EDAN20
Language Technology
http://cs.lth.se/edan20/
Chapter 14: Semantics and Predicate Logic

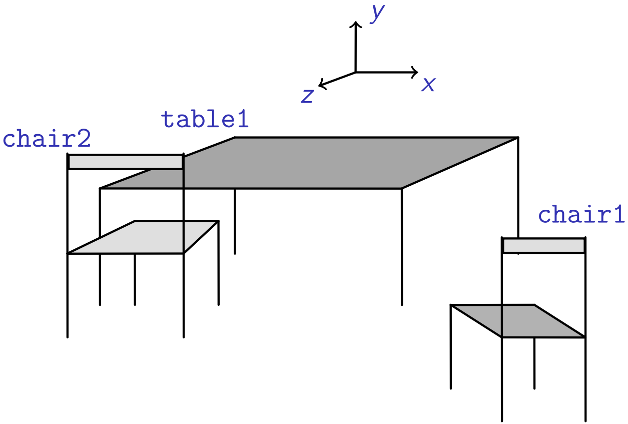
Pierre Nugues

Lund University
Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

October 3, 2019

The State of Affairs

Two people at a table, Pierre and Socrates, and a robot waiter.



Formal Semantics

Its goal is to:

- Represent the state of affairs.
- Translate phrases or sentences such as *The robot brought the meal* or *the meal on the table* into logic formulas
- Solve references: Link words to real entities
- Reason about the world and the sentences.

A way to represent things and relations is to use first-order predicate calculus (FOPC) and predicate–argument structures

Predicates

Constants:

```
% The people:
  'Socrates'.
  'Pierre'.

% The chairs:
  chair1.     % chair #1
  chair2.     % chair #2

% The unique table:
  table1.     % table #1
```

Predicates to encode properties:

```
person('Pierre').
person('Socrates').

object(table1).
object(chair1).
object(chair2).

chair(chair1).
chair(chair2).
table(table1).
```

Predicates to encode relations:

```
in_front_of(chair1, table1).
on('Pierre', table1).
```

Prolog

Prolog is a natural tool to do first-order predicate calculus

- Things, either real or abstract, are mapped onto constants – or atoms: `'Socrates'`, `'Pierre'`, `chair1`, `chair2`.
- Predicates can encode properties: `person('Pierre')`, `person('Socrates')`, `object(table1)`, `object(chair1)`.
- Predicates can encode relations: `in_front_of(chair1, table1)`, `on('Pierre', table1)`.
- Variables unify with objects

Querying the State of Affairs

Constants:

```
?- table(chair1).
false.
?- chair(chair2).
true.
```

Variables:

```
?- chair(X).
X = chair1;
X = chair2
```

Conjunctions:

```
?- chair(X), in_front_of(X, Y), table(Y).
X = chair1, Y = table1
```

Logical Forms

Logical forms map sentences onto predicate-argument structures

I would like to book a late flight to Boston

```
would(like_to(i,
   book(i,
       np_pp(a(late(flight)),
           X^to(X, boston)))))
```

Compositionality

The principle of compositionality assumes that a sentence's meaning depends on the meaning its phrases

"The meaning of the whole is a function of the meaning of its parts."

A complementary assumption is that phrases carrying meaning can be mapped onto constituents – syntactic units.

The principle of compositionality ties syntax and semantics together.

We saw that a predicate-argument structure could represent a sentence – the whole. How to represent the parts – the constituents?

λ-Calculus

The λ-calculus is a device to abstract properties or relations.

$$\lambda x.property(x)$$

or

$$\lambda y.\lambda x.relation(x,y)$$

A λ-expression is incomplete until a value has been given to it.
Supplying such a value is called a β-reduction.

$$(\lambda x.property(x))entity\#1$$

yields

$$property(entity\#1)$$

In Prolog, `X^property(X)` represents $\lambda x.property(x)$

Nouns

Proper nouns: *Mark*, *Nathalie*, *Ludwig*

Common nouns (properties): *lecturer*, *book*:

$$\lambda x.lecturer(x) \quad \lambda x.lecturer(x)(Bill) = lecturer(Bill)$$

Adjectives

$$\lambda x.big(x) \quad \lambda x.big(x)(Bill) = big(Bill)$$

Adjectives and nouns: *big table*

$$\lambda x.(big(x) \wedge table(x))$$

Noun compounds are difficult: *lecture room*

$$\lambda x.(lecture(x) \wedge room(x)) \quad ?? \text{ Wrong!}$$

A better form is:

$$\lambda x.(modify(x, lecture) \wedge room(x))$$

although not completely satisfying

Verbs

Verbs of being are similar to adjectives or nouns

Intransitive verbs	$\lambda x.rushed(x)$
	$\lambda x.rushed(x)(Bill) = rushed(Bill)$
Transitive verbs	$\lambda y.\lambda x.ordered(x, y)$
Prepositions	$\lambda y.\lambda x.to(x, y)$

Determiners

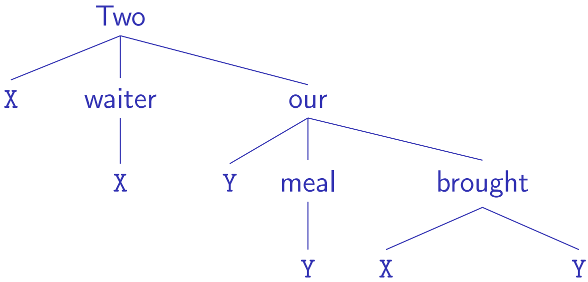
A caterpillar is eating $\exists x, caterpillar(x) \wedge eating(x)$, or `exists(X, caterpillar(X), eating(X))`
Every caterpillar is eating $\forall x, caterpillar(x) \Rightarrow eating(x)$, or `all(X, caterpillar(X), eating(X))`
A caterpillar is eating a hedgehog $\exists x, caterpillar(x) \wedge (\exists y, hedgehog(y) \wedge eating(x, y))$, or `exists(X,caterpillar(X),exists(Y, hedgehog(Y), eating(X, Y))`
Every caterpillar is eating a hedgehog $\forall x, caterpillar(x) \Rightarrow (\exists y, hedgehog(y) \wedge eating(x, y))$, or `all(X, caterpillar(X), exists(Y, hedgehog(Y), eating(X, Y))`

Determiners: An Example

Two waiters brought our meals

translated into

```
two(X, waiter(X), our(Y, meal(Y), brought(X, Y)))
```



General Representation of Determiners

`determiner`

`X`

Semantics of the noun phrase: `SemNP(X...)`

Semantics of the rest of the sentence: `SemRest`

Representation:

```
(X^NP)^(X^Rest)^a(X, NP, Rest)
```

λ-Representations

The partial, intermediate representations of
The waiter brought the meal are

waiter	`X^waiter(X)`
The waiter	`(X^Rest)^the(X, waiter(X), Rest)`
brought	`Y^X^brought(X, Y)`
meal	`Y^meal(Y)`
the meal	`(Y^Verb)^the(Y, meal(Y), Verb)`

The operation to compose *brought the meal* is more complex It should produce something like:

```
X^the(Y, meal(Y), brought(X, Y))
```

λ-Representations (II)

We parse the verb phrase *brought the meal* using the rule

```
vp(SemVP) --> verb(SemVerb), np(SemNP).
```

We have:

```
SemVerb = Y^X^brought(X, Y)
SemNP = (Y^Verb)^the(Y, meal(Y), Verb)
```

We just write the unification: `Verb=brought(X, Y)`
Prolog returns:

```
?- SemVerb = Y^X^brought(X, Y),
    SemNP = (Y^Verb)^the(Y, meal(Y), Verb),
    Verb=brought(X, Y).
SemVerb = Y^X^brought(X, Y),
SemNP = (Y^brought(X, Y))^the(Y, meal(Y), brought(X, Y)),
Verb = brought(X, Y).
```

Compositionality: The Lexicon

```
noun(X^waiter(X)) --> [waiter].
noun(X^patron(X)) --> [patron].
noun(X^meal(X)) --> [meal].
verb(X^rushed(X)) --> [rushed].
verb(Y^X^ordered(X, Y)) --> [ordered].
verb(Y^X^brought(X, Y)) --> [brought].
determiner((X^NP)^(X^Rest)^a(X, NP, Rest)) --> [a].
determiner((X^NP)^(X^Rest)^the(X, NP, Rest)) --> [the].
```

Interleaving Syntax and Semantics

```
s(Semantics) --> np((X^Rest)^Semantics), vp(X^Rest).
np((X^Rest)^SemDet) -->
  determiner((X^NP)^(X^Rest)^SemDet),
  noun(X^NP).
vp(Subject^Verb) --> verb(Subject^Verb).
vp(Subject^Predicate) -->
  verb(Object^Subject^Verb),
  np((Object^Verb)^Predicate).

?- s(Semantics, [the, patron, ordered, a, meal], []).
Semantics = the(_4,patron(_4),a(_32,meal(_32),ordered(_4,_32)))
```

Resolving References: exists

A hedgehog has a nest

```
a(X, hedgehog(X), a(Y, nest(Y), have(X, Y)).

?- hedgehog(X), a(Y, nest(Y), have(X, Y)).

exists(X, Property1, Property2) :-
  Property1,
  Property2,
  !.
```

Resolving References: all

All hedgehogs have a nest

```
all(X, hedgehog(X), a(Y, nest(Y), have(X, Y)).
```

There is no hedgehog, which has no nest

```
all(X, Property1, Property2) :-
  \+
  (Property1,
  \+ Property2),
  Property1,
  !.
```

Application: Spoken Language Translator (Agnäs et al. 1994)

English	*What is the earliest flight from Boston to Atlanta?*
French	*Quel est le premier vol Boston-Atlanta?*
English	*Show me the round trip tickets from Baltimore to Atlanta*
French	*Indiquez-moi les billets aller-retour Baltimore-Atlanta*
English	*I would like to go about nine am*
French	*Je voudrais aller aux environs de 9 heures*
English	*Show me the fares for Eastern Airlines flight one forty seven*
French	*Indiquez-moi les tarifs pour le vol Eastern Airlines cent quarante sept*

Semantic Interpretation

Question:

What is the earliest flight from Boston to Atlanta?

Modeling a flight from Boston to Atlanta:

$$\exists x(\mathit{flight}(x) \wedge \mathit{from}(x, \mathit{Boston}) \wedge \mathit{to}(x, \mathit{Atlanta}) \wedge \exists y(\mathit{time}(y) \wedge \mathit{departs}(x, y)))$$

Finding the earliest flight:

$$\arg\min_{y} \exists x(\mathit{flight}(x) \wedge \mathit{from}(x, \mathit{Boston}) \wedge \mathit{to}(x, \mathit{Atlanta}) \wedge \exists y(\mathit{time}(y) \wedge \mathit{departs}(x, y)))$$

SLT uses the logical form as a universal representation, independent from the language.

It converts sentences from and to this representation

Semantic Parsing

SLT does not use variables for the nouns.

I would like to book a late flight to Boston is

converted into the Prolog term:

```
would(like_to(i,
    book(i,
        np_pp(a(late(flight)),
            X^to(X, boston)))))
```

Grammar Rules

```
1  rule(s_np_vp,
     s([sem=VP]),
     [np([sem=NP,agr=Ag]),
     vp([sem=VP,subjsem=NP,aspect=fin,agr=Ag])]).
2  rule(vp_v_np,
     vp([sem=V,subjsem=Subj,aspect=Asp,agr=Ag]),
     [v([sem=V,subjsem=Subj,aspect=Asp,agr=Ag,
       subcat=[np([sem=NP])]]),
     np([sem=NP,agr=_])]).
3  rule(vp_v_vp,
     vp([sem=V,subjsem=Subj,aspect=Asp,agr=Ag]),
     [v([sem=V,subjsem=Subj,aspect=Asp,agr=Ag,
       subcat=[vp([sem=VP,subjsem=Subj])]]),
     vp([sem=VP,subjsem=Subj,aspect=ini,agr=])]).
```

Lexicon

#	Lexicon entries
1	`lex(boston,np([sem=boston,agr=(3-s)])).`
2	`lex(i,np([sem,agr=(1-s)])).`
3	`lex(flight,n([sem=flight,num=s])).`
4	`lex(late,adj([sem=late(NBAR),nbarsem=NBAR])).`
5	`lex(a,det([sem=a(NBAR),nbarsem=NBAR,num=s])).`
6	`lex(to,prep([sem=X^to(X,NP),npsem=NP])).`
7	`lex(to,inf([])).`
8	`lex(book,v([sem=have(Subj,Obj),subjsem=Subj,aspect=ini, agr=_,subcat=[np([sem=Obj])]])).`
9	`lex(would,v([sem=would(VP),subjsem=Subj,aspect=fin, agr=_,subcat=[vp([sem=VP,aubjsem=Subj])]])).`
10	`lex(like,v([sem=like_to(Subj,VP),subjsem=Subj,aspect=ini, agr=_,subcat=[inf([]),vp([sem=VP,subjsem=Subj])]])).`

Transferring Logical Forms

```
trule(<Comment>
  <QLF pattern 1>  <Operator> <QLF pattern 2>).
```

Operator is >=, =<, or ==.

Lexical rules	Syntactic rules

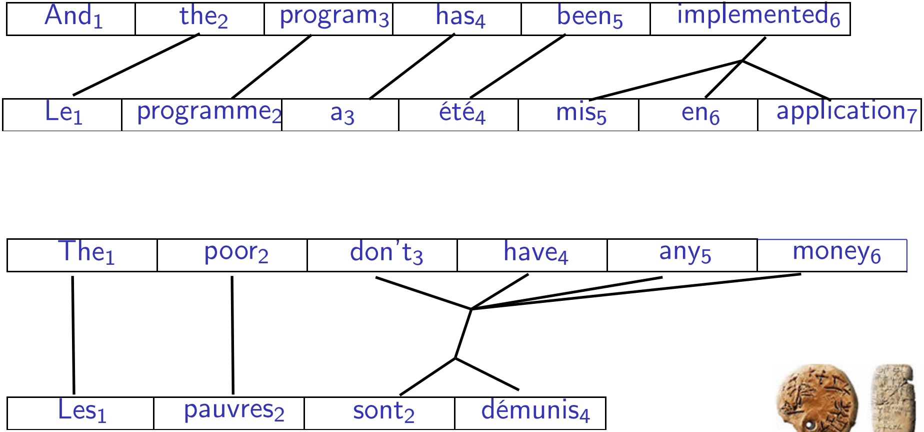
`trule([eng, fre],`
`  flight1 > vol1).`
	`trule([eng, fre],`
	`  form(tr(relation,nn),`
	`    tr(noun1),`
	`    tr(noun2))`
	`  >=`
	`  [and, tr(noun2),`
	`    form(prep(tr(relation)),`
	`      tr(noun1))]).`

Parallel Corpora (Swiss Federal Law)

German	French	Italian
Art. 35 Milchtransport	**Art. 35 Transport du lait**	**Art. 35 Trasporto del latte**
1 Die Milch ist schonend und hygienisch in den Verarbeitungsbetrieb zu transportieren. Das Transportfahrzeug ist stets sauber zu halten. Zusammen mit der Milch dürfen keine Tiere und milchfremde Gegenstände transportiert werden, welche die Qualität der Milch beeinträchtigen können.	1 Le lait doit être transporté jusqu'à l'entreprise de transformation avec ménagement et conformément aux normes d'hygiène. Le véhicule de transport doit être toujours propre. Il ne doit transporter avec le lait aucun animal ou objet susceptible d'en altérer la qualité.	1 Il latte va trasportato verso l'azienda di trasformazione in modo accurato e igienico. Il veicolo adibito al trasporto va mantenuto pulito. Con il latte non possono essere trasportati animali e oggetti estranei, che potrebbero pregiudicarne la qualità.
2 Wird Milch ausserhalb	2 Si le lait destiné à	2 Se viene collocato fuori

Alignment (Brown et al. 1993)

Canadian Hansard



RDF and SPARQL

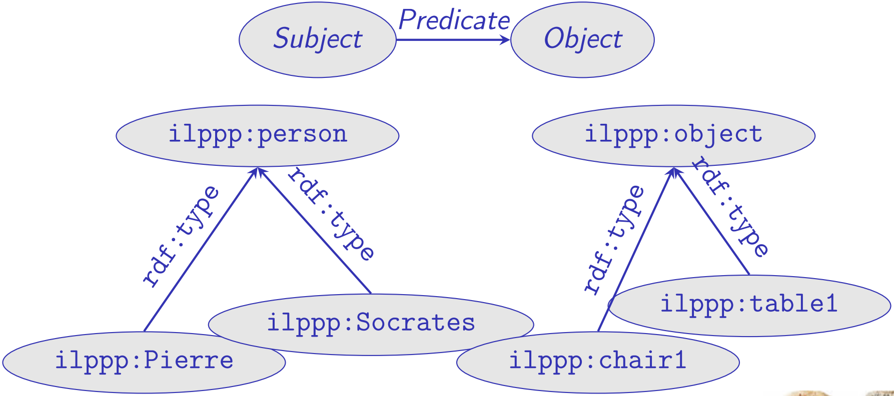
RDF: A popular graph format to encode knowledge.
SPARQL: A query language for RDF
In many ways, very similar to Prolog.

```
ilppp:Pierre rdf:type ilppp:person.
ilppp:Socrates rdf:type ilppp:person.

ilppp:table1 rdf:type ilppp:object.
ilppp:chair1 rdf:type ilppp:object.
ilppp:chair2 rdf:type ilppp:object.

ilppp:chair1 ilppp:in_front_of ilppp:table1.
ilppp:Pierre ilppp:on ilppp:table1.
```

RDF Triples


Subject
Predicate
Object
ilppp:person
ilppp:object
rdf:type
rdf:type
rdf:type
rdf:type
ilppp:table1
ilppp:Socrates
ilppp:Pierre
ilppp:chair1

RDF and SPARQL

Prolog:

```
?- object(X), object(Y), in_front_of(X, Y).
X = chair1,
Y = table1.
```

SPARQL:

```
SELECT ?x ?y
WHERE
{
  ?x rdf:type ilppp:object.
  ?y rdf:type ilppp:object.
  ?x ilppp:in_front_of ?y
}
```

Variables	?x	?y
Values	ilppp:chair1	ilppp:table1

DBpedia, Yago, Wikidata, and Freebase

Graph databases consisting of billions of RDF triples.
Coming from a variety of sources such as Wikipedia infoboxes:

```
{{Infobox settlement
| official_name          = Busan Metropolitan City
...
| area_total_km2         = 767.35
...
| population_total       = 3,525,913
...
}}
```

DBpedia: The result of a systematic triple extraction from infoboxes

```
dbpedia:Busan foaf:name "Busan Metropolitan City"@en .
dbpedia:Busan dbo:populationTotal "3525913".
dbpedia:Busan dbo:areaTotal "7.6735E8" .
...
```

SPARQL Endpoint

A network service accepting SPARQL queries such as:

```
prefix dbo: <http://dbpedia.org/ontology/>
prefix foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?entity ?population
WHERE
{
  ?entity foaf:name "Busan Metropolitan City"@en.
  ?entity dbo:populationTotal ?population.
}
```

Address of the DBpedia endpoint: http://dbpedia.org/sparql

Wikidata

Wikidata provides another endpoint based on Wikipedia data:
`https://query.wikidata.org`

```
query = '''
SELECT ?entity ?population
WHERE
{
  ?entity rdfs:label "Busan"@en .
  ?entity wdt:P1082 ?population.

}'''

url = 'https://query.wikidata.org/bigdata/namespace/wdq/sparql'
data = requests.get(url, params={'query': prefixes + query, 'fo
```

DBpedia

The DBpedia query returns:

Variables	entity	population
Values	`http://dbpedia.org/resource/Busan`	3525913

where `http://dbpedia.org/resource/Busan` or `dbpedia:Busan` is a unique entity name based on the Wikipedia web addresses (URI nomenclature).