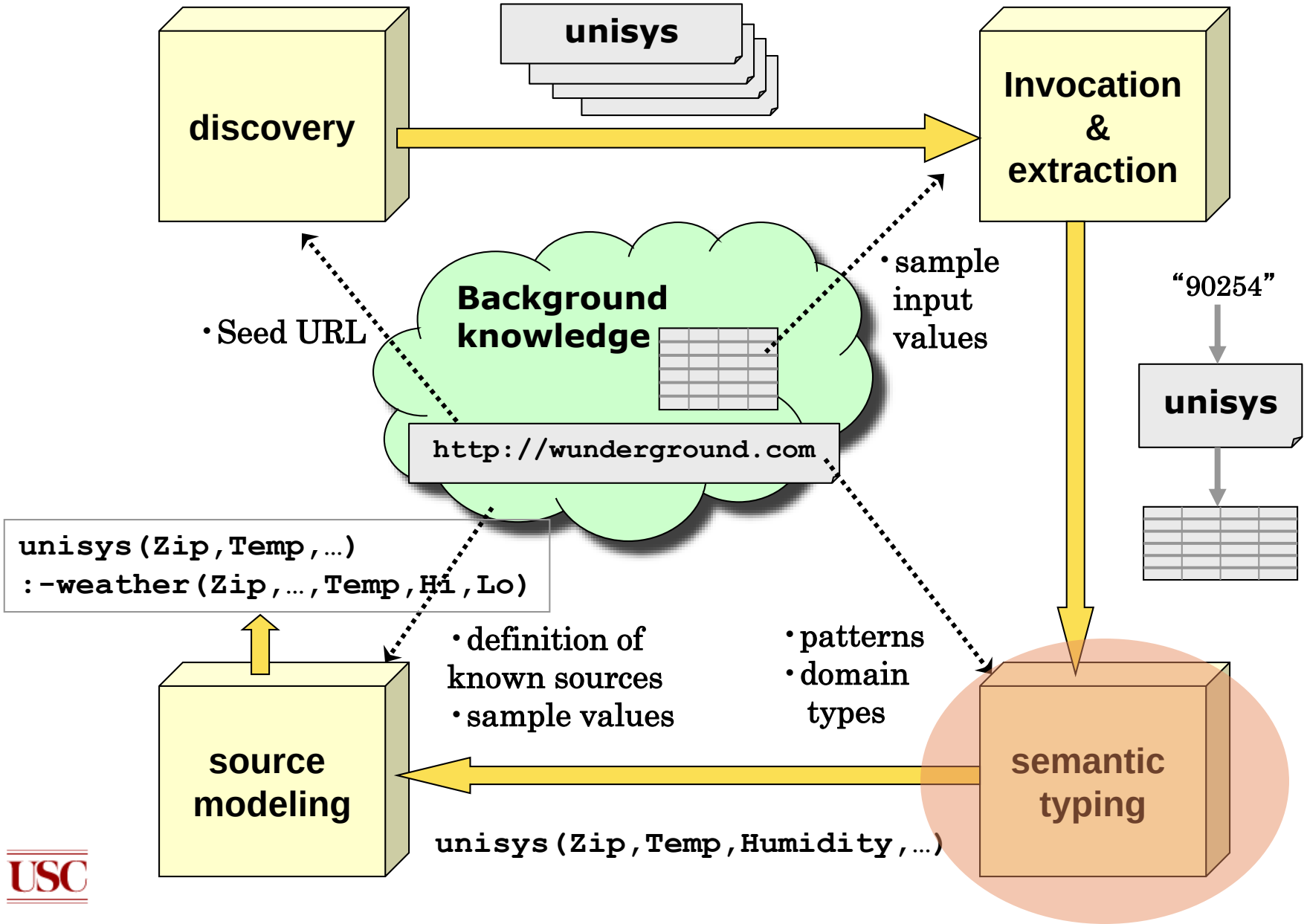
```
<img src="images/*.png" alt="*"><br>
<font face="Arial, Helvetica, sans-serif">
  <small><b>Temp: * (*)</b></small></font>
<font face="Arial, Helvetica, sans-serif">
  <small>Site: <b>* (*, *)</b><br>
       Time: <b>* 10 DEC 08</b>
```

Extracted Data

Sun	Sunny	71F	21C	KCQT	Los_Angeles_Dow	CA	11 AM PST

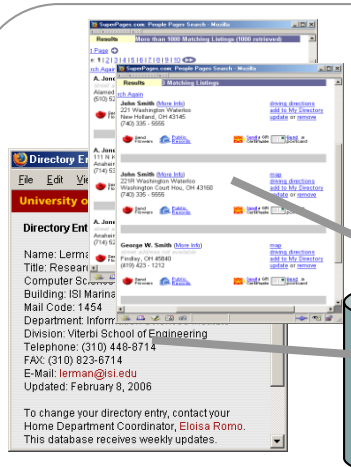
USC

Semantic Typing



Semantic Typing
[Lerman, Plangprasopchok, & Knoblock]

✓**Idea: Learn a model of the content of data and use it to recognize new examples**



Background knowledge

learn

```
:StreetAddress:   :Email:
 4DIG CAPS Rd      ALPHA@ALPHA.edu
 3DIG N CAPS Ave   ALPHA@ALPHA.com
  …                  …
 :State:           :Telephone:
  CA                (3DIG) 3DIG-4DIG
  2UPPER            +1 3DIG 2DIG 4DIG
                    …
```

Patterns

Person	Address	Work
E Lewis	3518 Hilltop Rd	(419) 531 - 0504
Andrew Lewis	3543 Larchmont Pkwy	(518) 474 - [illegible]
C. S. Lewis	555 Willow Run Dr	(612) 578 - [illegible]
Carmen Jones	355 Morgan Ave N	(612) 522 - [illegible]
John Jones	3574 Brookside Rd	(555) 531 - 9566
Location	**State_prov**	**Postal_code**
Toledo	OH	64325-3000
Toledo	OH	64356
Seattle	WA	8422
Seattle	WA	8435
Omaha	NE	52456-6444

label

:FullName:	:StreetAddress:	:Telephone:
E Lewis	3518 Hilltop Rd	(419) 531 - 0504
drew Lewis	3543 Larchmont Pkwy	(518) 474 - 4799
Lewis	555 Willow Run Dr	(612) 578 - 5555
rmen Jones	355 Morgan Ave N	(612) 522 - 5555
John Jones	3574 Brookside Rd	(555) 531 - 9566
:City:	**:State:**	**:Zipcode:**
Toledo	OH	64325-3000
Toledo	OH	64356
Seattle	WA	8422
Seattle	WA	8435
Omaha	NE	52456-6444

USC

Labeling New Data

- Use learned patterns to link new data to types in the ontology
  - Score how well patterns describe a set of examples
    - *Number of matching patterns*
    - *How many tokens of the example match pattern*
    - *Specificity of the matched patterns*
  - Output top-scoring types

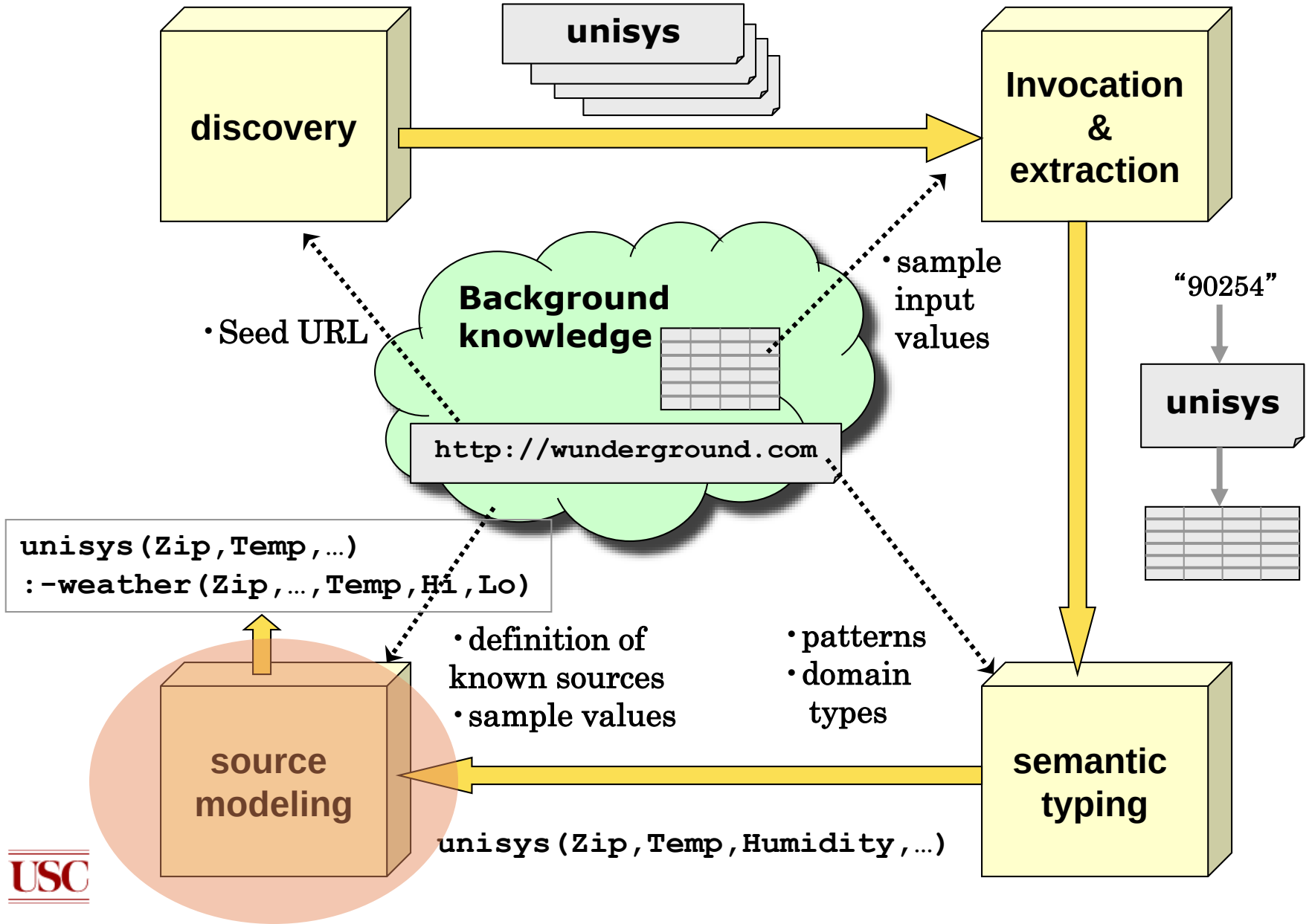
Person	Address	Work
E Lewis	3518 Hilltop Rd	(419) 531 - 0504
Andrew Lewis	3543 Larchmont Pkwy	(518) 474 - 4799
C. S. Lewis	555 Willow Run Dr	(612) 578 - 5555
Carmen Jones	355 Morgan Ave N	(612) 522 - 5555
John Jones	3574 Brookside Rd	(555) 531 - 9566
Location	**State_prov**	**Postal_code**
Toledo	OH	64325-3000
Toledo	OH	64356
Seattle	WA	8422
Seattle	WA	8435
Omaha	NE	52456-6444

patterns

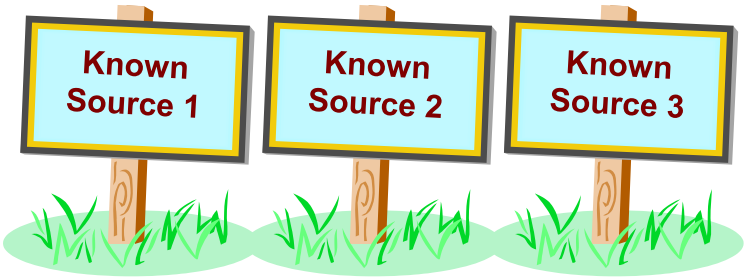
```
:StreetAddress:    :Email:
 4DIG CAPS Rd       ALPHA@ALPHA.edu
 3DIG N CAPS Ave    ALPHA@ALPHA.com
  …                   …
 :State:           :Telephone:
  CA                 (3DIG) 3DIG-4DIG
  2UPPER             +1 3DIG 2DIG 4DIG
  …                  …
```

USC

Source Modeling [Carman & Knoblock]



Inducing Source Definitions

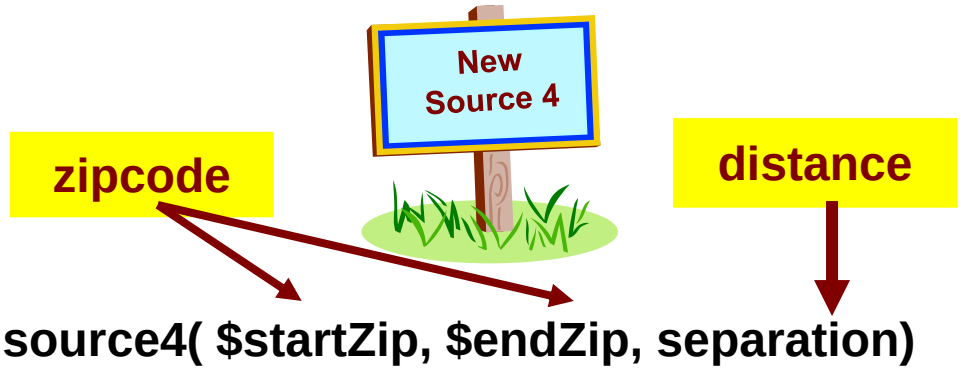


source1($zip, lat, long) :-
centroid(zip, lat, long).

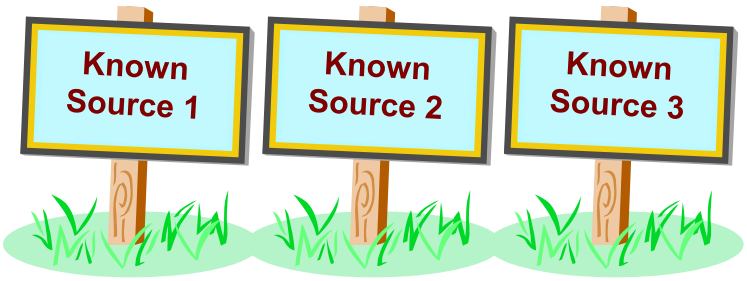
source2($lat1, $long1, $lat2, $long2, dist) :-
greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
convertKm2Mi(dist1, dist2).

- Step 1: classify input & output semantic types



Generating Plausible Definition




```
source1($zip, lat, long) :-
    centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
    greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
    convertKm2Mi(dist1, dist2).
```

- Step 1: classify input & output semantic types
- Step 2: generate plausible definitions

```
source4($zip1, $zip2, dist):-
    source1(zip1, lat1, long1),
    source1(zip2, lat2, long2),
    source2(lat1, long1, lat2, long2, dist2),
    source3(dist2, dist).
```

```
source4($zip1, $zip2, dist):-
    centroid(zip1, lat1, long1),
    centroid(zip2, lat2, long2),
    greatCircleDist(lat1, long1, lat2, long2, dist2),
    convertKm2Mi(dist1, dist2).
```

Invoke and Compare the Definition

- Step 1: classify input & output semantic types
- Step 2: generate plausible definitions
- Step 3: invoke service & compare output

```
source4($zip1, $zip2, dist):-
    source1(zip1, lat1, long1),
    source1(zip2, lat2, long2),
    source2(lat1, long1, lat2, long2, dist2),
    source3(dist2, dist).
```

```
source4($zip1, $zip2, dist):-
    centroid(zip1, lat1, long1),
    centroid(zip2, lat2, long2),
    greatCircleDist(lat1, long1, lat2, long2,dist2),
    convertKm2Mi(dist1, dist2).
```

match

$zip1	$zip2	dist *(actual)*	dist *(predicted)*
80210	90266	842.37	843.65
60601	15201	410.31	410.83
10005	35555	899.50	899.21

Example of a Learned Source Model for Weather Domain

- Given a set of known sources and their descriptions
  - wunderground($Z,CS,T,F0,S0,Hu0,WS0,WD0,P0,V0) :-
    weather(0,Z,CS,D,T,F0,_,_,S0,Hu0,P0,WS0,WD0,V0)
  - convertC2F(C,F) :- centigrade2farenheit(C,F)
- Learn a description of a new source in terms of the known sources
  - unisys($Z,CS,T,F0,C0,S0,Hu0,WS0,WD0,P0,V0) :-
    wunderground(Z,CS,T,F0,S0,Hu0,WS0,WD0,P0,V0),
    convertC2F(C0,F0)

Background Source Descriptions

```
wunderground( $Z,CS,T,F0,C0,S0,Hu0,WS0,WD0,P0,V0,FL1,FH1,S1,
                    FL2,FH2, S2,FL3,FH3,S3,FL4,FH4,S4,FL5,FH5,S5):-
   Weather(_w0),hasForecastDay(_w0,0),hasZIP(_w0,Z),
      hasCityState(_w0,CS),hasTimeWZone(_w0,T),
      hasCurrentTemperatureFarenheit(_w0,F0),
      hasCurrentTemperatureCentigrade(_w0,C0),
      hasSkyConditions(_w0,S0),hasHumidity(_w0,Hu0),
      hasPressure(_w0,P0), hasWindSpeed(_w0,_ws1),
      WindSpeed(_ws1), hasWindSpeedInMPH(_ws1,WS0),
      hasWindDir(_ws1,WD0), hasVisibilityInMi(_w0,V0),
   Weather(_w1), hasForecastDay(_w1,1), hasZIP(_w1,Z),
      hasCityState(_w1,CS), hasLowTemperatureFarenheit(_w1,FL1),
      hasHighTemperatureFarenheit(_w1,FH1), hasSkyConditions(_w1,S1),
   …

convertC2F($C,F) :- centigrade2farenheit(C,F)
```

Target explained using background sources

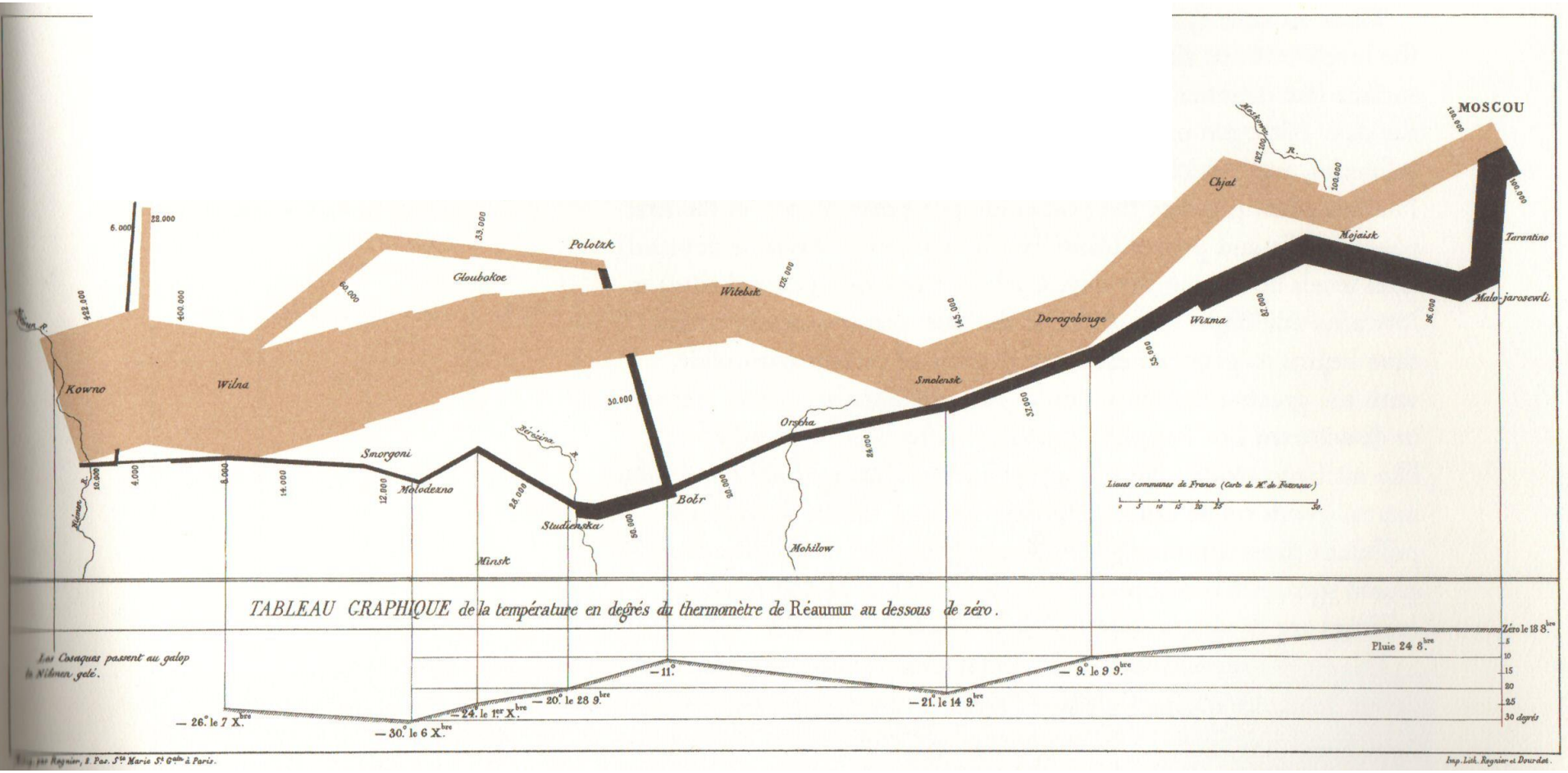
```
unisys($Z,_,_,_,_,_,_,_,_,F9,_,C,_,F13,F14,Hu,_,F17,_,_,_,_,S22,_,S24,
        _,_,_,_,_,_,_,_,_,_,_,S35,S36,_,_,_,_,_,_,_,_,_) :-
    wunderground(Z,_,_,F9,_,Hu,_,_,_,_,F14,F17,S24,_,_,S22,_,_,
                 S35,_,_,S36,F13,_,_),
    convertC2F(C,F9)
```

USC

Experimental Evaluation

- Experiments in 5 domains
  - Flight – lookup the current status of a flight
  - Geospatial – map streeet addresses into lat/long coordinates
  - Weather – find the current and forecasted weather
  - Currency – convert between various currencies
  - Mutual Funds – look up current data on a mutual fund

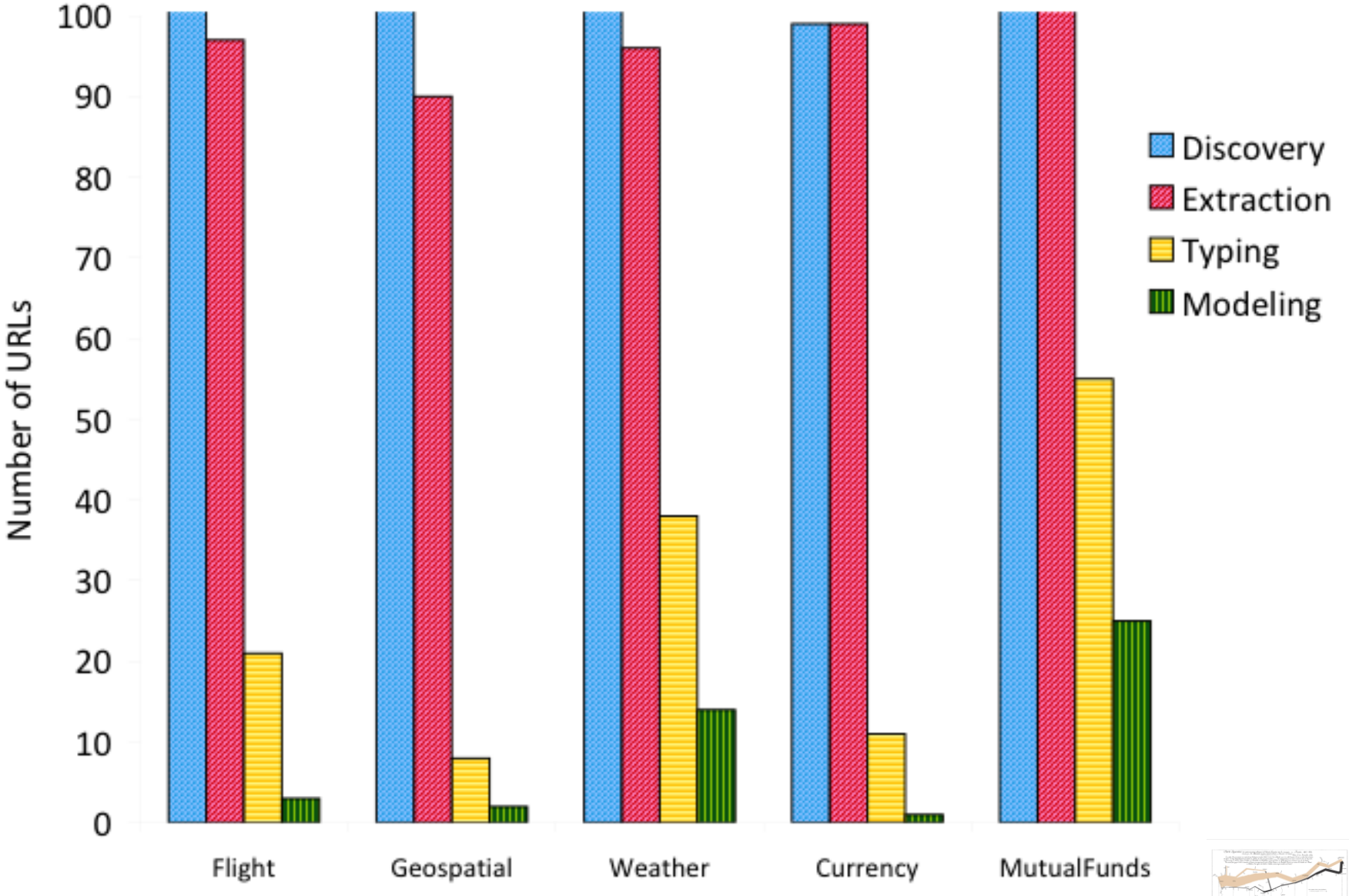
- Evaluation:
  - 1) Can the system correctly learn a model for those sources that perform the same task
  - 2) What is the precision and recall of the attributes in the model



[Edward Tufte, *The Visual Display of Quantitative Information*]

USC

Candidate Sources after Each Step



Evaluation of the Models

domain	Precision	Recall	F_1-measure
weather	0.64	0.29	0.39
geospatial	1.00	0.86	0.92
flights	0.69	0.35	0.46
currency	1.00	1.00	1.00
mutualfund	0.72	0.30	0.42

Conclusions

- Integrated approach to learning:
  - *How to invoke a web service*
  - *The semantic types of the output*
  - *A definition of what the service does*
- Provides an approach to generate source descriptions for the Semantic Web
  - Little motivation for providers to annotate services
  - Instead we generate metadata automatically
- Also provides an approach to automatically discover new sources of data