Hybrid Logic Dependency Semantics

What goes *into* the OpenCCG surface realiser?

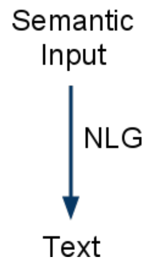
Lecture 2

January 22, 2013

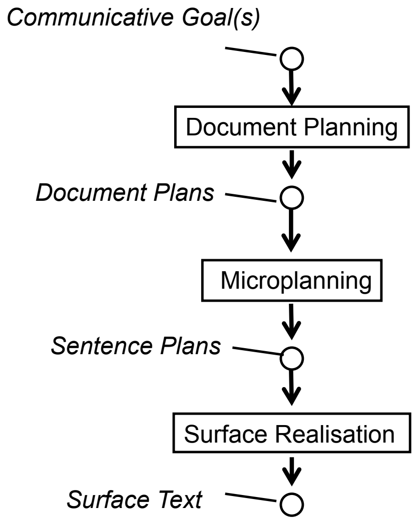
Brief recap - What is NLG?

How computer programs can be made to produce (high-quality) natural language text from

- computer-internal representations of information
- other texts



Brief recap - the NLG pipeline

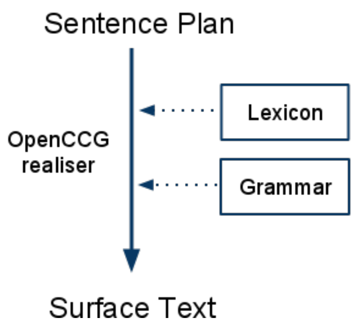


The first part of this course is concerned with the bottom part of the pipeline:

surface realisation

Using the (state-of-the-art) **OpenCCG** surface realiser

Surface realisation with OpenCCG



Today's lecture:

What do OpenCCG sentence plans look like?

i.e. What are “hybrid logic dependency structures”?

An OpenCCG sentence plan

Hybrid logic:

```
@x restaurant ^ @x <THEME> w ^ @y inexpensive ^
@y <THEME> w ^ @z attractive ^ @z <THEME> w ^
@w Giovanni's
```

XML:

```
<satop nomvar="X">
  <prop name="restaurant"/>
</satop>

<satop nomvar="X">
  <diamond mode="theme">
    <nomvar name="W"/>
  </diamond>
</satop>
```

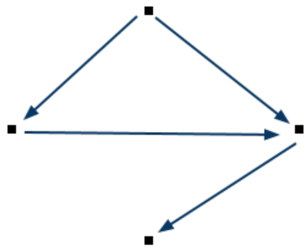
English:

“Giovanni's is an attractive inexpensive restaurant.”

Directed graphs

OpenCCG input representations are fundamentally **directed graphs**:

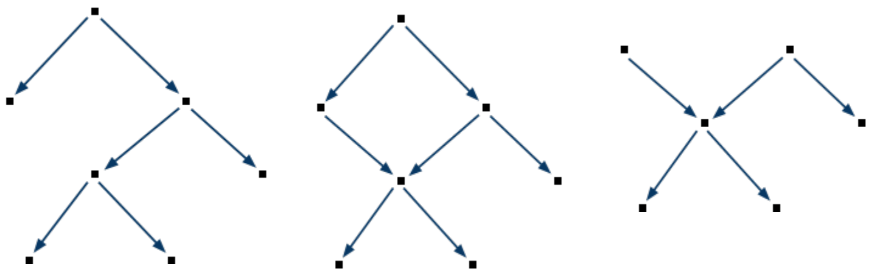
- **nodes** - "points"
- **edges** - "arrows" connecting two points



If there is an edge from node X to node Y, there is a **dependency** between entities X and Y

- i.e., Y is a dependent of X

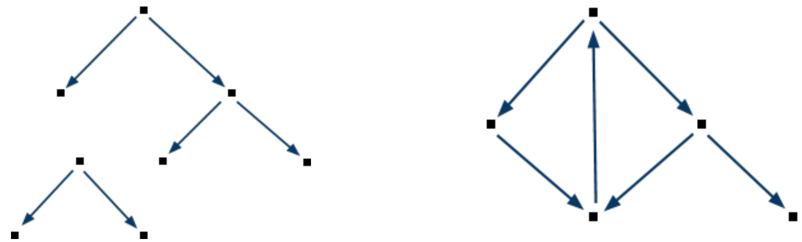
Directed graphs - topological constraints?



OpenCCG directed graphs can be either **trees** or **non-trees**.

“Re-entrancy” and “multi-rootedness” are permitted

Directed graphs - topological constraints?

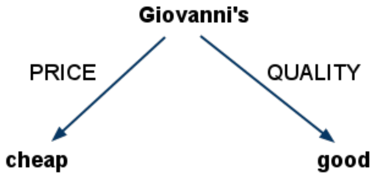


OpenCCG directed graphs can even be **non-connected** or **cyclic**

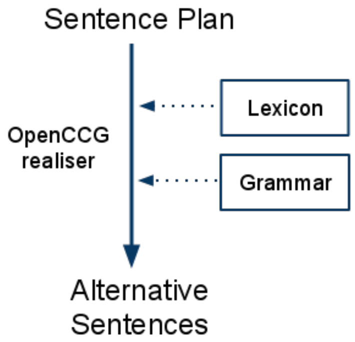
Labelled directed graphs

OpenCCG input representations are **labelled** directed graphs:

- **node labels** - different types of entity
- **edge labels** - different types of relation/dependency



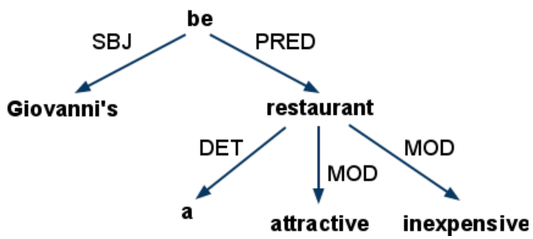
Surface realisation with OpenCCG



Sentence plan:

- a labelled directed graph
- can be **deep** or **surface** or anything in between

Surface sentence plans

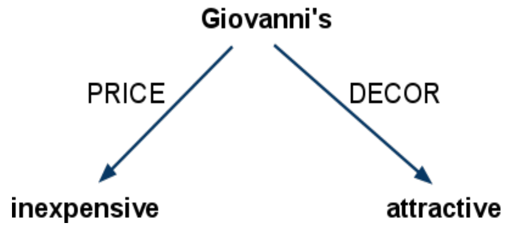


Basically a syntactic dependency structure.

Possible realisations are highly constrained:

- *Giovanni's is an attractive inexpensive restaurant.*
- *Giovanni's is an inexpensive attractive restaurant.*

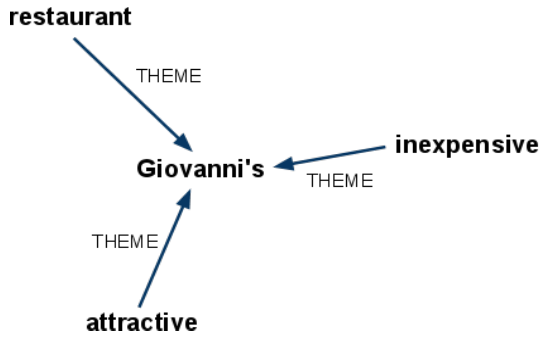
Deep sentence plans



Many possible realisations:

- *Giovanni's is an attractive inexpensive restaurant.*
- *Giovanni's is both inexpensive and attractive.*
- *Giovanni's does cheap food and has attractive decor.*
- *At Giovanni's, the food offers good value. Moreover, the decor is attractive.*

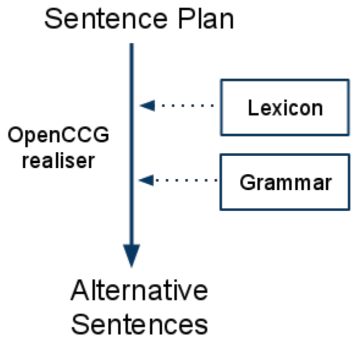
Intermediate sentence plans



More realisations than surface, but fewer than deep:

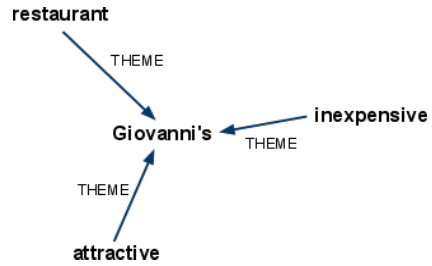
- *Giovanni's is an attractive inexpensive restaurant.*
- *Giovanni's is a restaurant that is cheap and attractive.*
- *Giovanni's, an attractive restaurant, serves inexpensive food.*

Surface realisation with OpenCCG



Sentence plan:

- a labelled directed graph



How can we **represent** labelled directed graphs?
i.e., we need a graph description language (a logic)

Graphs and logic

Think about first order logic.

Formulas:

- $\exists x\ \exists y.\ x \neq y \wedge boy(x) \wedge girl(y) \wedge love(x,y) \wedge \sim love(y,x)$

Every formula describes a set of **models**:

- the set of models in which the formula is true

Models are **graphs**!

Node labels are unary predicates (properties).
Edge labels are binary predicates (relations).

Graphs and logic

Want to encode labelled directed graphs in a linear format

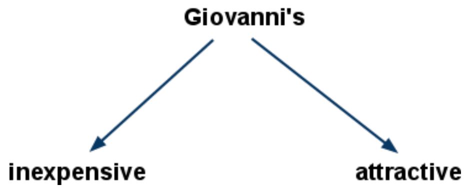
i.e., convert graph into a logic formula that describes *just that graph*

But which logical system shall we use to encode labelled directed graphs?

- first-order logic is way more expressive than we need
- **modal logic** is perfectly suited to describe graphs (Kripke structures) - hence used in OpenCCG

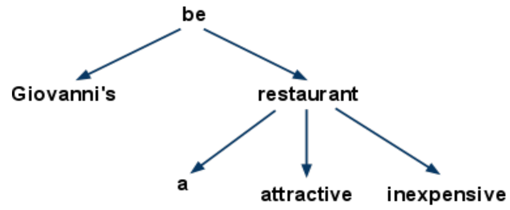
Describing directed graphs - modal propositional logic

The <> modal operator is used to signal a link between two nodes (i.e. a relation/dependency between two entities)




```
Giovanni's ^ (<> inexpensive) ^ (<> attractive)
```

Another example




```
be ^ (<> Giovanni's) ^ (<> (restaurant ^ (<> a) ^
(<> attractive) ^ (<> inexpensive)))
```

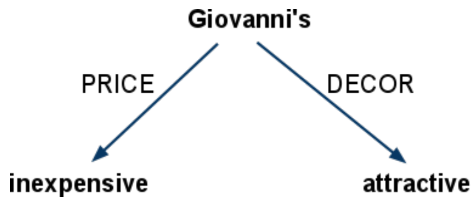
Using indentation instead of parentheses to show the relation between the graph and the formula more clearly

```
be
^ <> Giovanni's
^ <> restaurant
     ^ <> a
     ^ <> attractive
     ^ <> inexpensive
```

Describing *labelled* directed graphs - *multimodal* propositional logic

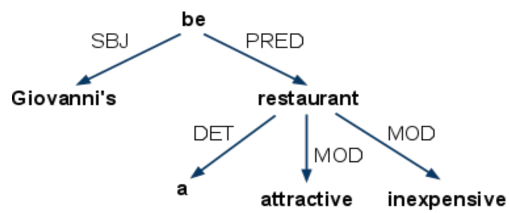
Instead of just one modal operator <>, there is a range of different multimodal operators, e.g., `<PRICE>, <DECOR>`

- denote different *types* of relation/dependency between two entities




```
Giovanni's ^ (<PRICE> inexpensive) ^ (<DECOR>
attractive)
```

Another example

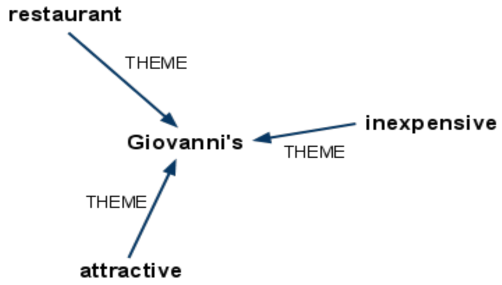



```
be ^ (<SBJ> Giovanni's) ^ (<PRED> (restaurant ^ (<DET> a)
   ^ (<MOD> attractive) ^ (<MOD> inexpensive)))
```

Using indentation:

```
be
^ <SBJ> Giovanni's
^ <PRED> restaurant
     ^ <DET> a
     ^ <MOD> attractive
     ^ <MOD> inexpensive
```

Graphs that are not trees?

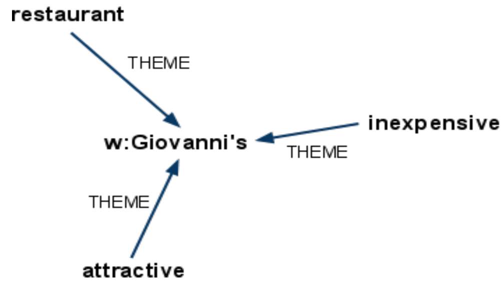



```
restaurant ^ <THEME> Giovanni's
inexpensive ^ <THEME> Giovanni's
attractive ^ <THEME> Giovanni's
```

But: normal modal logic has no way of ensuring that it is the **same** entity which is the THEME in all three cases.

Also, no way of combining the three fragments into a single formula - conjunction won't do.

Hybrid multimodal logic - nominals



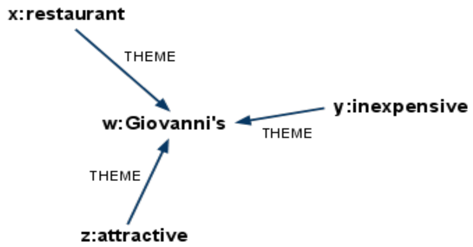

```
restaurant ^ <THEME> (w ^ Giovanni's)
inexpensive ^ <THEME> w
attractive ^ <THEME> w
```

Nominals capture reentrancy, but not multi-rootedness.

Hybrid multimodal logic - @ operator

```
@x restaurant

@y <THEME> Giovanni's
```




```
(@x restaurant ^ <THEME> (w ^ Giovanni's))^
(@y inexpensive ^ <THEME> w)^
(@z attractive ^ <THEME> w)
```

Elementary predications

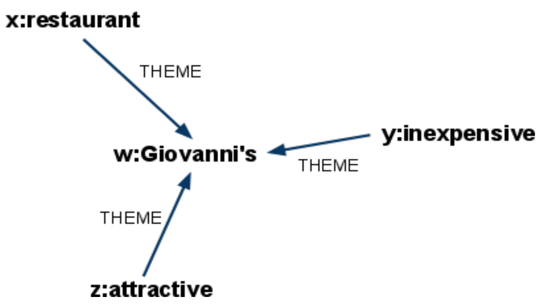
Every hybrid logic formula can be turned into an equivalent **conjunction** of **elementary predications (EP)**

Two kinds of EP:

1. Node label statements:
- `@x restaurant`
- node x is labelled “restaurant”

2. Edge statements:
- `@x <THEME> y`
- there is an edge labelled “theme” from node x to node y

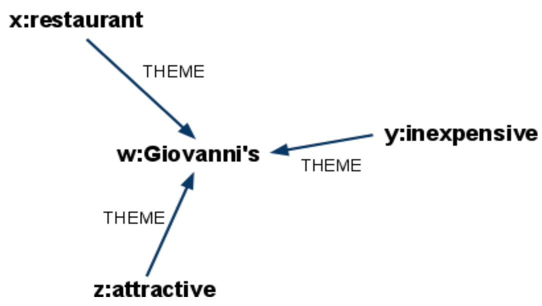
Elementary predications



Hybrid logic formula:

```
  (@x restaurant ^ <THEME> (w ^ Giovanni's))
^ (@y inexpensive ^ <THEME> w)
^ (@z attractive ^ <THEME> w)
```

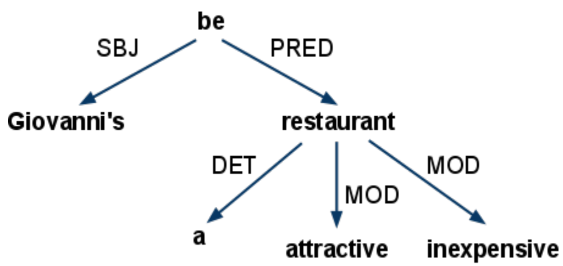
Elementary predications



Conjunction of EPs:

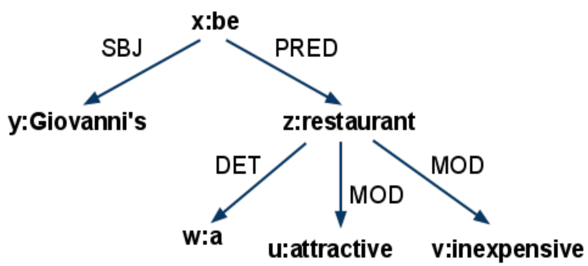
```
@x restaurant ^ @y inexpensive ^ @z attractive
^ @w Giovanni's ^ @x <THEME> w ^ @y <THEME> w
^ @z <THEME> w
```

Another example




```
be ^ (<SBJ> Giovanni's) ^ (<PRED> (restaurant ^
(<DET> a) ^ (<MOD> attractive) ^ (<MOD>
inexpensive)))
```

Another example




```
@x be ^ @y Giovanni's ^ @z restaurant ^ @w a ^
@u attractive ^ @v inexpensive ^ @x <SBJ> y ^
@x <PRED> z ^ @z <DET> w ^ @z <MOD> u ^
@z <MOD> v
```

Elementary predications in XML

Node label statements: `@x attractive`

```
<satop nomvar="X">
  <prop name="attractive"/>
</satop>
```

Edge statements: `@x <THEME> y`

```
<satop nomvar="X">
  <diamond mode="theme">
    <nomvar name="Y"/>
  </diamond>
</satop>
```

What you need to know

How to convert a labelled directed graph into a set of elementary predications of hybrid multimodal logic

How to convert a set of elementary predications of hybrid multimodal logic into a labelled directed graph

- Reading for Week 2:
  - Michael White. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Research on Language and Computation,* 4(1):39–75, 2006.

Learn more about hybrid logic

Patrick Blackburn (2000): "Representation, Reasoning and Relational Structures: a Hybrid Logic Manifesto". Logic Journal of the IGPL, 8(3), 339-365.
URL: http://www.loria.fr/~blackbur/papers/manifesto.pdf

Patrick Blackburn (1993): "Modal Logic and Attribute Value Structures". In *Diamonds and Defaults*, edited by M. de Rijke, Kluwer Academic Publishers, 1993, pages 19-65.
URL: http://www.loria.fr/~blackbur/papers/attribute.pdf