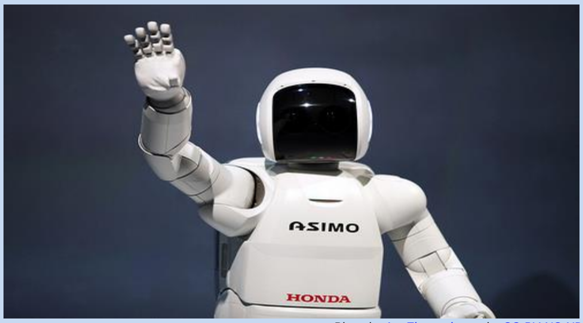
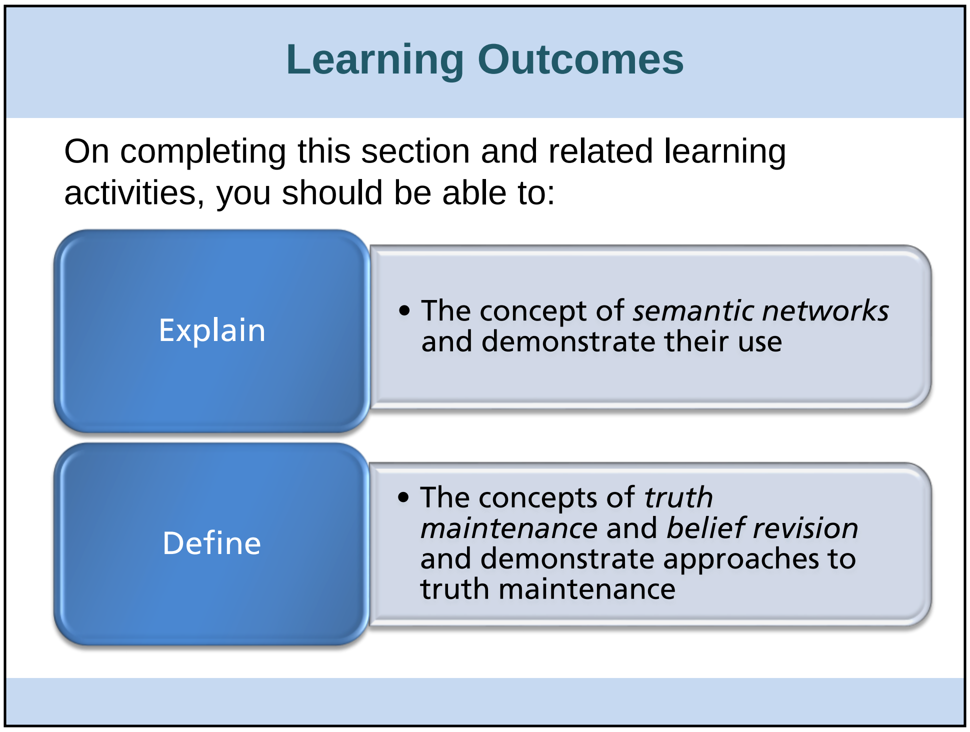


Photo by Ars Electronica under CC-BY-NC-ND.

Knowledge Representation Part 3

Workshop 4 Section 3

CT621 Artificial Intelligence - MScSED



Learning Outcomes

On completing this section and related learning activities, you should be able to:

Explain	• The concept of *semantic networks* and demonstrate their use
Define	• The concepts of *truth maintenance* and *belief revision* and demonstrate approaches to truth maintenance

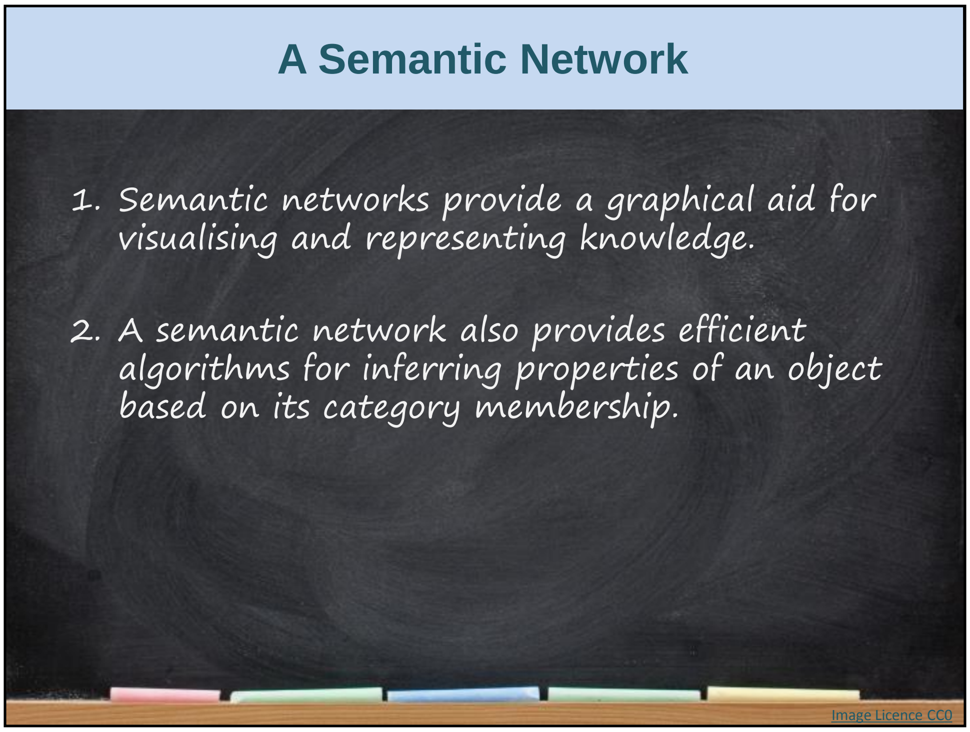
This workshop will begin by introducing the concept of a *semantic network* and will demonstrate how they are used to visually represent knowledge. It will then explore the area of *truth maintenance* and *belief revision*.

Foundation of Semantic Networks

- Foundation of Semantic Networks first introduced by Charles S. Pierce
- In 1909, he proposed a graphical notation of nodes and edges called "existential graphs“
- He called it "the logic of the future"

Charles S. Pierce
Photograph under Public Domain

Let’s begin with some background on Semantic Networks.



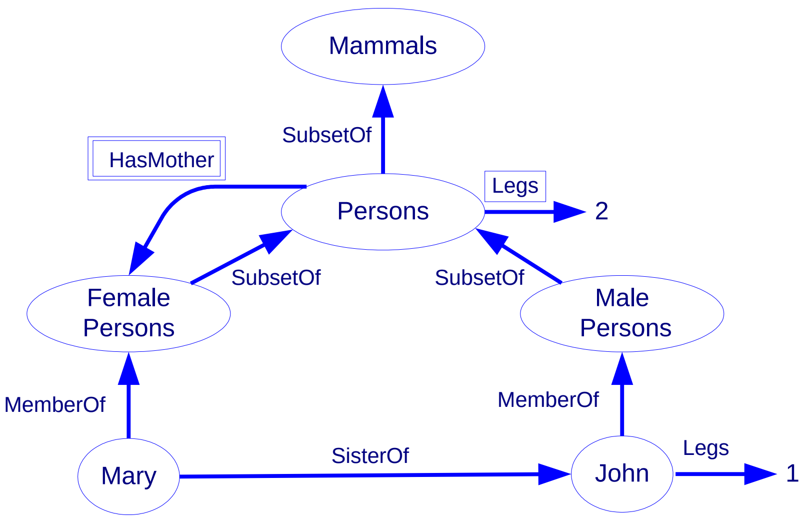
A Semantic Network

1. Semantic networks provide a graphical aid for visualising and representing knowledge.
2. A semantic network also provides efficient algorithms for inferring properties of an object based on its category membership.

Image Licence CC0

A semantic network can be visually depicted as a set of nodes that are connected by labelled arcs. The nodes represent objects and categories and the arcs represent relations between concepts.

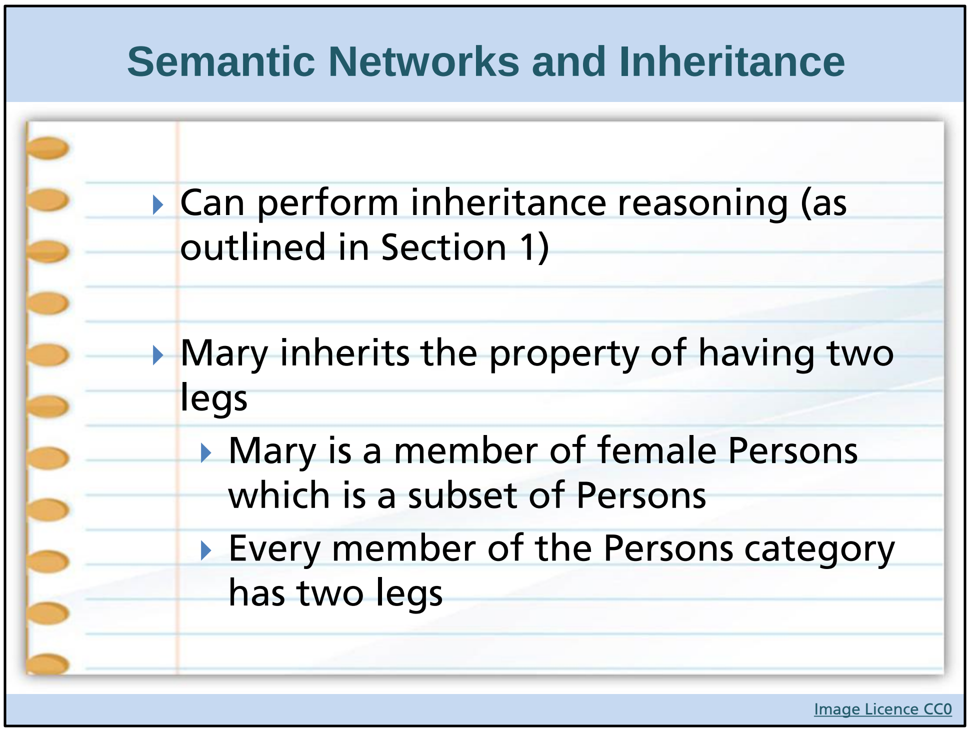
A Semantic Network Example



An example of a semantic network is shown in the slide; it is capable of representing objects (e.g. `Mary`), categories (e.g. `Female Persons`), and relations among objects (e.g. `SisterOf`). Therefore, from the diagram is easy to determine that the object `Mary` is a member of the category `FemalePersons`. This category is in turn a subset of the category `Persons`.

Note that a double-boxed link is used for the `HasMother` relation between `Persons` and `FemalePersons`. This means that every member of the `Persons` category has a `HasMother` relation with another object that is a member of the `FemalePersons` category (every person has a mother).

The special notation is used because we cannot use the `HasMother` relation between categories, i.e. categories do not have mothers.
Similarly, single-boxed notation is used when one side of the relation is a category and the other side is an object, e.g. every member of the `Persons` category has two legs.



Semantic Networks and Inheritance

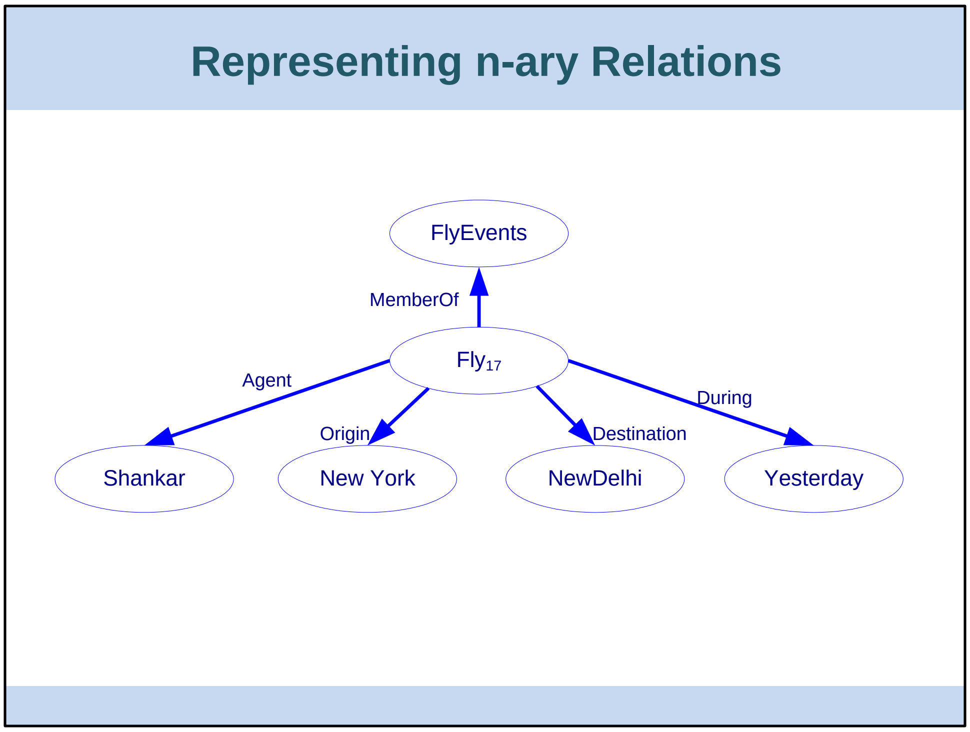
- Can perform inheritance reasoning (as outlined in Section 1)
- Mary inherits the property of having two legs
  - Mary is a member of female Persons which is a subset of Persons
  - Every member of the Persons category has two legs

Image Licence CC0

Not only do semantic networks provide graphical aids for representing knowledge, but they also provide algorithms for inferring properties of an object based on its category membership. The semantic network notation easily facilitates reasoning based on inheritance. We already referred to this type of reasoning in Section 1 of this workshop. In the example depicted on the previous slide, Mary inherits the property of having two legs.

The inheritance algorithm will follow the `memberOf` link from Mary to the `Female Persons` category. It will then follow the subset link between the `Female Persons` category and the `Persons` category. The `Persons` category has a property that each of its members has two legs. Therefore, the algorithm can infer that Mary has two legs.

It should be noted that inheritance can become complicated when multiple inheritance is facilitated. Multiple inheritance occurs when one category can be a subset of more than one other category. In such scenarios it is possible that the inheritance algorithm may face conflicting properties. For this reason, multiple inheritance is banned under some Object Oriented Programming languages such as Java.

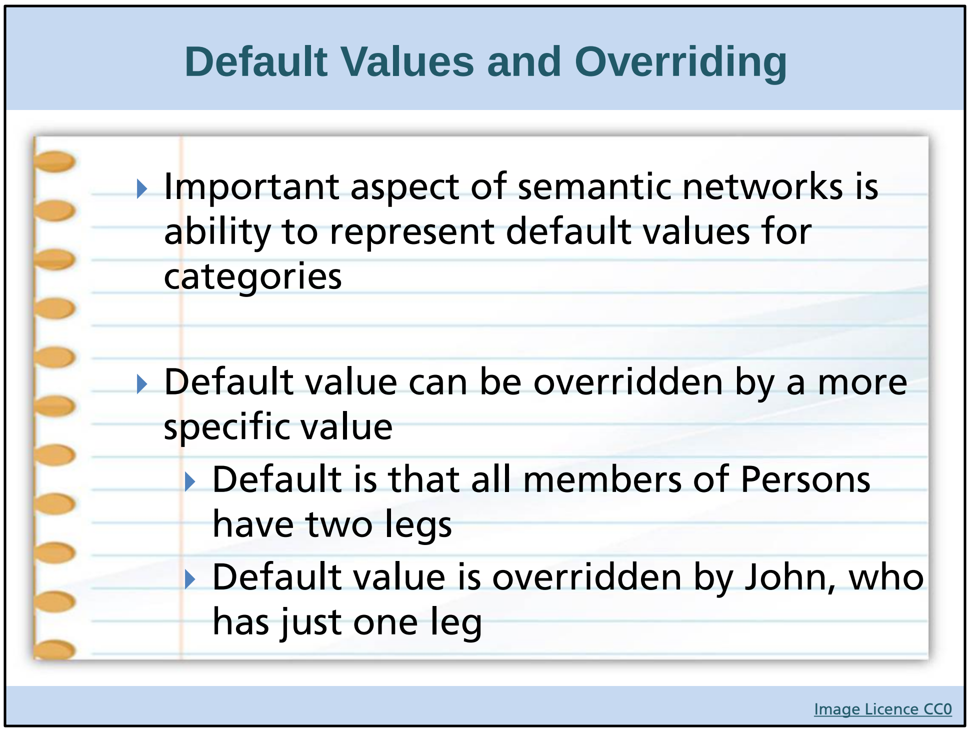


Representing n-ary Relations

One restriction of semantic networks is that they are limited to representing binary relations. They cannot represent *n-ary* relations such as `Fly(Shankar, NewYork, NewDelhi)`.

To handle this, we change the proposition itself to an event belonging to an event category. In this way, semantic networks force the creation of many reified concepts; much of the ontology used in this workshop originated from a semantic network system. Note that the event we create (by reifying the proposition) is identified as $\texttt{Fly}_{17}$ – the number 17 here is an arbitrary assignment that uniquely identifies this event from others, e.g. other members of the `FlyEvents` category; we could have also identified by another name, such as $\texttt{Fly}_{\texttt{Shan}}$.

Reification is the representation of facts and/or assertions.

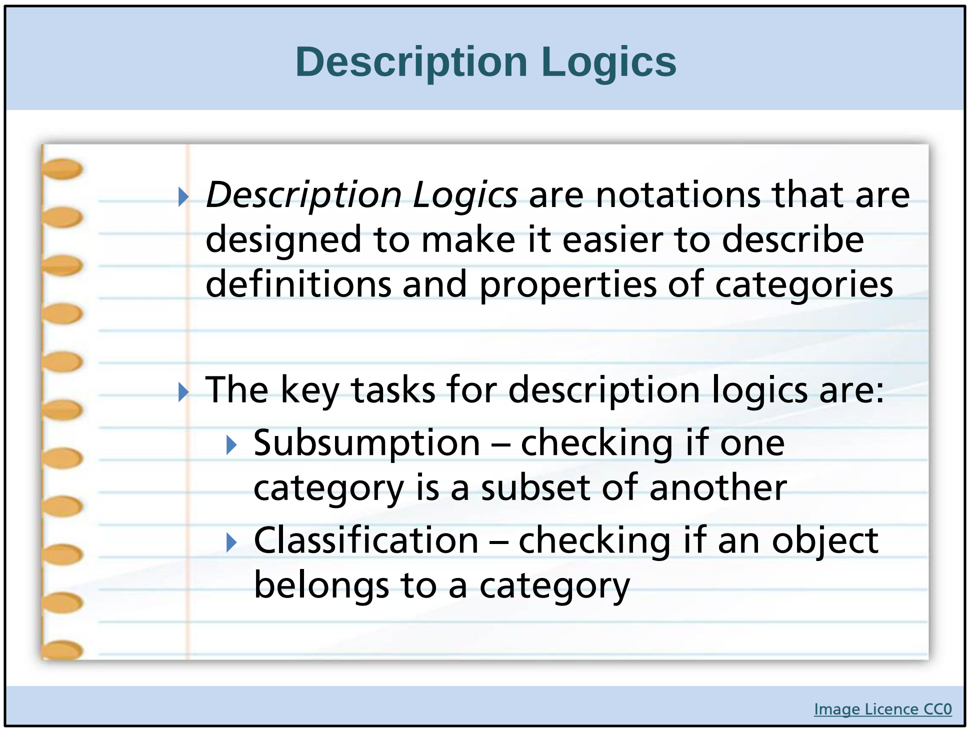


Default Values and Overriding

- Important aspect of semantic networks is ability to represent default values for categories
- Default value can be overridden by a more specific value
  - Default is that all members of Persons have two legs
  - Default value is overridden by John, who has just one leg

An important feature of semantic networks is the ability to represent default values for categories. For example, we know that the default value for the `Female Persons` category is that all members have two legs. Therefore, a `Person` is a assumed to have two legs unless this is contradicted by more specific information. Notice that the default property that each `Person` has two legs is overridden in the case of John. The semantic network specifies that John has only a single leg.

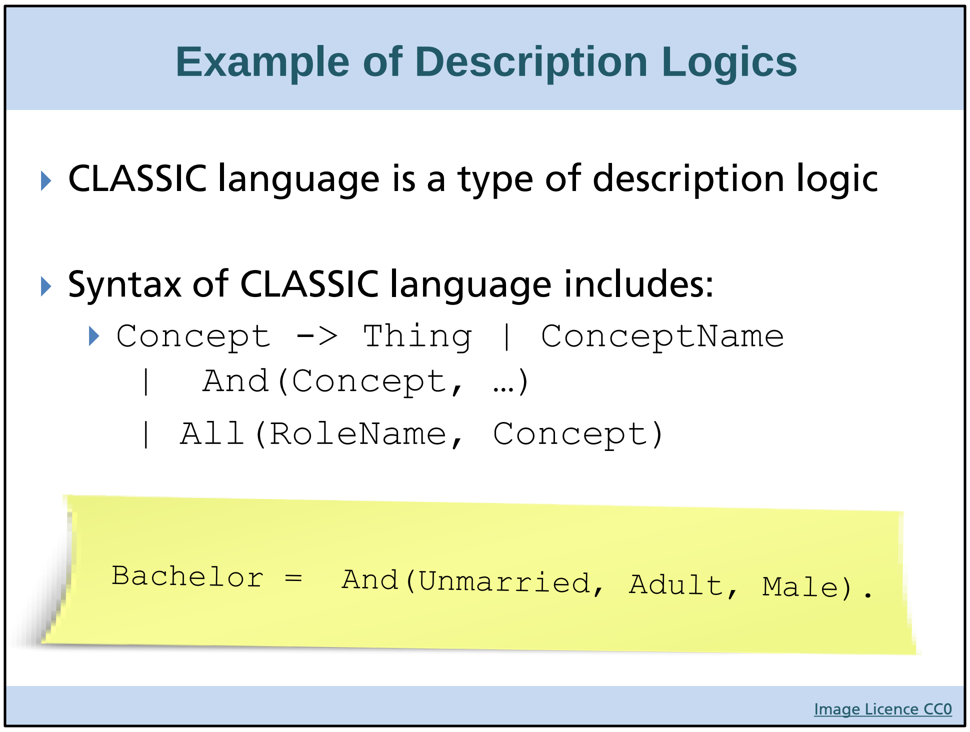
The default semantics is enforced by the inheritance algorithm because it follows links upwards from the object itself and stops as soon as it finds a value.



Description Logics

- *Description Logics* are notations that are designed to make it easier to describe definitions and properties of categories
- The key tasks for description logics are:
  - Subsumption – checking if one category is a subset of another
  - Classification – checking if an object belongs to a category

The syntax of first-order logic makes it easy to make statements about objects. *Description Logics*, which has evolved from semantic networks are notations that are designed to make it easier to describe definitions and properties of categories.



Example of Description Logics

- CLASSIC language is a type of description logic
- Syntax of CLASSIC language includes:

  -
    ```
    Concept -> Thing | ConceptName
      |  And(Concept, …)
      | All(RoleName, Concept)
    ```

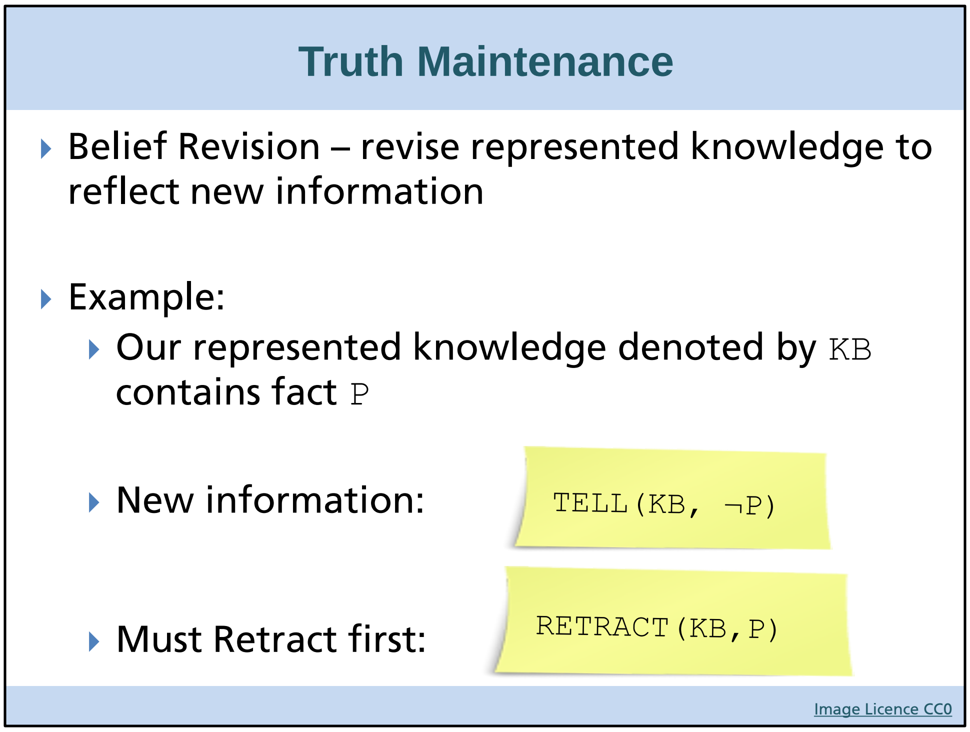
```
Bachelor =  And(Unmarried, Adult, Male).
```

Image Licence CC0

The Classic language is an example of description logic. A part of the syntax for this language is shown on the slide.

A sample statement of this language is shown in the slide. This statement in First-order logic would be: `Bachelor(x) <=> Unmarried(x) ^ Adult(x) ^ Male(x).`

Any description in *CLASSIC* can be translated into first order logic. However, CLASSIC can provide a intuitive way of representing more complex statements.



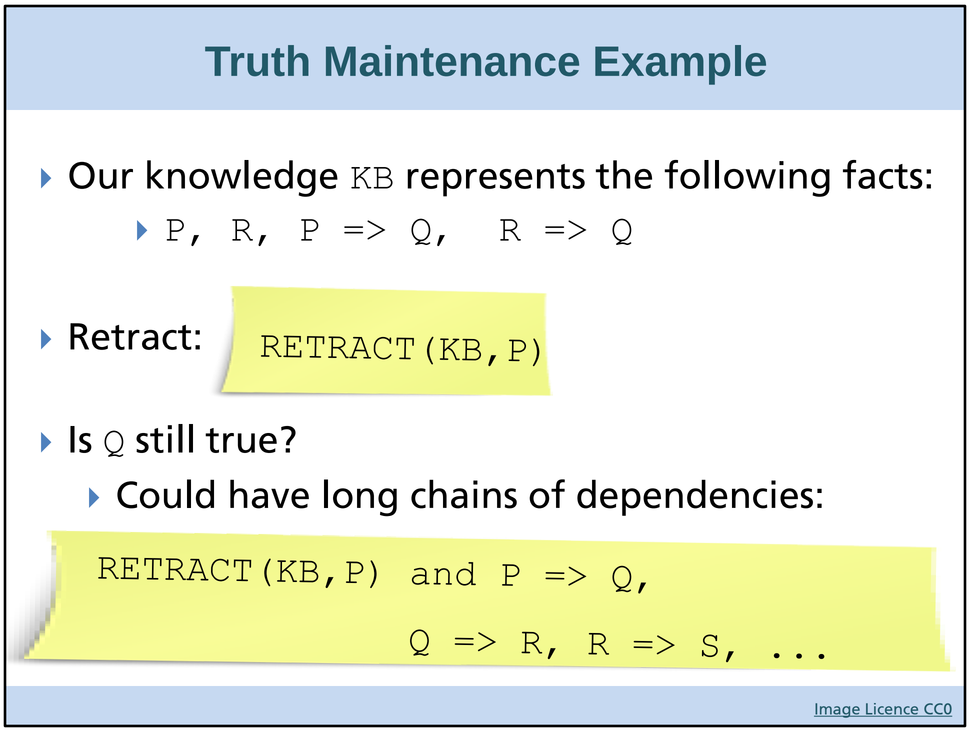
Truth Maintenance

- Belief Revision – revise represented knowledge to reflect new information
- Example:
  - Our represented knowledge denoted by `KB` contains fact `P`
  - New information: `TELL(KB, ¬P)`
  - Must Retract first: `RETRACT(KB,P)`

Image Licence CC0

In the Workshop 4 Section 1 we demonstrated how inferences are drawn through default reasoning. However, some of these inferred facts could turn out to be contradicted when new information is included. These inferred facts would then have to be retracted, a process known as belief revision.

As an example, assume that the knowledge represented, which we denote as `KB`, contains the fact `P`. This fact could be a default conclusion. We have new information that states that `P` is false, so we want to execute `Tell(KB, ¬P)`. Before doing this, we must retract `P` in order to avoid contradiction. However, in addition to the retraction of `P`, we must also retract any sentences inferred from `P`. For example, if the knowledge we represented expressed that `P => Q`, (and subsequently inferred that `Q` is true) `Q` would no longer be true because of this retraction.



Truth Maintenance Example

- Our knowledge `KB` represents the following facts:
  - `P, R, P => Q, R => Q`
- Retract: `RETRACT(KB,P)`
- Is `Q` still true?
  - Could have long chains of dependencies:

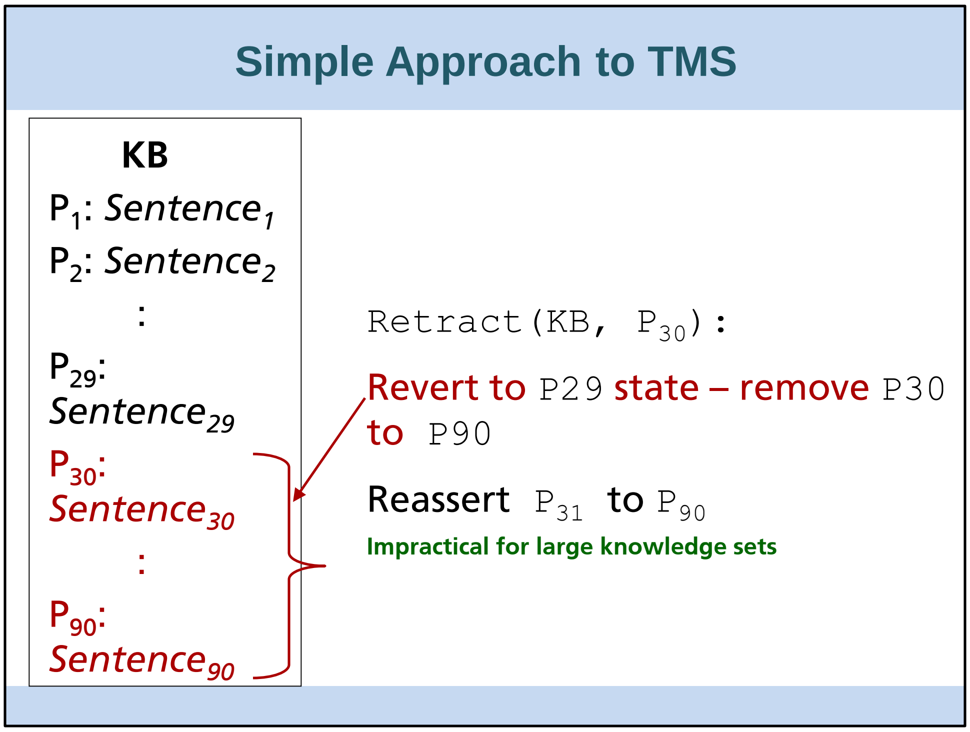
```
RETRACT(KB,P) and P => Q,
                  Q => R, R => S, ...
```

Image Licence CC0

Take a similar example to the previous slide. It contains the facts listed above. Again, faced with new information that `P` is false, we want to retract `P`. What can we say about `Q` in this case?

In this case, `Q` is still true because `R` is true and `R` ⇒ Q.

A system capable of handling the addition of new facts that contradict existing information in the KB is known as a *truth maintenance system (TMS).* Note that in contrast to the two simple examples covered here, a TMS may have to deal with a long list of dependencies. For example, we want to retract `P`, but `P` ⇒ `Q`, Q ⇒ `R`, `R` ⇒ S and so on. As you can see, this could be quite a complicated process. In the last couple of slides, we will cover two approaches to truth maintenance.



Simple Approach to TMS

KB

P_1: *Sentence₁*

P_2: *Sentence₂*

:

P_{29}: *Sentence₂₉*

P_{30}: *Sentence₃₀*

:

P_{90}: *Sentence₉₀*

`Retract(KB, P`$_{30}$`):`

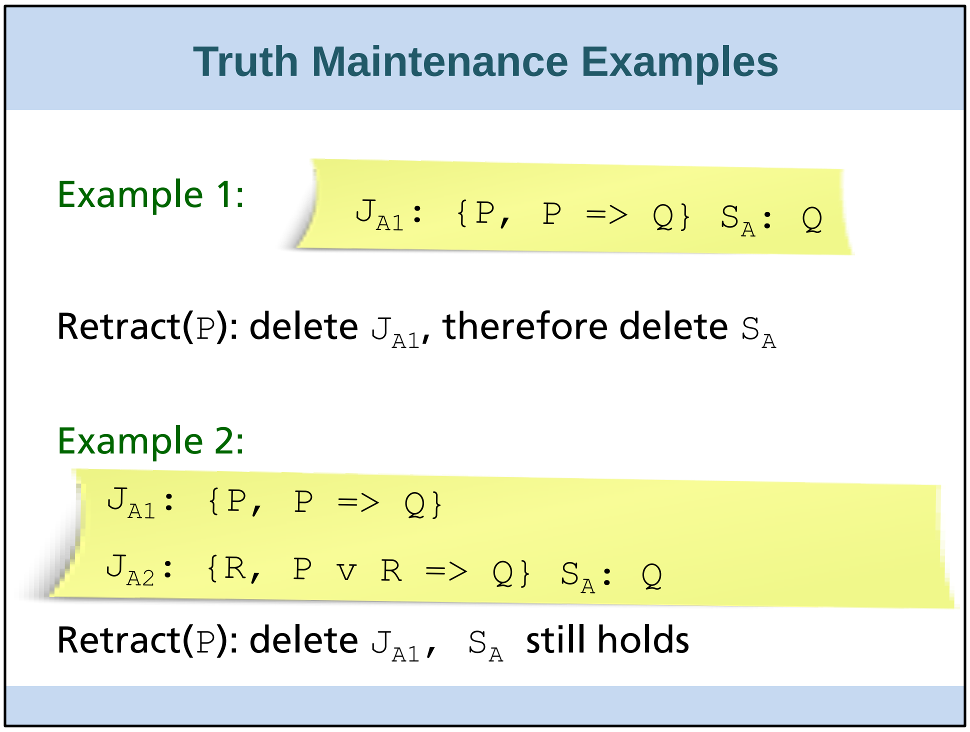
Revert to `P29` state – remove `P30` to `P90`

Reassert P_{31} to P_{90}

Impractical for large knowledge sets

One approach to TMS is to keep track of the order in which sentences are added to our knowledge, by numbering them from P_1 to P_n as shown in the slide.

When we want to retract P_i, for example, we remove all sentences from our knowledge that were added since P_i was added, including P_i itself. This ensures that anything derived from P_i is also removed. Once this has been done, the sentences P_{i+1} to P_n can be added again. This process guarantees that the knowledge base will be consistent. However, this approach is somewhat impractical for handling a large knowledge base, to which many facts would be added.



Truth Maintenance Examples

Example 1:

`J`$_{A1}$`: {P, P => Q} S`$_A$`: Q`

Retract(`P`): delete `J`$_{A1}$, therefore delete `S`$_A$

Example 2:

`J`$_{A1}$`: {P, P => Q}`

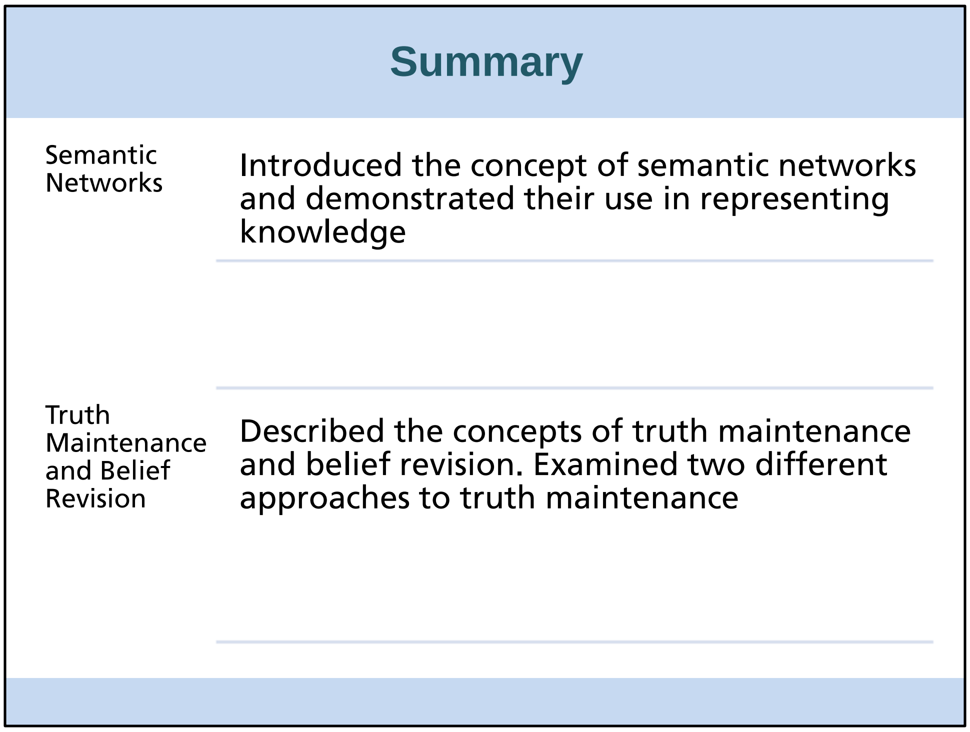
`J`$_{A2}$`: {R, P v R => Q} S`$_A$`: Q`

Retract(`P`): delete `J`$_{A1}$, `S`$_A$ still holds

An example of a more efficient approach to truth maintenance, *Justification-based TMS (JTMS)*, is shown here. With each sentence in our collection of knowledge, a *justification* is stored – the justification consists of the set of sentences from which the sentence was inferred.

For example, the sentence `Q` has the justification `J`$_{A1}$. As you can see, `Q` holds because `P` is true and `P` ⇒ `Q`. If `P` is retracted, the sentence `S`$_A$ would be removed, as its justification `J`$_{A1}$ no longer holds. The JTMS will delete all sentences for which `P` is a member of every justification. Note that one sentence may have multiple justifications, as is the case in the second example shown on this slide.

In the second example, we have the same sentence, but this time it has two justifications `J`$_{A1}$ and `J`$_{A2}$. When P is retracted, `J`$_{A1}$ no longer holds, but `J`$_{A2}$ is still true, so the sentence `S`$_A$ is retained.



Summary

Semantic Networks	Introduced the concept of semantic networks and demonstrated their use in representing knowledge
Truth Maintenance and Belief Revision	Described the concepts of truth maintenance and belief revision. Examined two different approaches to truth maintenance

References

ArsElectronica (2010) ASIMO [photo], (CC BY-NC-ND 2.0), available: https://www.flickr.com/photos/36085842@N06/4945217670/ [accessed 18 Apr 2019].

All CC0 licensed images were accessed from Articulate 360’s Content Library