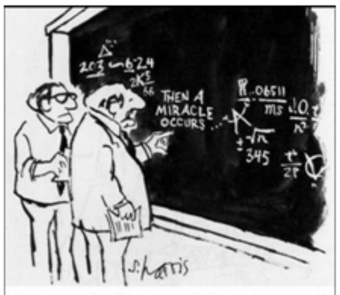
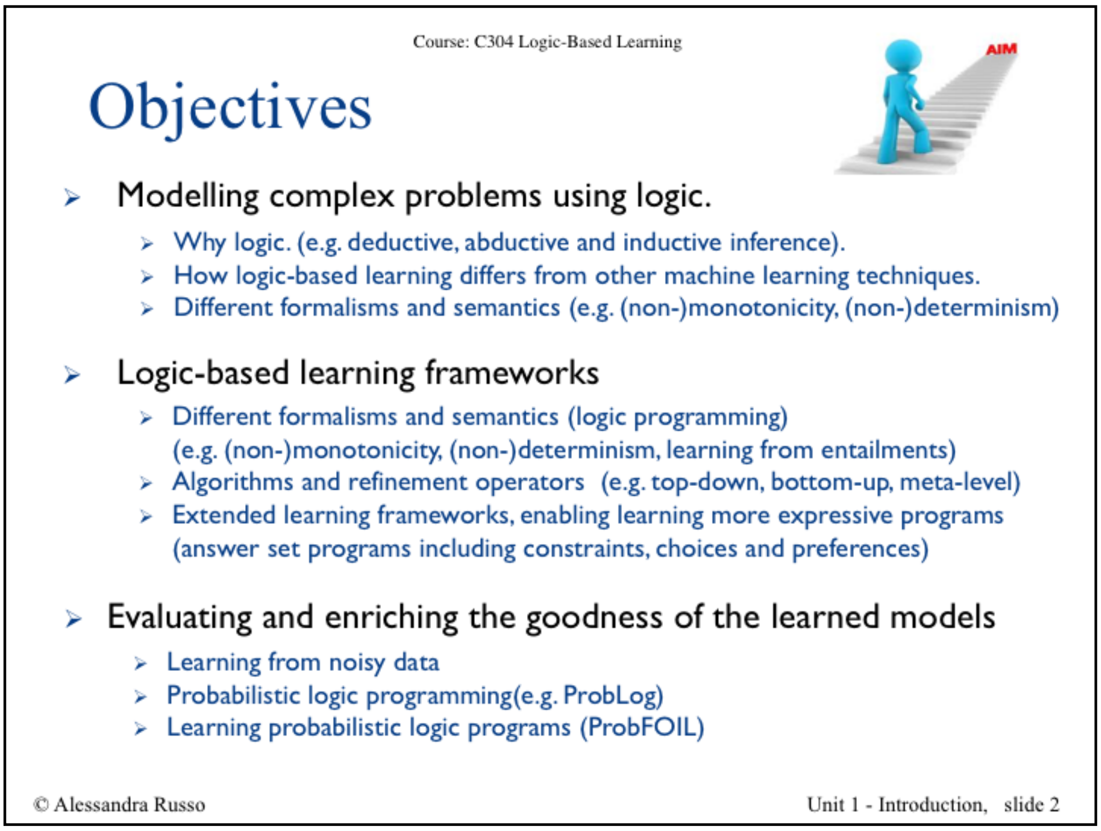
Logic-based Learning

Alessandra Russo and Mark Law

{a.russo, mark.law}@imperial.ac.uk



"I think you should be more explicit here in step two."


Course: C304 Logic-Based Learning

Objectives

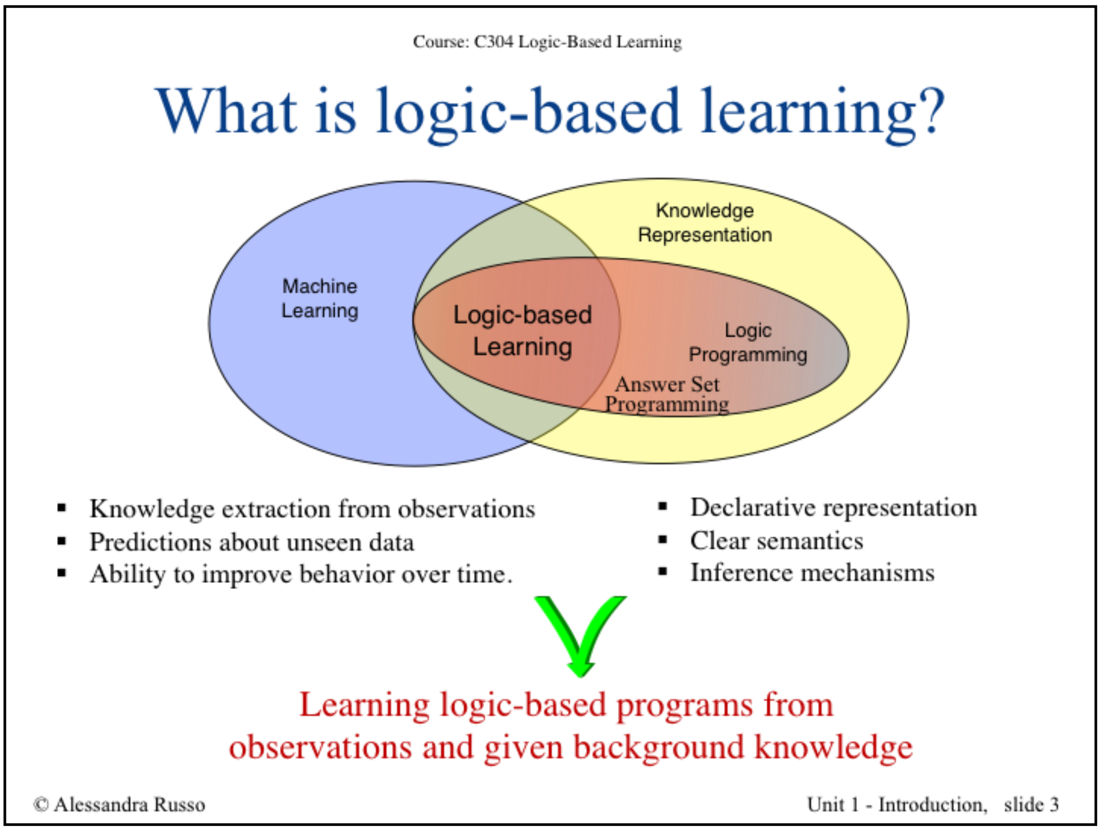
- Modelling complex problems using logic.
  - Why logic. (e.g. deductive, abductive and inductive inference).
  - How logic-based learning differs from other machine learning techniques.
  - Different formalisms and semantics (e.g. (non-)monotonicity, (non-)determinism)
- Logic-based learning frameworks
  - Different formalisms and semantics (logic programming) (e.g. (non-)monotonicity, (non-)determinism, learning from entailments)
  - Algorithms and refinement operators (e.g. top-down, bottom-up, meta-level)
  - Extended learning frameworks, enabling learning more expressive programs (answer set programs including constraints, choices and preferences)
- Evaluating and enriching the goodness of the learned models
  - Learning from noisy data
  - Probabilistic logic programming(e.g. ProbLog)
  - Learning probabilistic logic programs (ProbFOIL)

© Alessandra Russo   Unit 1 - Introduction, slide 2

Machine Learning is a wide and important field of Artificial Intelligence that is concerned with the construction of *computer programs that automatically improve with experience*. It has been initially promoted by the interest of automatically searching for patters in large dataset, with the objective of understanding (i.e. acquire knowledge about) the context of the dataset and use this new “hidden” knowledge to compute better solutions to given problems (e.g. decide retail strategies based on predictions of consumer behaviour to maximise profits). Within the past 50 years, this initial effort has lead to the development of more sophisticated methods, based on a variety of computational and statistical techniques, for automatically extracting new information from (large) databases, consequently widening its focus. The objective of Machine Learning systems should be not only to identify relations within data, but also to use this information, together with new data to make predictions and also to evolve the learned knowledge in order to improve their task with experience over time. But the increased complexity of the data and the context in which these data are collected has also highlighted limitations of traditional machine learning and data mining approaches in their representation language and ability to incorporate contextual and prior knowledge.

The **aims** of this course are to
- learn the underlying principles of logic-based machine learning
- familiarize with state-of-the-art learning algorithms,
- develop basic skills for formalizing a learning task to solve a given learning problem,
- explore frameworks for learning richer programs and from noisy data,
- introduce recent advancements on probability logic-based inference,
- explore learning in the context on probabilistic logic-based learning.

At the end of this course you will have acquired the skills needed to formalise a given learning problem in the appropriate computational learning framework, solve it, and compare pro and cons of the learned outcomes.


Course: C304 Logic-Based Learning

What is logic-based learning?

Machine Learning
Knowledge Representation
Logic-based Learning
Logic Programming
Answer Set Programming

- Knowledge extraction from observations
- Predictions about unseen data
- Ability to improve behavior over time.

- Declarative representation
- Clear semantics
- Inference mechanisms

Learning logic-based programs from observations and given background knowledge

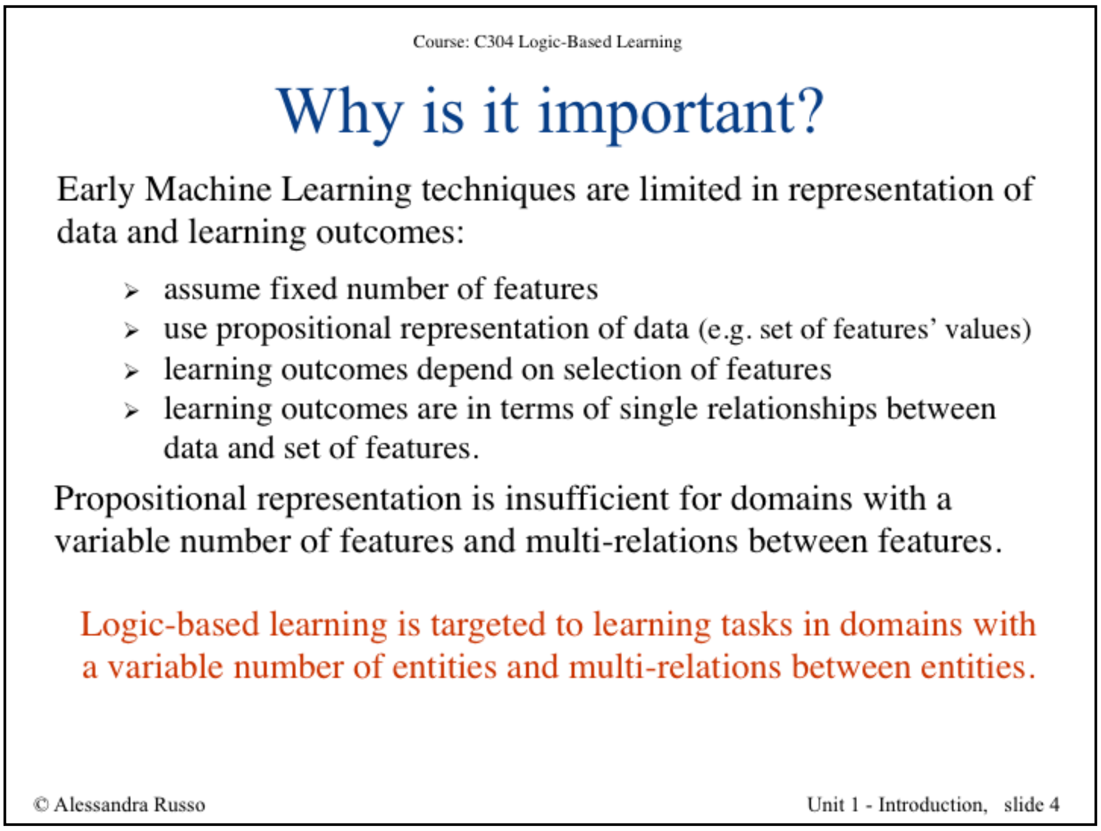
© Alessandra Russo

Unit 1 - Introduction, slide 3

The Artificial Intelligence field has witnessed within the last 30 years the establishment of a subfield, called Computational Logic, which advocates the use of logic-based formalisms (e.g. Datalog, Horn clauses, first-order predicate logic) for representing knowledge when modeling and reasoning about real-world problems. Logic provides a declarative way for expressing concepts, a clear semantics, sound inference mechanisms and a variety of systems and tools capable of performing automated reasoning. Logic Programming is a particular subset of Computational Logic that benefits from a programming paradigm based on formal logic. Programs are defined as sets of sentences in a given logic-based language expressing facts and rules about a given problem domain (see wiki definition). Different logic programming languages have been developed, and the most common are Datalog, Prolog, Answer Set Programming (ASP). They each have specific characteristics and a formal underpinning computational semantics that precisely define the truth and falsity of the consequences of a given program and computed solutions.

A parallel field of Artificial Intelligence is Machine Learning, where the focus is instead on the study of systems capable of extracting knowledge hidden in observed data, in order to make predictions on unseen data and use these predictions to improve their behavior over time.

Logic-based learning is a research area at the intersection of Knowledge Representation and Machine Learning, which looks at methods for constructing logic-based models (programs) from examples and existing background knowledge. The last two decades have seen a proliferation of logic-based learning frameworks, with specific algorithms and systems, for different classes of logic-based models. *These approaches inherit from the knowledge representation field the representation formalism and semantics of the logic-based models they are set to learn, and from Machine Learning the empirical orientation of building intelligent learning machines that are capable of generating hypothesis and knowledge from examples.* In the case of logic-based learning, these intelligent learning machines are logic-based programs that can learn logic-based models and revise them over time to perform given tasks more efficiently and more accurately. A particular subset of logic-based learning is Inductive Logic Programming (ILP) and Answer Set Programming (ASP), where the logic-based models that are learned are logic programs and Answer Set Programs.



Course: C304 Logic-Based Learning

Why is it important?

Early Machine Learning techniques are limited in representation of data and learning outcomes:

- assume fixed number of features
- use propositional representation of data (e.g. set of features' values)
- learning outcomes depend on selection of features
- learning outcomes are in terms of single relationships between data and set of features.

Propositional representation is insufficient for domains with a variable number of features and multi-relations between features.

Logic-based learning is targeted to learning tasks in domains with a variable number of entities and multi-relations between entities.

© Alessandra Russo

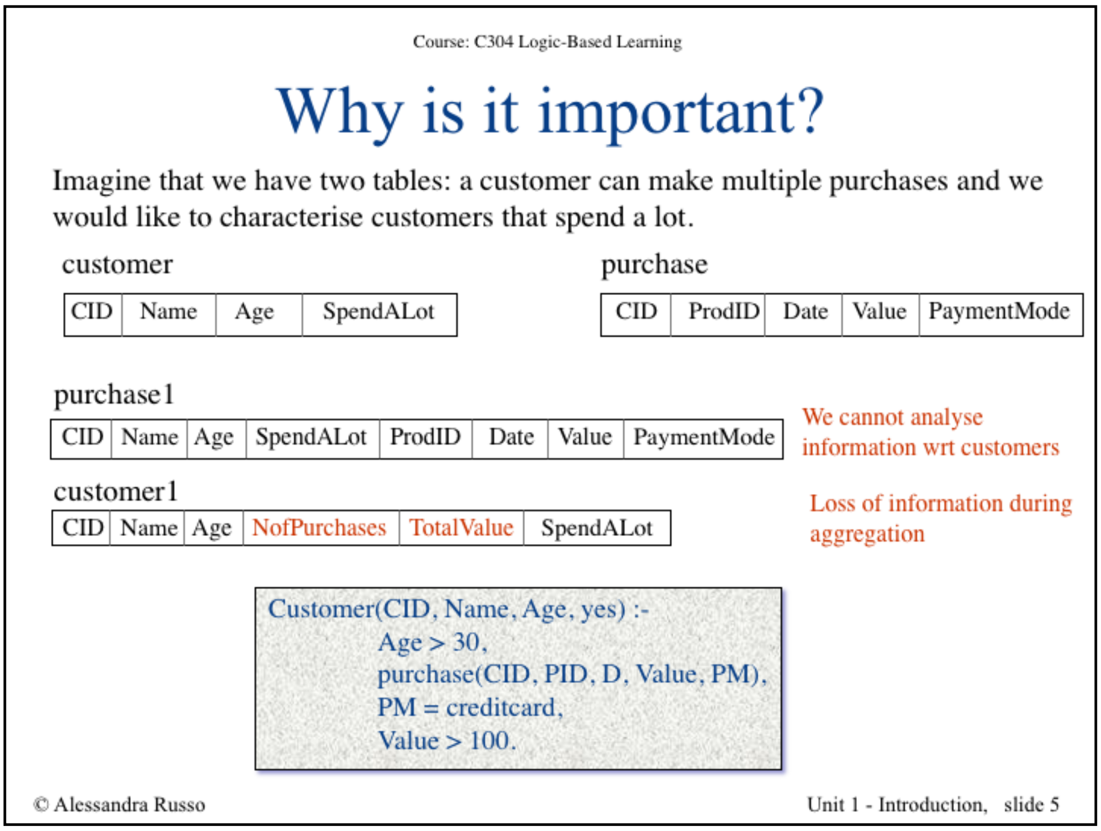
Unit 1 - Introduction, slide 4

As also stated in Luc De Raedt's textbook, in the past 50 years a variety of Machine Learning techniques have been developed (see Mitchel 1997 book). However, most of the early approaches are limited in their knowledge representation. Early decision trees and Bayesian network can handle data that are expressed as collection of Boolean values (i.e. features). *They cannot handle domains involving multiple entities and multiple relationship between entities*. This limits also their resulting learning outcomes and the application domains where these techniques can be applied. These types of Machine Learning approaches are also referred to as "Propositional Learning" approaches: they aim to find patterns/models in a given single table, where each row corresponds to a data point and columns express the values of a fix number of features for that data point. One raw in this table is the classification value (i.e. label) observed for that data point.

Such limitations were pointed out in the early 1980 when researchers such as Plotkin and Michalski started to explore the use of more expressive languages for learning. *Since then, the focus has been on how to develop general learning systems that are capable of handling domains where the number of variables (or features) can vary from dataset to dataset and the relationships between features may be multiple* (e.g. in social networks person's features may vary from person to person and social interaction may be multiple*).*

Logic-based learning can provide solutions to these limitations. First-order (or predicate) logic is more expressive than propositional logic as it can capture different types (or number) of domains of discourses and can express multiple relationship between objects in the domain through predicates (i.e. relations) of different arities and through rules that logically combine different predicates. It can be seen as an example of a larger class of learning approaches, called *relational learning*, for which the patterns/models they find are expressed in relational formalisms of first-order logic.

For further reading on historical background on Machine Learning, you could read Chapter 1 of "Foundations of Rule Learning" by Johannes Fürnkranz, Dragan Gamberger, Nada Lavrač.


Course: C304 Logic-Based Learning

Why is it important?

Imagine that we have two tables: a customer can make multiple purchases and we would like to characterise customers that spend a lot.

customer

CID	Name	Age	SpendALot

purchase

CID	ProdID	Date	Value	PaymentMode

purchase1

CID	Name	Age	SpendALot	ProdID	Date	Value	PaymentMode

We cannot analyse information wrt customers

customer1

CID	Name	Age	NofPurchases	TotalValue	SpendALot

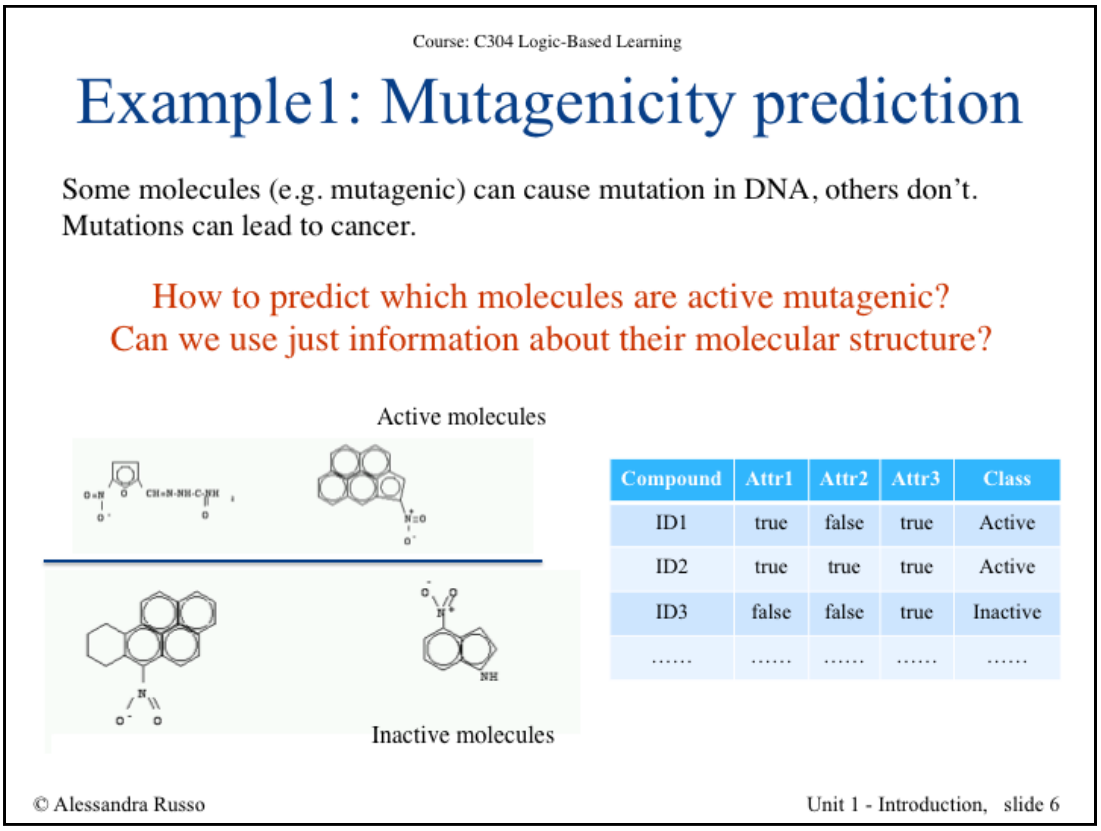
Loss of information during aggregation

```
Customer(CID, Name, Age, yes) :-
        Age > 30,
        purchase(CID, PID, D, Value, PM),
        PM = creditcard,
        Value > 100.
```

© Alessandra Russo

Unit 1 - Introduction, slide 5

Consider for instance the above simple problem of data mining, where the task is to analyse data and find rules that allow to make inference of customers that spend a lot. The dataset may include multiple tables (e.g. customer, purchases, etc..). A simple way to transform these tables into something that can be analysed would be to select one table as main table for learning and try to incorporate the content of other tables into (new) summary attributes. But the problem is that whatever transformation we choose, we loose information. For instance, assume that the new table is called purchase1, we could add the customer's information to it. But the table would be used to learn now information about purchases instead of customers. Alternatively, we could create a new table called "customer1", aggregate the row in the purchase table and add two new (aggregate) attributes (TotalValue and NofPurchases) to this new table. In this case we would loose information about the individual purchases and the learned outcome will not be as refined as the initial collection of attributes. What we would like is to get instead a general rule of the form described in the box above, which identifies relevant relationships over the entire set of attributes.

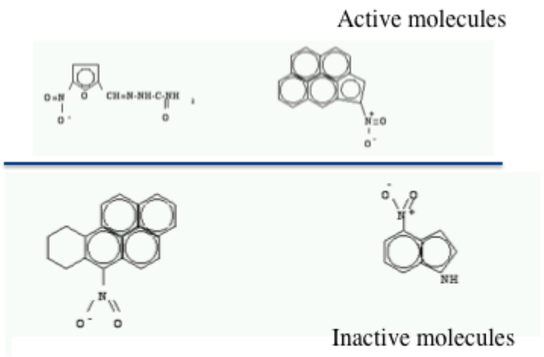


Course: C304 Logic-Based Learning

Example1: Mutagenicity prediction

Some molecules (e.g. mutagenic) can cause mutation in DNA, others don't. Mutations can lead to cancer.

How to predict which molecules are active mutagenic?
Can we use just information about their molecular structure?



Compound	Attr1	Attr2	Attr3	Class
ID1	true	false	true	Active
ID2	true	true	true	Active
ID3	false	false	true	Inactive
……	……	……	……	……

© Alessandra Russo

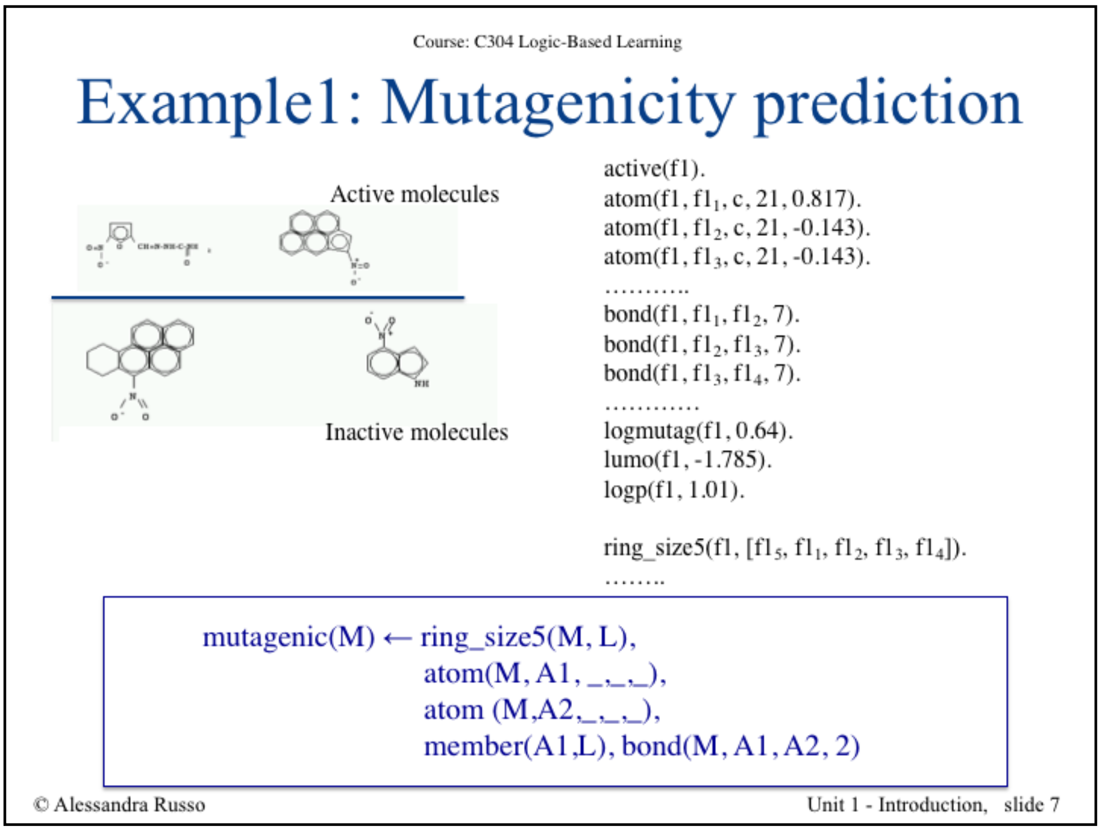
Unit 1 - Introduction, slide 6

To illustrate the benefits of logic-based learning, we briefly present some of its applications to real world problems. One of these applications is mutagenicity (Srinivasan & King,1999). Mutagenicity is the ability of chemical agents to cause changes in the genetic material (e.g. DNA) of an organism so triggering an increase in the frequency of genetic mutations. Increased genetic mutation can cause cancer. So mutagens are among carcinogen agents. The problem is to be able to predict which molecules are mutagenic and which are not, based on their structure properties.

We can consider examples of molecules, grouped into molecules that are active mutagenic and molecules that are not (i.e. labelled respectively positive and negative examples). The Machine Learning task would aim, in this case, to find a structure pattern, called **structure alert**, that discriminates the active from the non active mutagenic molecules. Traditional Machine Learning methods would need to first identify key (structural) features (or attributes) and represent the data in a single table where each compound example is encoded by a set of attribute-value (see table above in the slide). An example attribute is whether a "benzene ring" is present or not in the molecule. Finding the key relevant attributes is a non trivial and critical task, as the "goodness' of the prediction depends on the choice of the attributes. Changing the attributes clearly give different learning outcomes and predictions.

By using logic-based learning we try, instead, to address the key question of whether there is a principled way for representing molecule structures (instead of just pick and choose features) that would allow a general purpose machine learning system to reason directly about the molecular structure and identify structure-based properties that characterise the two given classes of examples, and at the same time be able to infer for any unseen molecule whether it is active mutagenic or not.

Logic allows for a reach representations of a given problem domain, in this case it would allow for a symbolic representation of the molecule's structure, for which general-purpose logic-based learning algorithms can be used to solve this learning/classification task. We show in the next slide an example representation.



Course: C304 Logic-Based Learning

Example1: Mutagenicity prediction

Active molecules

Inactive molecules

active(f1).
atom(f1, $f1_1$, c, 21, 0.817).
atom(f1, $f1_2$, c, 21, -0.143).
atom(f1, $f1_3$, c, 21, -0.143).
………..
bond(f1, $f1_1$, $f1_2$, 7).
bond(f1, $f1_2$, $f1_3$, 7).
bond(f1, $f1_3$, $f1_4$, 7).
…………
logmutag(f1, 0.64).
lumo(f1, -1.785).
logp(f1, 1.01).

ring_size5(f1, [$f1_5$, $f1_1$, $f1_2$, $f1_3$, $f1_4$]).
……..

mutagenic(M) ← ring_size5(M, L),
atom(M, A1, _,_,_),
atom (M,A2,_,_,_),
member(A1,L), bond(M, A1, A2, 2)

© Alessandra Russo Unit 1 - Introduction, slide 7

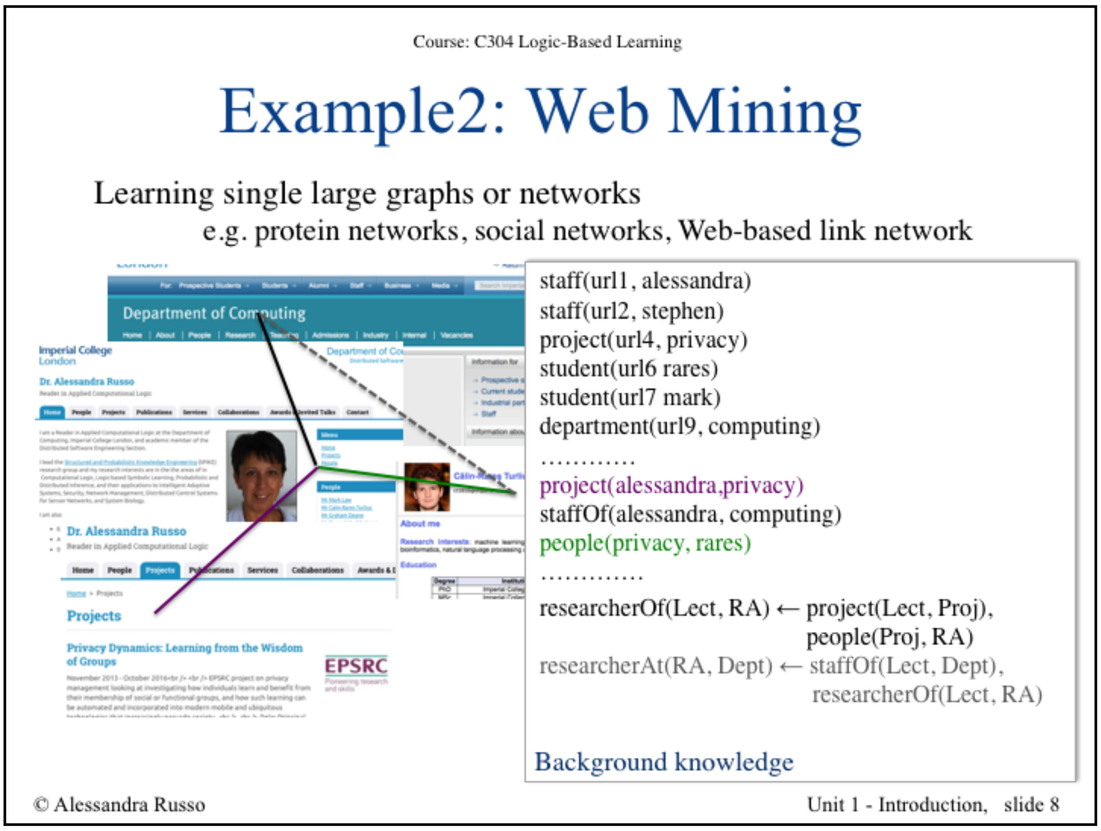
The molecular structure of the given compounds can be expressed by means of relations. Each compound is given a name (e.g. f1) and its atoms (e.g. $f1_1$, $f1_2$, $f1_3$, $f1_4$,…). The relation "atom" specifies 5 main structure properties of an atom: the molecule they are in (e.g. f1), the element name (e.g. c denotes carbon), the type (e.g. 21) and the charge (e.g. 0.817). The relation "bond" captures instead the binding between atoms that appear in a molecule, and the type of the binding. The latter 3 relations ("logmutag", "lumo" and "logp") express additional property values of a molecule: e.g. the lumo value (energy of the lowest unoccupied molecular orbital) and logp (hydrophobicity value of the molecule). This is a natural and declarative encoding of structure of molecules in terms of atoms and bond connectivity which does not depend on the specific feature selections.

In addition, it is also possible to represent additional properties of the molecular structure, as for instance the fact that the structure of molecule f1 has a ring of size 5, which involves atoms $f1_1$, $f1_2$, $f1_3$, $f1_4$, $f1_5$. This is expressed by the fact ring_size5(f1, [$f1_5$, $f1_1$, $f1_2$, $f1_3$, $f1_4$]).

Using the above representation and one of the earliest logic-based learning system (Progol), it has been possible to compute general structure-based property that can correctly predict when a molecule is an active mutagenic. Example of such a learned principle is the rule in the box above, which states that

"*A molecule is active mutagenic if its structure contains a ring of size 5, and atoms A1 and A2 that are connected by a double bond and such that A1 and A2 belong to the ring*".

Comparisons published in various papers have shown that a logic-based learning algorithm (e.g. Progol) using the above structured representation can outperform a feature-based learning algorithm.


Course: C304 Logic-Based Learning

Example2: Web Mining

Learning single large graphs or networks
e.g. protein networks, social networks, Web-based link network

Department of Computing
Imperial College London
Dr. Alessandra Russo
Dr. Alessandra Russo
Projects
Privacy Dynamics: Learning from the Wisdom of Groups
EPSRC

staff(url1, alessandra)
staff(url2, stephen)
project(url4, privacy)
student(url6 rares)
student(url7 mark)
department(url9, computing)
…………
project(alessandra,privacy)
staffOf(alessandra, computing)
people(privacy, rares)
………….

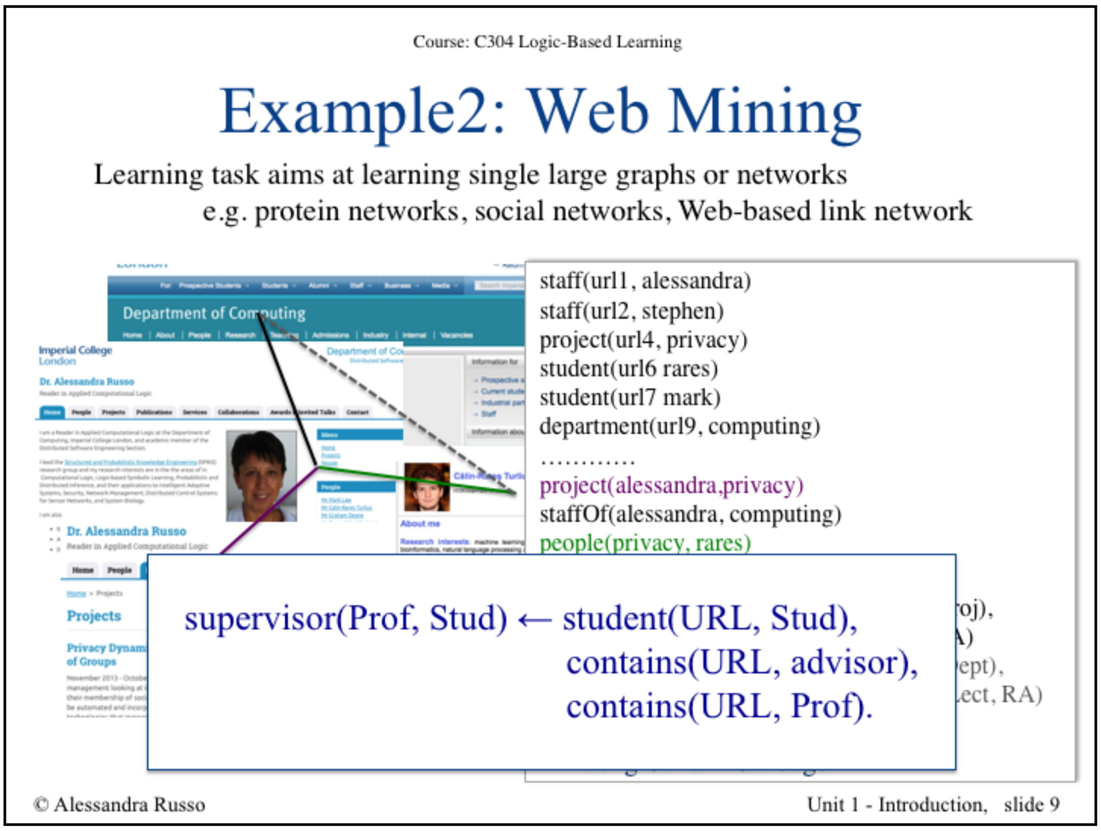
researcherOf(Lect, RA) ← project(Lect, Proj),
                                          people(Proj, RA)
researcherAt(RA, Dept) ← staffOf(Lect, Dept),
                                          researcherOf(Lect, RA)

Background knowledge

© Alessandra Russo Unit 1 - Introduction, slide 8

Among example application is that of **link mining**. Link mining aims at extracting large single graphs from collection of data. Examples include extraction of protein networks from protein-based data, social networks from twitter messages, web-based networks from web-links data. Let's consider here the latter case. In a typical university Web-site, we find web pages of academics, students, PhD students, project, taught courses, etc.. The set of data is therefore about a variety of entities (e.g. different websites), and a varieties of classes (e.g. students, staff, faculty, departments, etc.). We can assume that there is for each Webpage a "binary" fact defining the type of that webpage: for instance **staff(url1, alessandra)** for my own webpage, **staff(url2, stephen)** for Stephen's Webpage, etc.. We can also express initial relationships between webpages, which express semantic information of the web links. For instance, we can have **supervisor(alessandra, rares)**, **project(alessandra, privacy)**, **belongsTo(alessandra, computing)**. We can in principle decide how much refined knowledge we want to represent in our logic-based model for our learning problem.

Additional knowledge-based information can be expressed in the form of rules. For instance, we might want to express the knowledge that an "RA is a researcherOf a Lect if RA works on a Proj of Lect" (see first rule above), or that an "RA is a researcherAt a Dept if RA is a researcherOf Lect who is staffOf that Dept".



Course: C304 Logic-Based Learning

Example2: Web Mining

Learning task aims at learning single large graphs or networks
e.g. protein networks, social networks, Web-based link network

staff(url1, alessandra)
staff(url2, stephen)
project(url4, privacy)
student(url6 rares)
student(url7 mark)
department(url9, computing)
…………
project(alessandra,privacy)
staffOf(alessandra, computing)
people(privacy, rares)

oj),
A)
ept),
ect, RA)

supervisor(Prof, Stud) ← student(URL, Stud),
contains(URL, advisor),
contains(URL, Prof).

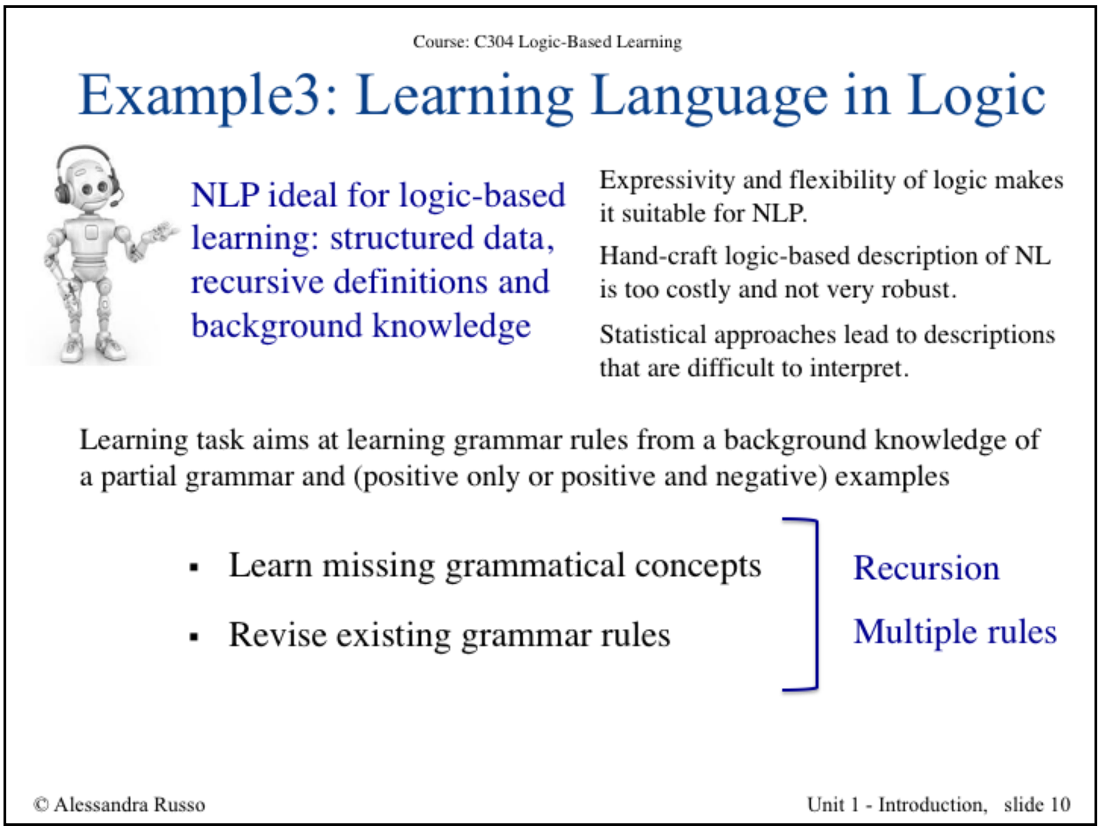
© Alessandra Russo Unit 1 - Introduction, slide 9

Suppose now that, given the Web hyperlinks of a particular university, we want to learn general rules that would allow us to predict the labels of unseen hyperlinks and extract semantic knowledge hidden in Web data. We could for instance learn a rule:

“supervisor(Prof, Stud) ← student(URL, Stud),
contains(URL, advisor), contains(URL, Prof)”

One of the advantages of logic-based learning is then the combination of logic and relational learning. As we will see in the next few lectures one of the key features in logic-based learning is the possibility of representing declaratively (semantic) knowledge and information about the problem domain and the data from which we want to learn.

Note that, as also stated in some previous slides, representing all this information in a single table would cause inevitably loss of information.


Course: C304 Logic-Based Learning

Example3: Learning Language in Logic

NLP ideal for logic-based learning: structured data, recursive definitions and background knowledge

Expressivity and flexibility of logic makes it suitable for NLP.

Hand-craft logic-based description of NL is too costly and not very robust.

Statistical approaches lead to descriptions that are difficult to interpret.

Learning task aims at learning grammar rules from a background knowledge of a partial grammar and (positive only or positive and negative) examples

- Learn missing grammatical concepts
- Revise existing grammar rules

Recursion

Multiple rules

© Alessandra Russo

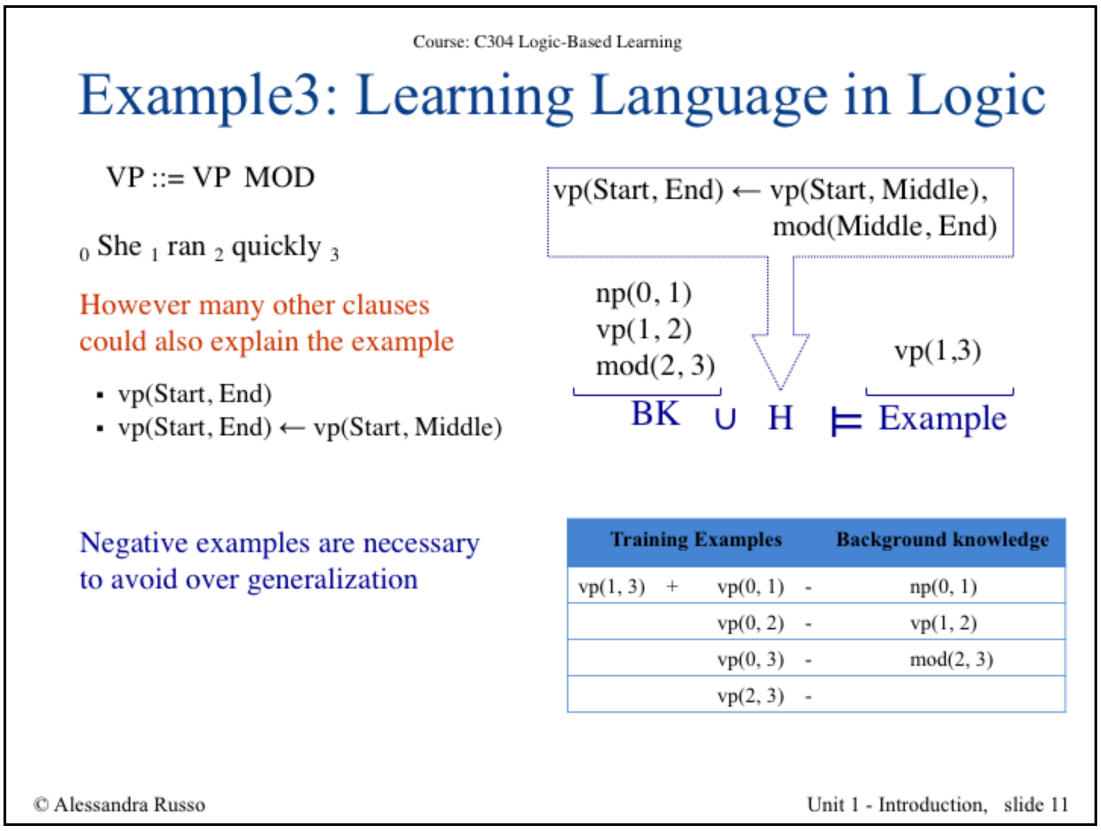
Unit 1 - Introduction, slide 10

As shown in the previous slide, problem domains for which logic-based learning appears to be particularly appropriate are those domains and learning tasks that require dealing with not only structured data but also (background) knowledge. Another example is **language learning** or learning natural language processing. An example of this learning task is learning rules for parsing an expressions, written in a given language, into another language, for instance parsing natural language into database queries. This form of logic-based learning, called *induction of semantic parser*, is at the core of query-answer systems. An example is for instance the use of automated answer phone systems that have to translate sentences spoken in natural language into database queries. Can such parsers be learned? Learning language grammar would help improve the robustness of rule-based systems which are conventionally used for this class of problems, as they will not need to be fully pre-defined for all possible grammatical constructs. The expressivity and flexibility of logic-based representation used in a logic-based learning approach makes it appropriate for reasoning about NLP problems. But manually developing a logic-based description of a natural language has been proved to be a difficult task, and rule-based NLP, which requires a predefined rule-based representation of the grammar, is seen as a too costly and not very robust approach for commercial applications.

Statistical approaches for natural language processes partly help in addressing this problem, but they tend to provide a superficial, syntactic-based analysis of the text. Logic-based learning is an ideal alternative that balance richness of language used in the learning with deeper semantic understanding.

Grammar rules can be easily represented as clauses, which are often recursive. This is also one of the reason why Inductive Logic Programming (for learning logic programs) are particularly appropriate (recursion is easy to represent in Prolog). Complex terms (e.g. lists) are also appropriate for expressing structured data (e.g. representing sentences as nested lists of words).

Moreover the task of Learning Language needs to deal with simultaneous completion and/or revision of existing logic-based partial representation (often with recursive definition) of a grammar.



Course: C304 Logic-Based Learning

Example3: Learning Language in Logic

VP ::= VP MOD

$_0$ She $_1$ ran $_2$ quickly $_3$

However many other clauses could also explain the example

- vp(Start, End)
- vp(Start, End) ← vp(Start, Middle)

$$\underbrace{\begin{matrix}np(0,1)\\ vp(1,2)\\ mod(2,3)\end{matrix}}_{BK} \cup \overset{vp(Start, End) \leftarrow vp(Start, Middle), mod(Middle, End)}{H} \models \underbrace{vp(1,3)}_{Example}$$

Negative examples are necessary to avoid over generalization

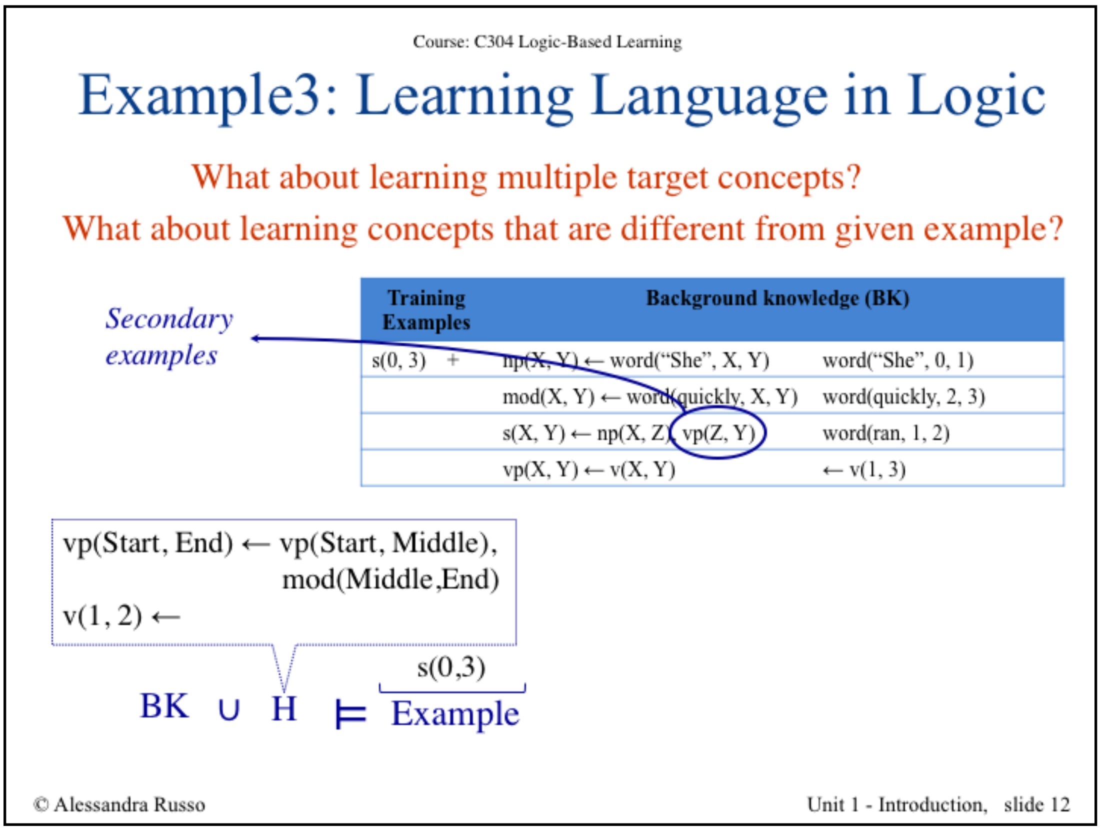
Training Examples				Background knowledge
vp(1, 3)	+	vp(0, 1)	-	np(0, 1)
		vp(0, 2)	-	vp(1, 2)
		vp(0, 3)	-	mod(2, 3)
		vp(2, 3)	-	

© Alessandra Russo   Unit 1 - Introduction, slide 11

The grammar rule on the top right part of the slide gives an example of an English grammar rule where VP means "verbal phrase" and MOD means "modifier". The rule states that "a verbal phrase is a verbal phrase followed by a modifier". For example "ran quickly" is a verbal phrase since "ran" is a verbal phrase and "quickly" is a modifier. Natural language parsing can be done by assuming delimiters between words and considering facts about words. In the sentence above the index 0, 1, 2 and 3 are the delimiters and atoms vp(1, 2) and mod(2, 3) are facts (basic background knowledge about the language) that state that "ran" is a verbal phrase, "quickly" is a modifier, etc. So given the example sentence "ran quickly" and the above three facts in the background knowledge, it is possible to learn the general rule vp(Start,End) ← vp(Start, Middle), mod(Middle, End), which would cover the example vp(1,3), namely that "ran quickly" is a verbal phrase.

However, many other clauses would be acceptable solutions in our given learning task. For instance, the clause vp(Start, End) would cover the given example vp(1, 3) but it would also wrongly cover vp(0, 3) or vp(0, 1), which are clearly not verbal phrases. It is often the case that over generalization (which is what just clause vp(Start, End) would be) is computed when not enough negative examples are given. We could consider in this case to assume that every other instance of verbal phrase covered in the given sentence and not included in the positive examples, should be considered as a negative example. This is what is shown in the above table. In this case the two over generalizations would not be accepted as solutions as they would cover some of the negative examples.

In the above slide, the table shows a complete set of negative examples for a single given positive example, and only a single clause is learned. In reality a language learning problem does not necessarily requires complete set of negative examples and may need more than one single clause to be learned simultaneously. Language learning is not primarily a classification problem (i.e. given positive and negative example, how to differentiate correct sentences from non grammatically correct sentences), but rather a **parsing problem**: given some sentences, what parser rules should be used to correctly parse the sentences. Over generalization would lead to multiple parsing rules and the underlying learning problem is how to disambiguate between parsers.



Course: C304 Logic-Based Learning

Example3: Learning Language in Logic

What about learning multiple target concepts?

What about learning concepts that are different from given example?

Secondary examples

Training Examples	Background knowledge (BK)	
s(0, 3) +	np(X, Y) ← word("She", X, Y)	word("She", 0, 1)
	mod(X, Y) ← word(quickly, X, Y)	word(quickly, 2, 3)
	s(X, Y) ← np(X, Z), vp(Z, Y)	word(ran, 1, 2)
	vp(X, Y) ← v(X, Y)	← v(1, 3)

vp(Start, End) ← vp(Start, Middle),
mod(Middle,End)

v(1, 2) ←

$$BK \cup H \models \overbrace{\text{Example}}^{s(0,3)}$$

© Alessandra Russo

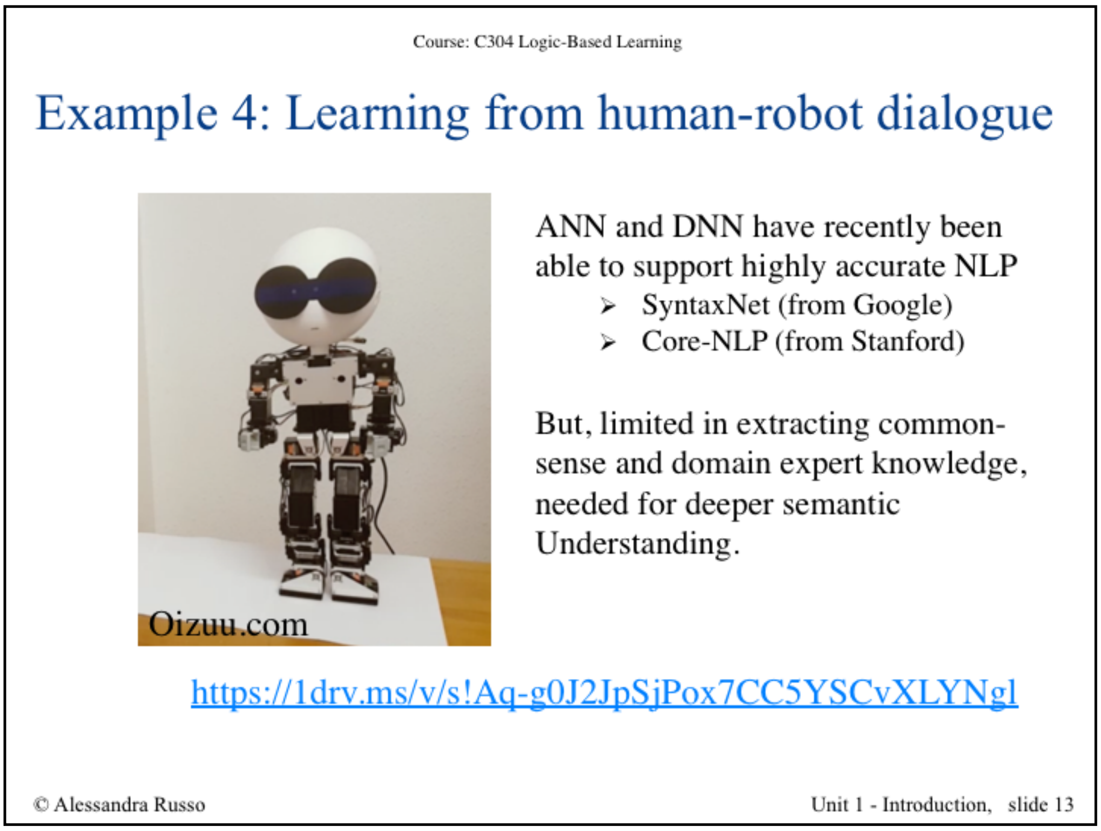
Unit 1 - Introduction, slide 12

In the previous example, we have considered the task of learning a single concept (i.e. a verbal phrase). But logic-based learning does not need to be limited to just learning one concept at the time. As we will see during this course, many different logic-based learning algorithms have been developed, some limited to just learning single clause hypothesis (H), others more general and capable of learning multi clause hypothesis (i.e. multiple concepts) from the same set of examples. Related to this more general property is the fact that concepts to be learned do not need to be the same as the concepts observed (or given as examples). In the previous slide, examples are about the predicate that is learned.

Consider, for instance, the table given in the above slide (taken from a book entitled "Learning Language in Logic – LNAI, 2000). The example is the same sentence considered in the previous slide ("She ran quickly"). The background knowledge is slightly larger and includes already a notion of verbal phrase which essentially corresponds to the grammar rule that a verb is a verbal phrase. But in this case we don't know that "ran" is a verb. So, on a very intuitive level to prove the example s(0, 3), the rules given in the background knowledge could be used but provided that we could prove the concept vp(1, 3). We could use the rule about vp( ) already given in the background knowledge but this would require v(1, 3) to be true, which is not the case.
Note that ← v(1, 3) means that "ran quickly" is not a verb! We call this type of expression *integrity constraints*.

So a new rule for vp would have to be learned. In this case a similar rule as the one shown in the previous slide could be learned (i.e. vp(X, Y) ← vp(X, Z), mod(Z, Y)), provided that vp(1, 2) could be proved. Again this could be possible if we were learning also the fact v(1,2), namely that "ran" is a verb. The output hypothesis would in this case be two clauses, of which the second is an atomic clause.

In this example, we have seen that it is also possible to conceive cases where more that one clause needs to be learned to cover the examples and also that the learning can take into account of integrity constraints, as part of the background knowledge. Not all existing logic-based learning systems are capable of learning in the presence of integrity constraints and/or learning multiple clauses. In slide 22, we'll give a brief history of the learning approaches and algorithms proposed in the last 20 years.



Example 4: Learning from human-robot dialogue

ANN and DNN have recently been able to support highly accurate NLP

- SyntaxNet (from Google)
- Core-NLP (from Stanford)

But, limited in extracting common-sense and domain expert knowledge, needed for deeper semantic Understanding.

https://1drv.ms/v/s!Aq-g0J2JpSjPox7CC5YSCvXLYNgl

This is an example of how logic-based learning could be used to learn common sense knowledge through human-machine dialogue. It's one of the first examples of applying machine learning algorithms, such as Neural Network, to process audio signals and extract logic-based representation of dialogue between human and machine, and then use question and answer for the system to learn common sense knowledge when wrong answers are provided in order to continuously refine its knowledge.

This work was done as student project by an MSc student in summer 2016.

Example5: Machine Comprehension of Text

- End-to-end semantic representation of English text into interpretable modes.
- Learning common-sense knowledge from Q&A

Text comprehension + Given correct and incorrect answers

Story:

1. John went to the local restaurant.
2. The waiter brought John a glass of water and took the order.
3. As John was waiting, he took out the book and began to read it.
4. The steak which he ordered finally arrived.
5. After John had finished the meal, he took the jacket but he forgot to take the book.
6. He paid the bill and went back to the hotel.

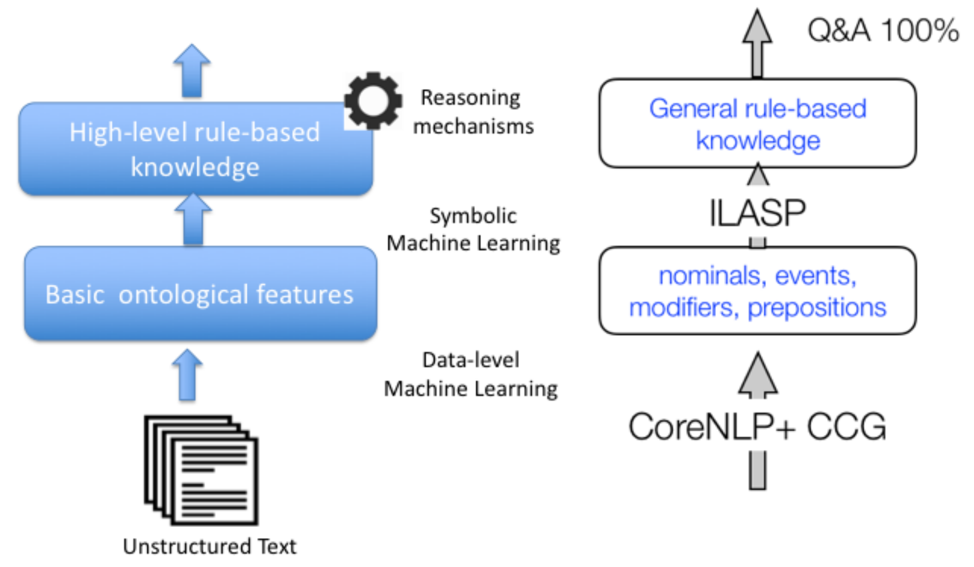
Questions:

1. Where is John?
2. Where was John before he went to the hotel?
3. Who took the order?
4. Who received a glass of water?
5. Did John have to wait?
6. What did John read?
7. What did John choose?
8. Where is the book?
9. Where is the jacket?

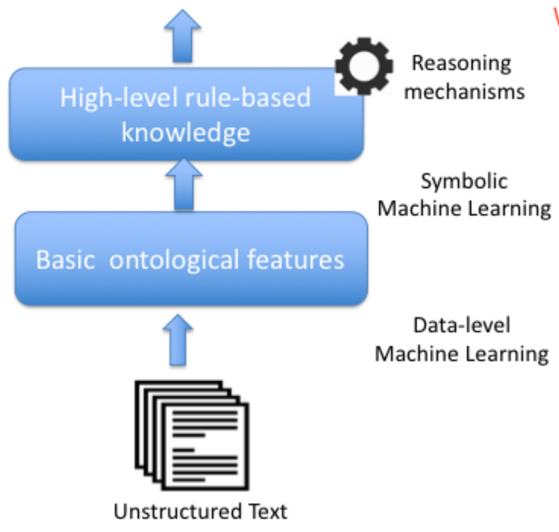
Learned Knowledge

- If a person goes somewhere then he/she will be at that location.
- If a person picks an object than he/she will carry that object
- If a person carries an object and he/she goes to a location then the object will be at that location

Example5: Machine Comprehension of Text


Q&A 100%
High-level rule-based knowledge
Reasoning mechanisms
General rule-based knowledge
ILASP
Symbolic Machine Learning
Basic ontological features
nominals, events, modifiers, prepositions
Data-level Machine Learning
CoreNLP+ CCG
Unstructured Text

Example5: Machine Comprehension of Text



High-level rule-based knowledge
Reasoning mechanisms
Symbolic Machine Learning
Basic ontological features
Data-level Machine Learning
Unstructured Text

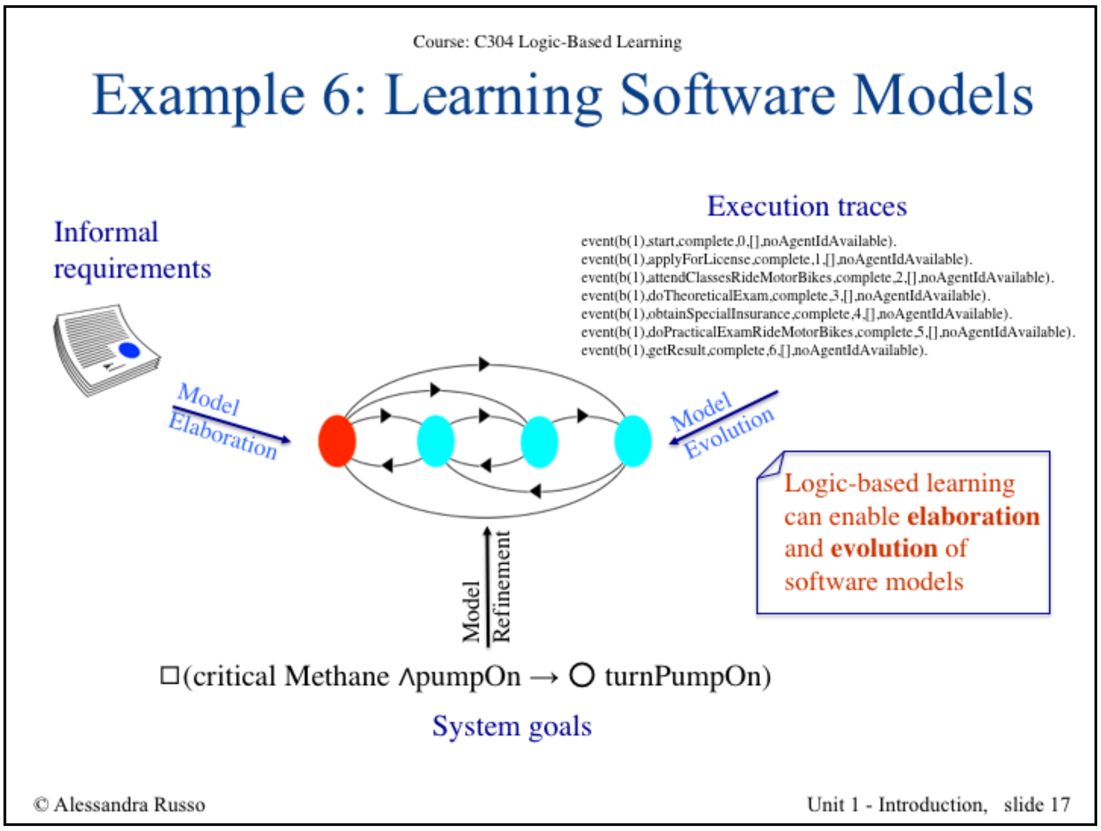
Where is Jack? At the restaurant

Q&A 100%

```
binaryEvent(E,be,P,L):-
   binaryEvent(E,go,P,L).
```

```
binaryEvent(e0,go,P0,L1) :-
   nominal(P0,Jack),
   nominal(L1,restaurant).
binaryEvent(e0,go,f0,f1).
nominal(f0, Jack).
nominal(f1, restaurant).
```

Jack went to the local restaurant



Course: C304 Logic-Based Learning

Example 6: Learning Software Models

Informal requirements

Execution traces

```
event(b(1),start,complete,0,[],noAgentIdAvailable).
event(b(1),applyForLicense,complete,1,[],noAgentIdAvailable).
event(b(1),attendClassesRideMotorBikes,complete,2,[],noAgentIdAvailable).
event(b(1),doTheoreticalExam,complete,3,[],noAgentIdAvailable).
event(b(1),obtainSpecialInsurance,complete,4,[],noAgentIdAvailable).
event(b(1),doPracticalExamRideMotorBikes,complete,5,[],noAgentIdAvailable).
event(b(1),getResult,complete,6,[],noAgentIdAvailable).
```

Logic-based learning can enable **elaboration** and **evolution** of software models

$\Box$(critical Methane $\wedge$pumpOn $\rightarrow$ $\bigcirc$ turnPumpOn)

System goals

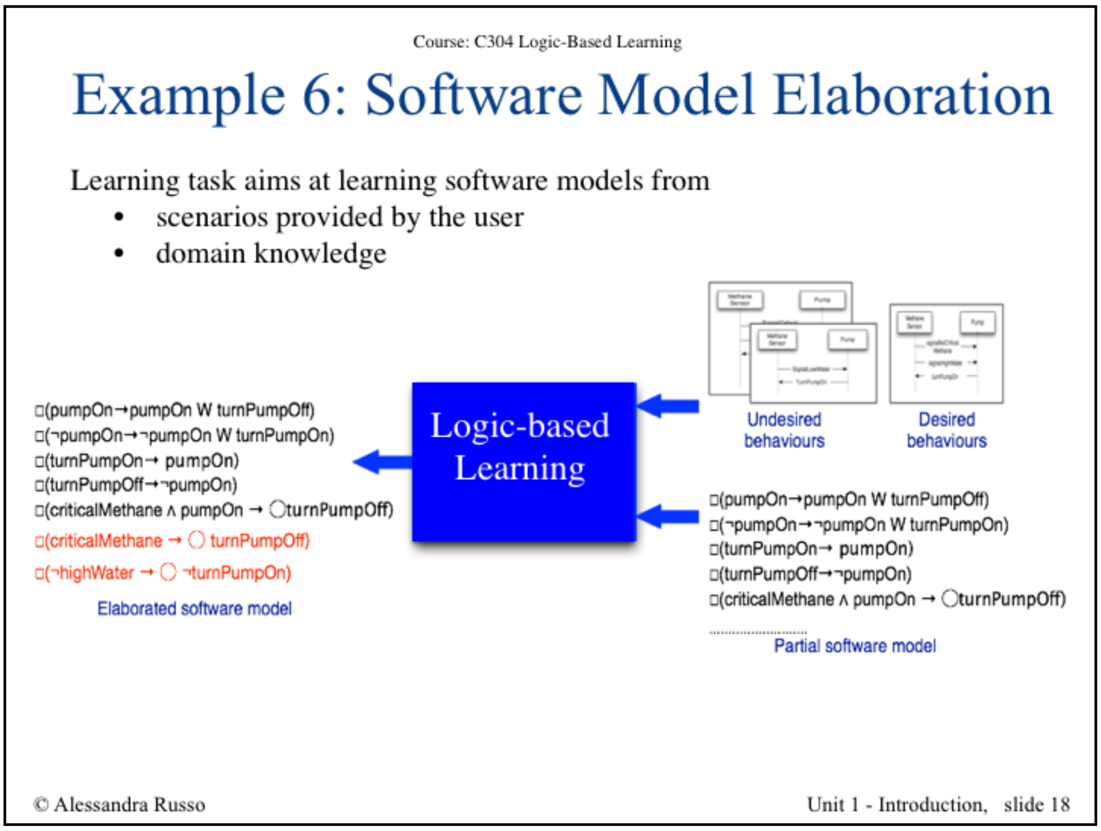
© Alessandra Russo   Unit 1 - Introduction,   slide 17

Recent developments in the area of logic-based learning have also enabled new challenging applications in other areas of computing. These include software engineering and ubiquitous systems. This slide presents three key activities in model-driven software engineering: *elaboration, refinement and evolution* of formal models (e.g. labeled transition systems) that describe the behavior of a system to be. *Elaboration* requires the “automated” development of formal models from informal requirements of a system (often expressed as scenarios, message sequence charts) and domain knowledge. The task in this case is to learn from a collection of scenarios (or desired traces) a model that covers all given desired scenarios, rejects all the undesired behaviours, whilst preserving some general system properties (e.g. system goals). Learning in this case is, as in the previous application, not a problem of classification of good scenarios from bad scenarios, but a process of model generation, in the same way as in the NLP the objective is to generate the model of a parser.

The second learning task is the *refinement* of an existing (incomplete or incorrect) system model. Model checking has shown to be a successful mechanism for verifying the correctness of a formal model of a system with respect to system properties. However when example traces are computed, by the model checker, that violate at least one of the property (i.e. a counterexample is computed) the task of understanding how to correct the model so to eliminate the counterexample without adding new (unwanted) error is often a very difficult manual task. Particular forms of logic-based machine learning can be integrated with model checking in order to learn from counterexamples the appropriate refinements/revisions of the given formal model so to eliminate the counterexample(s) and preserve desired positive behaviors.

Finally, as a third application, logic-based learning could also be used to support reverse engineering tasks, namely extraction/evolution of formal models from execution traces of implemented systems.

Key features of recent logic-based learning that have made all these applications possible are the ability to *learn in the presence of partial information and constraints (e.g. system properties or goals), to reason about default assumptions and ability to learn exceptions and multiple clauses.*



Course: C304 Logic-Based Learning

Example 6: Software Model Elaboration

Learning task aims at learning software models from

- scenarios provided by the user
- domain knowledge

Undesired behaviours

Desired behaviours

$\square(\text{pumpOn} \rightarrow \text{pumpOn W turnPumpOff})$
$\square(\neg\text{pumpOn} \rightarrow \neg\text{pumpOn W turnPumpOn})$
$\square(\text{turnPumpOn} \rightarrow \text{pumpOn})$
$\square(\text{turnPumpOff} \rightarrow \neg\text{pumpOn})$
$\square(\text{criticalMethane} \wedge \text{pumpOn} \rightarrow \bigcirc\text{turnPumpOff})$
....

Partial software model

Logic-based Learning

$\square(\text{pumpOn} \rightarrow \text{pumpOn W turnPumpOff})$
$\square(\neg\text{pumpOn} \rightarrow \neg\text{pumpOn W turnPumpOn})$
$\square(\text{turnPumpOn} \rightarrow \text{pumpOn})$
$\square(\text{turnPumpOff} \rightarrow \neg\text{pumpOn})$
$\square(\text{criticalMethane} \wedge \text{pumpOn} \rightarrow \bigcirc\text{turnPumpOff})$
$\square(\text{criticalMethane} \rightarrow \bigcirc\ \text{turnPumpOff})$
$\square(\neg\text{highWater} \rightarrow \bigcirc\ \neg\text{turnPumpOn})$

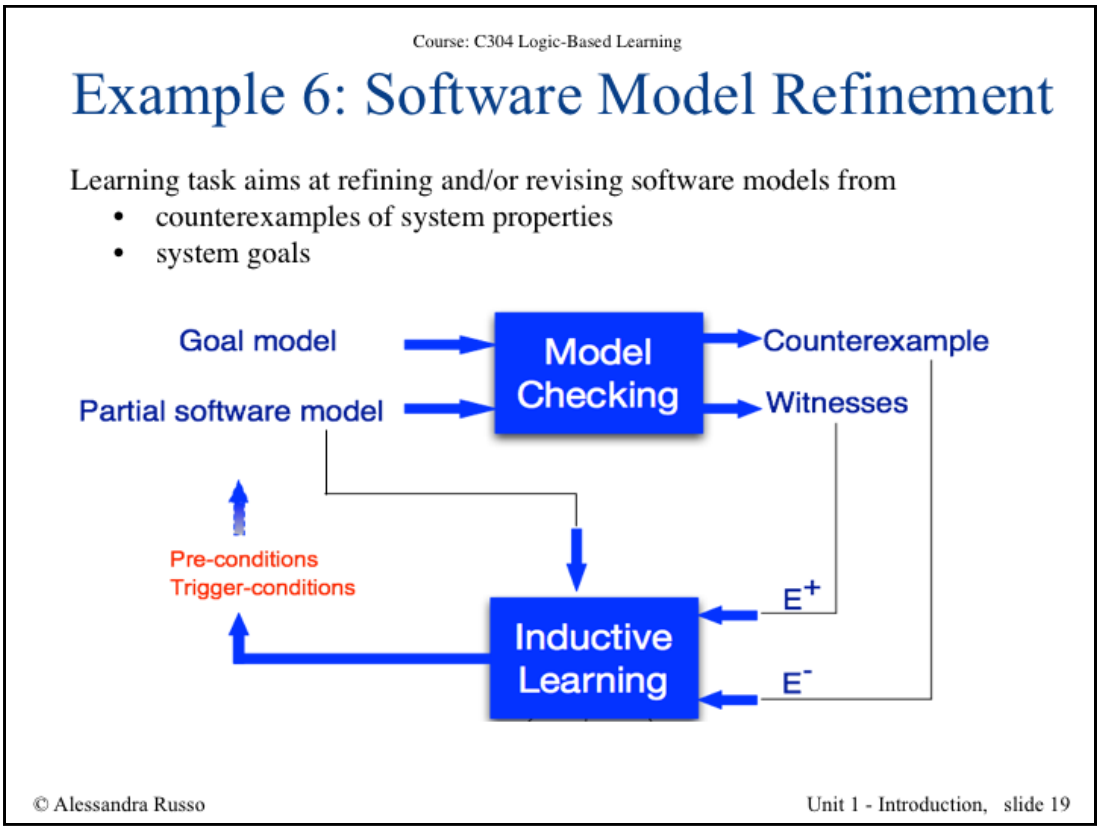
Elaborated software model

© Alessandra Russo Unit 1 - Introduction, slide 18

This slide gives an example of using logic-based machine learning for model elaboration. In this application domain, the learning task is to learn from a given set of scenarios that express desired and undesired partial traces of system behaviours, a model that covers the desired behavior and rejects the undesired ones. The input to the logic-based learning system includes two parts: i) a set of message sequence charts. These are (partial) temporal sequence of events that the system should perform or should not perform. The second component ii) is a domain knowledge that can be expressed in any formal language. In this slide, it is expressed using temporal logic. The problem domain is a software system that controls a mine pump. Example of domain knowledge are effects of events, such as “turning the pump off will make the pump be off”, and general principle of natural inertia: “if the pump is off, it will continue to stay off until the pump is turned on”.

The learning task can compute rules, for instance, about pre-conditions of events. These should also be expressed in a formal logic-based language. In our slide, the learned rules are themselves representable in temporal logic. They are learned so to guarantee that they are consistent with the given domain knowledge, the desired scenarios are going to be true and undesired scenarios false.

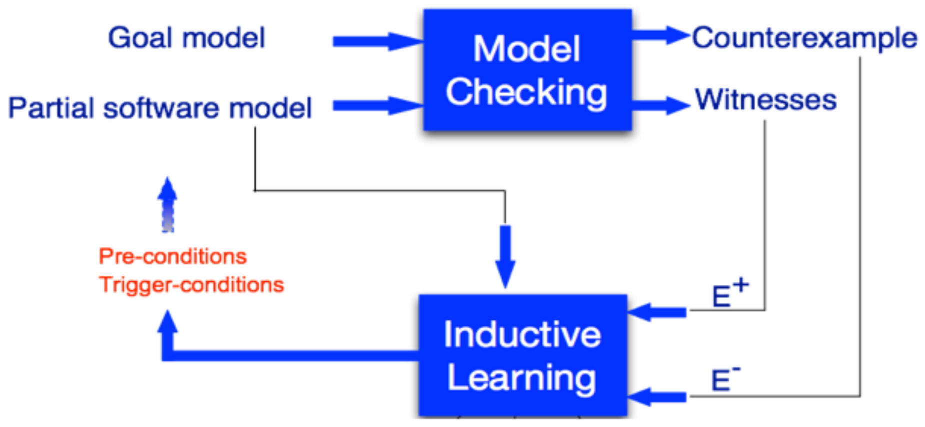
The learned rules can then be re-expressed in terms of a state transition system, or labeled transition system, thus providing a valuable solution for software engineering on how to automatically synthesize, (automatically generate) labeled transition systems from message sequence charts.


Course: C304 Logic-Based Learning

Example 6: Software Model Refinement

Learning task aims at refining and/or revising software models from

- counterexamples of system properties
- system goals



© Alessandra Russo

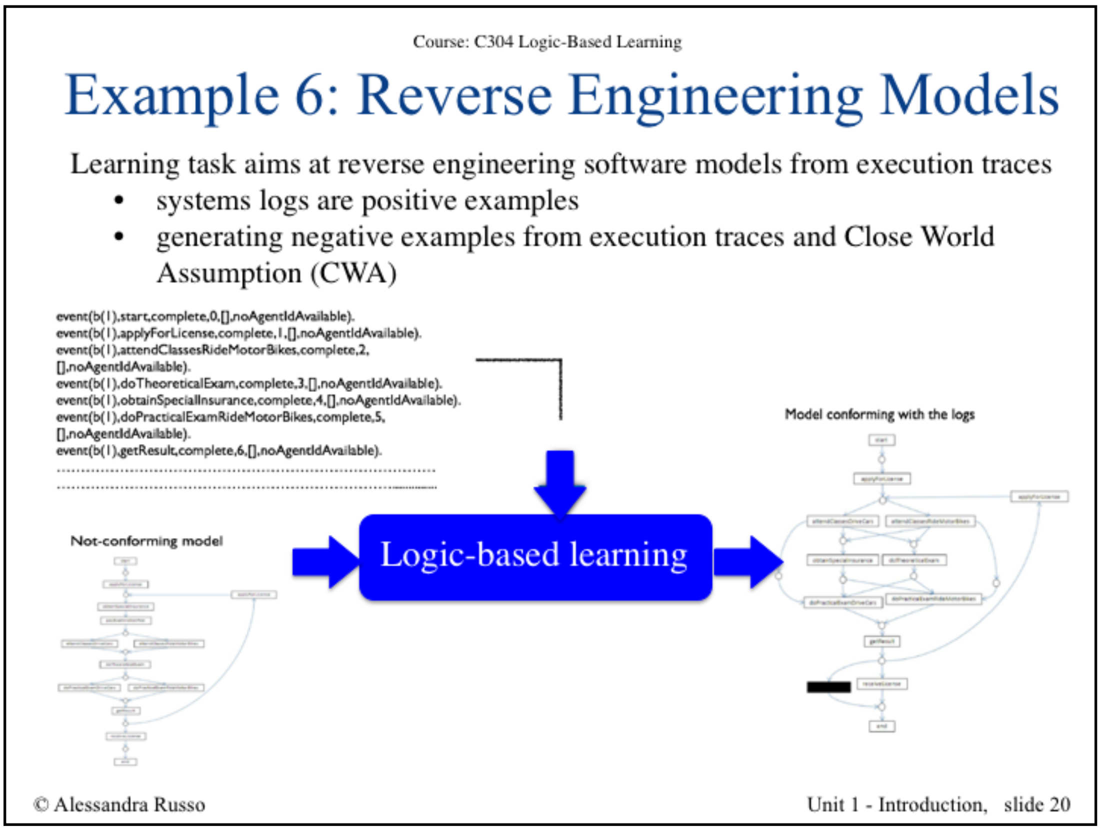
Unit 1 - Introduction, slide 19

This example gives an example of model refinement. In this second application the use of logic-based learning can be advantageous in many real-world domains, as it demonstrates that logic-based learning can be integrated with model checking in order to facilitate refinements of models that would no longer cover detected counterexamples.

The counterexamples are generated by the model checker when trying to verify the correctness of a (partial) software model with respect to a goal model (or set of system properties). Learning from just single negative examples would not help computing general refinements of the software model. Witnesses are examples of positive behaviours that can be generated from the software model and that satisfy the given goal model. Without witnesses, too specialised or over-general refinements of the model would be learned, causing the risk of rejecting also good behaviours. Witnesses are therefore needed in order to control over refinements.

The above diagram shows a framework for rigorous, tool-supported requirements elaboration. It provides a systematic, iterative approach for generating a set of operational requirements that are complete with respect to given system goals. The model checking formally verifies the satisfaction of the goals and produces counterexamples when incompleteness in the operational requirements is detected. The inductive learning process computes operational requirements from the counterexamples and user-provided, or automatically generated, positive examples. The learned operational requirements are guaranteed to eliminate the counterexamples, cover the witness examples and be consistent with the goals.

Note that to compute the full model, given that the model checker is not able to compute all possible counterexamples to a given set of system goals, the refinements requires an iterative process until no further goal violations are detected.



Example 6: Reverse Engineering Models

Learning task aims at reverse engineering software models from execution traces

- systems logs are positive examples
- generating negative examples from execution traces and Close World Assumption (CWA)

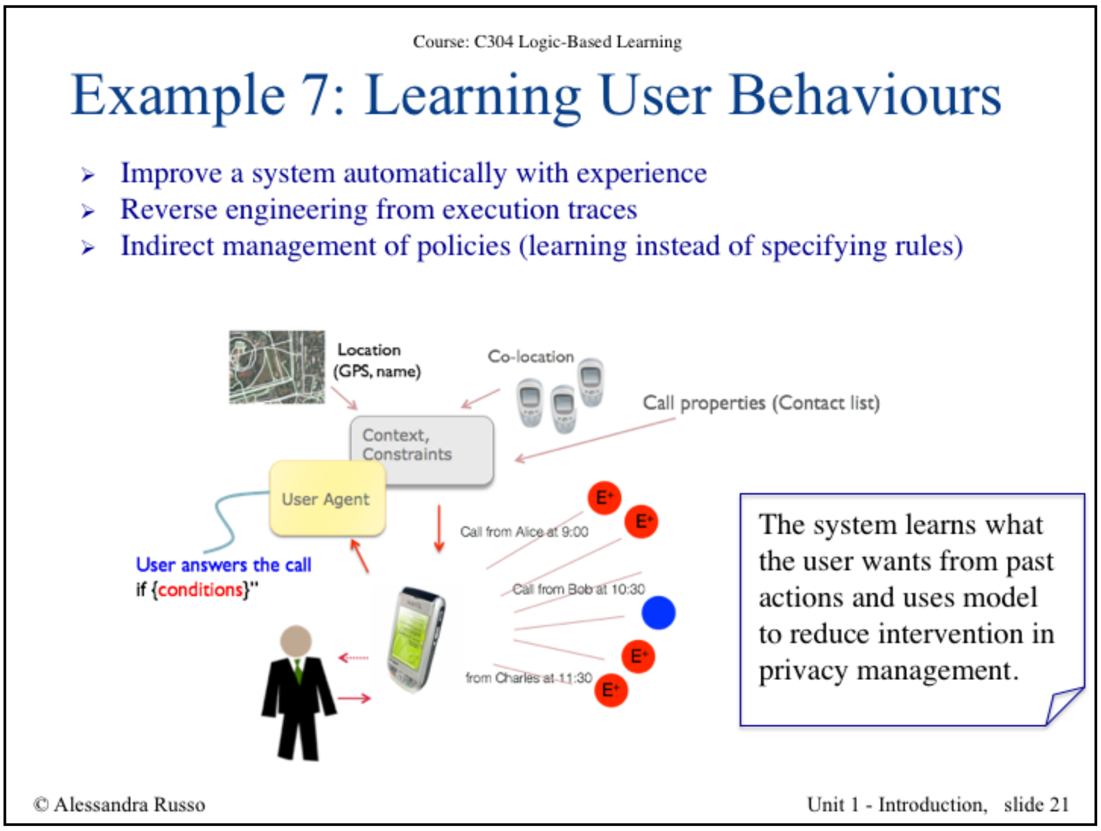
```
event(b(1),start,complete,0,[],noAgentIdAvailable).
event(b(1),applyForLicense,complete,1,[],noAgentIdAvailable).
event(b(1),attendClassesRideMotorBikes,complete,2,
[],noAgentIdAvailable).
event(b(1),doTheoreticalExam,complete,3,[],noAgentIdAvailable).
event(b(1),obtainSpecialInsurance,complete,4,[],noAgentIdAvailable).
event(b(1),doPracticalExamRideMotorBikes,complete,5,
[],noAgentIdAvailable).
event(b(1),getResult,complete,6,[],noAgentIdAvailable).
.....................................................................
.....................................................................
```

Another example application of logic-based learning is in the context of business processes and reverse engineering. Models are in this domain often expressed using BPEL languages or Petri Net. Often processes in companies and organisation evolve during time, through changes applied directly at implementation level. Good software engineering practice, on the other hand, requires software models and implementation to be in sync, to allow verification and analysis whenever needed (often exactly in the case when changes are applied to an existing model).

The reverse engineering task in this example consists in the revision of existing process models from logs of (changed/evolved) processes. This example is somewhat similar to the NLP application described above as the fundamental question asked here is the "computation of a revised model that recognizes the current logs. Furthermore, being the objective of the task to learn revised models, instead of learning a model from scratch, the new learned model has to be "as close as" possible to the existing one. Minimal changes in the existing model should be learned, for covering the given logs.

Negative examples could be generated through the same idea of "close world assumption". Going through the given ontology of transactions/activities included in a process, any sequence of activities that is not considered in the log could be in principle assumed to be a negative example. Of course this would require the logs to be as complete as possible and the processes to be deterministic. Recent work has shown that under this assumption logic-based learning has been effective in computing expected revised Petri Net models.

Similar applications have also been developed in the context of games and learning of Markov Decision Process models instead of Petri Net models. One of the advantage of using such general purpose logic-based machine learning systems is that different classes of models can be learned as long as these are expressible in logic terms. Given the expressive power of logic this means that quite a large class of different types of models could be learned.



Course: C304 Logic-Based Learning

Example 7: Learning User Behaviours

- Improve a system automatically with experience
- Reverse engineering from execution traces
- Indirect management of policies (learning instead of specifying rules)

Location (GPS, name)
Co-location
Call properties (Contact list)
Context, Constraints
User Agent
User answers the call
if {conditions}"
Call from Alice at 9:00
Call from Bob at 10:30
from Charles at 11:30
E+

The system learns what the user wants from past actions and uses model to reduce intervention in privacy management.

© Alessandra Russo Unit 1 - Introduction, slide 21

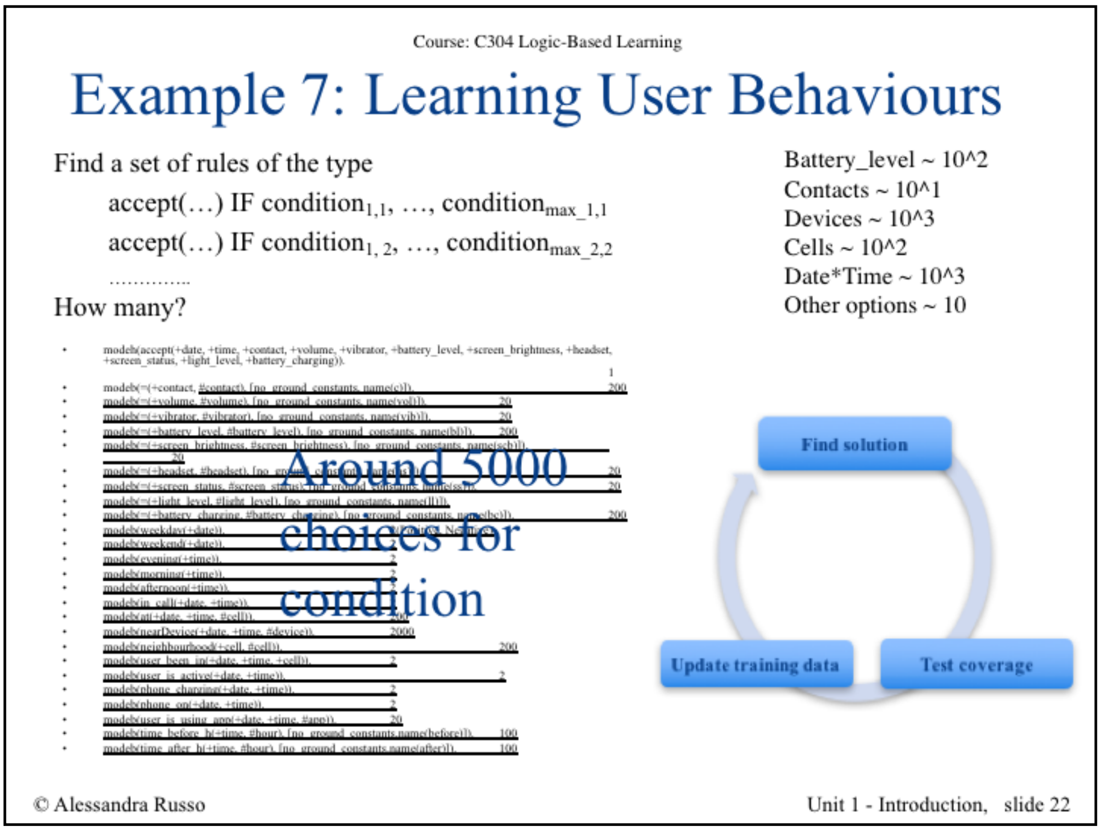
This example application is instead in the context of pervasive, ubiquitous computing. The objective is to learn user behaviours from past traces, sensed or recorded by the system and use these as examples to extract policy rules that can be automatically enforced into the system so to reduce the manual intervention of the user in setting up or changing existing policy managements. Here the type of policy management that we are interested is privacy management policy.

The problem demonstrated in this slide is how to learn privacy policies about user behavior in accepting or rejecting phone calls. Examples are therefore accepted calls and rejected calls of a given user. Key features are information that can be pervasively collected by the mobile device: for instance the caller's name, the location of the user, the time of the day, the current application running on the phone, etc.

The challenge is to learn rules that have a good level of accuracy in predicting as much as possible correct user's decision in accepting incoming calls, for a mobile device to intelligently adapt to the user behavior and reduce human intervention. Aspects that come into play is the potentially large quantity of data, level of noise in the data, e.g. when detecting co-located devices, etc.

But a clear advantage in using logic-based learning instead of any other machine learning approach is the possibility of automatically translate learned rules into structure English so to involve the user himself in selecting among possible alternative learned solutions and provide feedback loop to the learner itself. (This is shown in the next slides). In the context of ubiquitous computing when learning control programs that replace human's intervention, it is important to have the "human-in-the-loop" in order to guarantee the performance of the "autonomous" responses of the system to meet the human desires and expectations.

This is difficult to do when using numerical/statistical machine learning approaches as it is not always easy to extrapolate what has been learned into a form that is accessible to humans.



Course: C304 Logic-Based Learning

Example 7: Learning User Behaviours

Find a set of rules of the type

accept(…) IF condition$_{1,1}$, …, condition$_{max\_1,1}$

accept(…) IF condition$_{1,2}$, …, condition$_{max\_2,2}$

…………..

How many?

- modeh(accept(+date, +time, +contact, +volume, +vibrator, +battery_level, +screen_brightness, +headset, +screen_status, +light_level, +battery_charging)). 1
- modeb(=(+contact, #contact), [no_ground_constants, name(c)]). 200
- modeb(=(+volume, #volume), [no_ground_constants, name(vol)]). 20
- modeb(=(+vibrator, #vibrator), [no_ground_constants, name(vib)]). 20
- modeb(=(+battery_level, #battery_level), [no_ground_constants, name(bl)]). 200
- modeb(=(+screen_brightness, #screen_brightness), [no_ground_constants, name(sch)]). 20
- modeb(=(+headset, #headset), [no_ground_constants, name(hs)]). 20
- modeb(=(+screen_status, #screen_status), [no_ground_constants, name(ss)]). 20
- modeb(=(+light_level, #light_level), [no_ground_constants, name(ll)]).
- modeb(=(+battery_charging, #battery_charging), [no_ground_constants, name(bc)]). 200
- modeb(weekday(+date)). 2(Positive, Negative)
- modeb(weekend(+date)). 2
- modeb(evening(+time)). 2
- modeb(morning(+time)). 2
- modeb(afternoon(+time)). 2
- modeb(in_call(+date, +time)). 2
- modeb(at(+date, +time, #cell)). 200
- modeb(nearDevice(+date, +time, #device)). 2000
- modeb(neighbourhood(+cell, #cell)). 200
- modeb(user_been_in(+date, +time, +cell)). 2
- modeb(user_is_active(+date, +time)). 2
- modeb(phone_charging(+date, +time)). 2
- modeb(phone_on(+date, +time)). 2
- modeb(user_is_using_app(+date, +time, #app)). 20
- modeb(time_before_h(+time, #hour), [no_ground_constants,name(before)]). 100
- modeb(time_after_h(+time, #hour), [no_ground_constants,name(after)]). 100

Around 5000 choices for condition

Battery_level ~ 10^2
Contacts ~ 10^1
Devices ~ 10^3
Cells ~ 10^2
Date*Time ~ 10^3
Other options ~ 10

© Alessandra Russo

Unit 1 - Introduction, slide 22

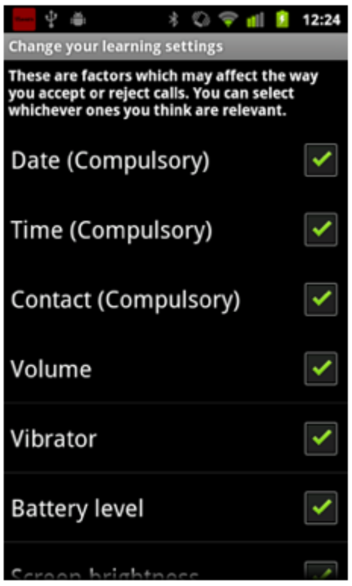
Rules to be learned in this example application are rules with one particular type of predicate "accept( )" and a large choice of possible conditions. These are conditions about sensed data, about the state of the phone and the called received.

There can be many choices of instances of conditions (e.g. 5000 choices).

Despite the large set of data the logic-based learning has been successfully used to compute user-behavior policies but exploiting underlying properties of the logical representation. *The problem in this case does not need recursion, no interdependent rules to be learned*. So an iterative learning process can be used in which one example at the time can be considered. This is because of the monotonicity of the problem: i.e. if an example is entailed by a current solution there is no point at considering it in the rest of the computation as it will be covered by the next solutions too. We will consider, later in the course, details on these properties (monotonicity and non-monotonicity) of a learning task.

Course: C304 Logic-Based Learning

Example 7: My mobile knows me!

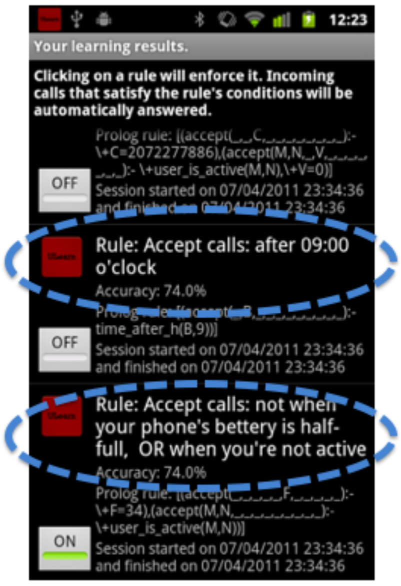


12:24

Change your learning settings

These are factors which may affect the way you accept or reject calls. You can select whichever ones you think are relevant.

- Date (Compulsory) ✓
- Time (Compulsory) ✓
- Contact (Compulsory) ✓
- Volume ✓
- Vibrator ✓
- Battery level ✓
- Screen brightness



12:23

Your learning results.

Clicking on a rule will enforce it. Incoming calls that satisfy the rule's conditions will be automatically answered.

OFF
```
Prolog rule: [(accept(_,_,C,_,_,_,_,_,_,_,_):-
\+C=2072277886),(accept(M,N,_,V,_,_,_,_,_,_,_):- \+user_is_active(M,N),\+V=0)]
Session started on 07/04/2011 23:34:36
and finished on 07/04/2011 23:34:36
```

Rule: Accept calls: after 09:00 o'clock

Accuracy: 74.0%

OFF
```
Prolog rule: [(accept(_,B,_,_,_,_,_,_,_,_,_):-
time_after_h(B,9))]
Session started on 07/04/2011 23:34:36
and finished on 07/04/2011 23:34:36
```

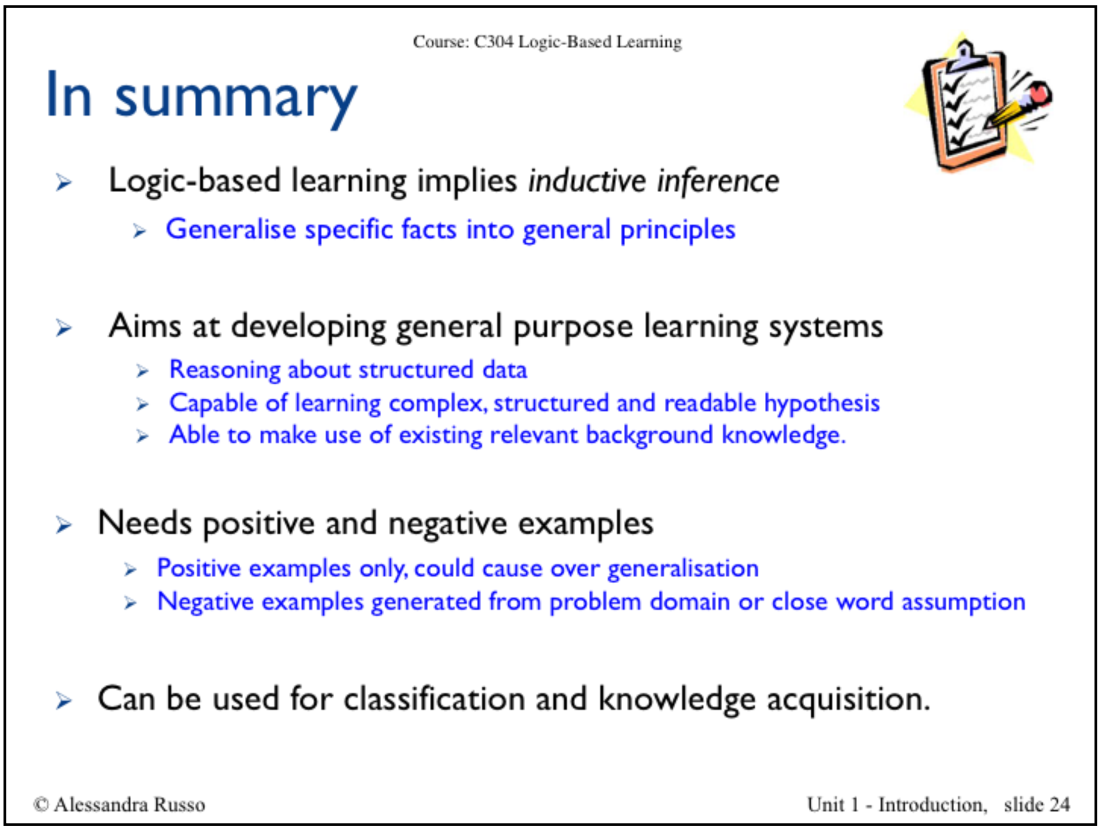
Rule: Accept calls: not when your phone's bettery is half-full, OR when you're not active

Accuracy: 74.0%

ON
```
Prolog rule: [(accept(_,_,_,_,_,_,F,_,_,_,_):-
\+F=34),(accept(M,N,_,_,_,_,_,_,_,_,_):-
\+user_is_active(M,N))]
Session started on 07/04/2011 23:34:36
and finished on 07/04/2011 23:34:36
```

© Alessandra Russo Unit 1 - Introduction, slide 23

This is an example of mobile user interface where learned rules are visualised to the user in structural English. The user is able to give feedback to the learner by choosing particular set of conditions upon which decisions to accept or reject calls should be made.


Course: C304 Logic-Based Learning

In summary

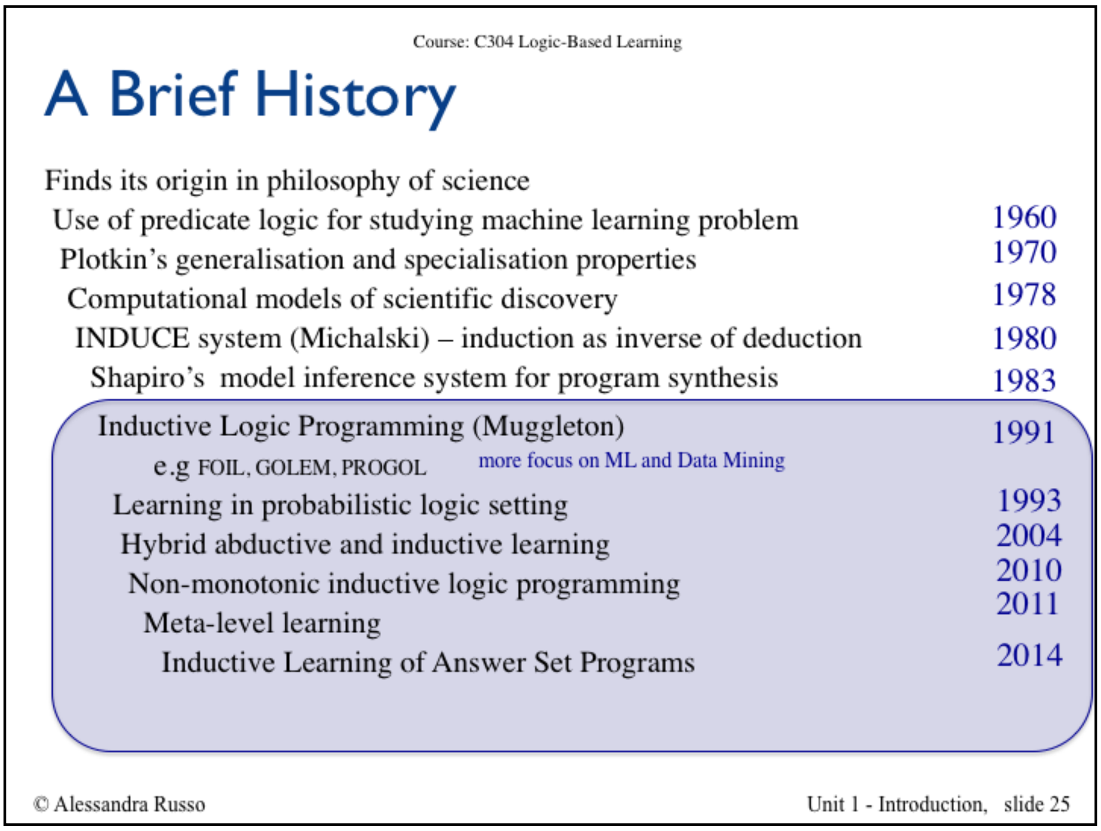
- Logic-based learning implies *inductive inference*
  - Generalise specific facts into general principles
- Aims at developing general purpose learning systems
  - Reasoning about structured data
  - Capable of learning complex, structured and readable hypothesis
  - Able to make use of existing relevant background knowledge.
- Needs positive and negative examples
  - Positive examples only, could cause over generalisation
  - Negative examples generated from problem domain or close word assumption
- Can be used for classification and knowledge acquisition.

© Alessandra Russo Unit 1 - Introduction, slide 24

In summary the purpose of logic-based learning is to develop general-purpose inductive reasoning systems that can be applied across different application domains.

Logic-based learning makes use of a form of reasoning known as inductive inference. We will see in the next lecture the main three types of reasoning (deductive, abductive and inductive). The characteristic of inductive reasoning is the ability to generalise specific facts into general principles. It was initially studied and explored within the context of natural science, where the emphasis is to conduct the inductive methodology of experiments, observation of data, generalisation of the hypothesis and testing of the hypothesis. To support the computation of scientific discovery processes, logic-based learning has to deal with two main computational problems: firstly, be able to express *complex readable hypothesis*. Scientific theories are complex and as such they requires an expressive formalism for representing them, that can also expressed in a readable form. Second, learning process should be able to make use and employ any existing relevant knowledge, called background knowledge, when generating meaningful hypothesis.

Logic-based learning is therefore not necessarily a machine learning technique for classification. It is used also for generating new knowledge, new principles, new rules about a particular application problem. To guide the search for the correct new knowledge it is important to have both positive and negative examples. As we have illustrated in the case of NLP and business process modelling, positive examples only may lead to hypothesis, or general principles that are over general. This means that they may generate more false positive (i.e. prove unseen/unused examples known to be false). Negative examples are therefore important to avoid such over-generalisations.



Course: C304 Logic-Based Learning

A Brief History

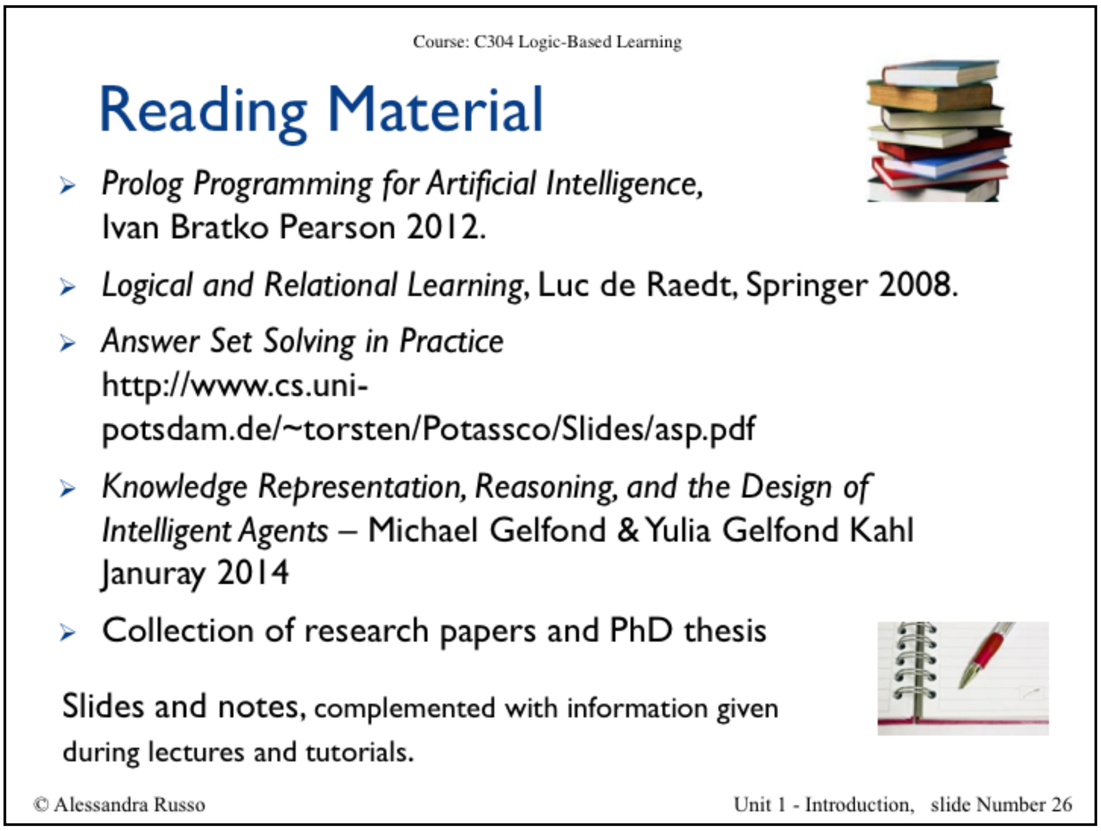
Finds its origin in philosophy of science	
Use of predicate logic for studying machine learning problem	1960
Plotkin's generalisation and specialisation properties	1970
Computational models of scientific discovery	1978
INDUCE system (Michalski) – induction as inverse of deduction	1980
Shapiro's model inference system for program synthesis	1983
Inductive Logic Programming (Muggleton)	1991
e.g FOIL, GOLEM, PROGOL — more focus on ML and Data Mining	
Learning in probabilistic logic setting	1993
Hybrid abductive and inductive learning	2004
Non-monotonic inductive logic programming	2010
Meta-level learning	2011
Inductive Learning of Answer Set Programs	2014

© Alessandra Russo — Unit 1 - Introduction, slide 25

The field of logic-based learning finds its origins in philosophical science, when philosophers like Charles Peirce, Rudolf Carnap and Karl Popper started to investigate and put forward the idea of empiricism and inductive methodology, according to which knowledge can be gained through a cycle of careful experimentation, observation, generalization and testing of hypotheses. Logic-based learning then started to be seen as a computational approach to inductive reasoning. Along this line, learning systems started to be developed to assist scientists in extracting scientific laws, and in parallel since the mid-60 researchers, motivated by limitations in early machine learning systems, started to use logic-based formalism to studying machine learning problems (such as pattern recognition).

First theoretical breakthrough was with the proposal of theoretical properties of generalisation and specialization operators for clausal logic, by Plotkin, that paved the path toward the notion of generalisation in logical-learning. Inference mechanisms and inverse operation of deduction were initially formalised in the INDUCE system (1980) where emphasis on the use of background knowledge during the learning process started also to become more predominant.

First contribution of induction in the context of logic programming came with Shapiro in the the early '80 when his approach called model inference system became the most powerful inductive inference system for program synthesis. The field of Inductive Logic Programming (ILP) started in the 1991, when advances by Muggleton of the theory of inductive inference in the context of Horn clauses. Learning systems started to be developed, including FOIL, GOLEM and PROGOL and successful application in real-life problem domains started to be internationally recognised (Ross King et al. 1992). Since then numerous learning approaches have been proposed that in turn aimed to overcome limitations on the completeness of the existing algorithms (e.g. PROGOL5) and expressiveness of the logical formalism used (i.e. moving from horn clauses to normal logic programs to answer set programming). In parallel, during the '90 research interest started also to be directed towards probabilistic logic frameworks and examples of learning in this new setting also started to be developed (Sato1995). In this course we will focus on developments in ILP in the last 20 years including very recent developments on logic-based learning of Answer Set Programs.



Course: C304 Logic-Based Learning

Reading Material

- *Prolog Programming for Artificial Intelligence,* Ivan Bratko Pearson 2012.
- *Logical and Relational Learning*, Luc de Raedt, Springer 2008.
- *Answer Set Solving in Practice* http://www.cs.uni-potsdam.de/~torsten/Potassco/Slides/asp.pdf
- *Knowledge Representation, Reasoning, and the Design of Intelligent Agents* – Michael Gelfond & Yulia Gelfond Kahl Januray 2014
- Collection of research papers and PhD thesis

Slides and notes, complemented with information given during lectures and tutorials.

© Alessandra Russo Unit 1 - Introduction, slide Number 26

The first textbook is a good introduction to logic programming and basic notions of Prolog that we will be referring to during this course.

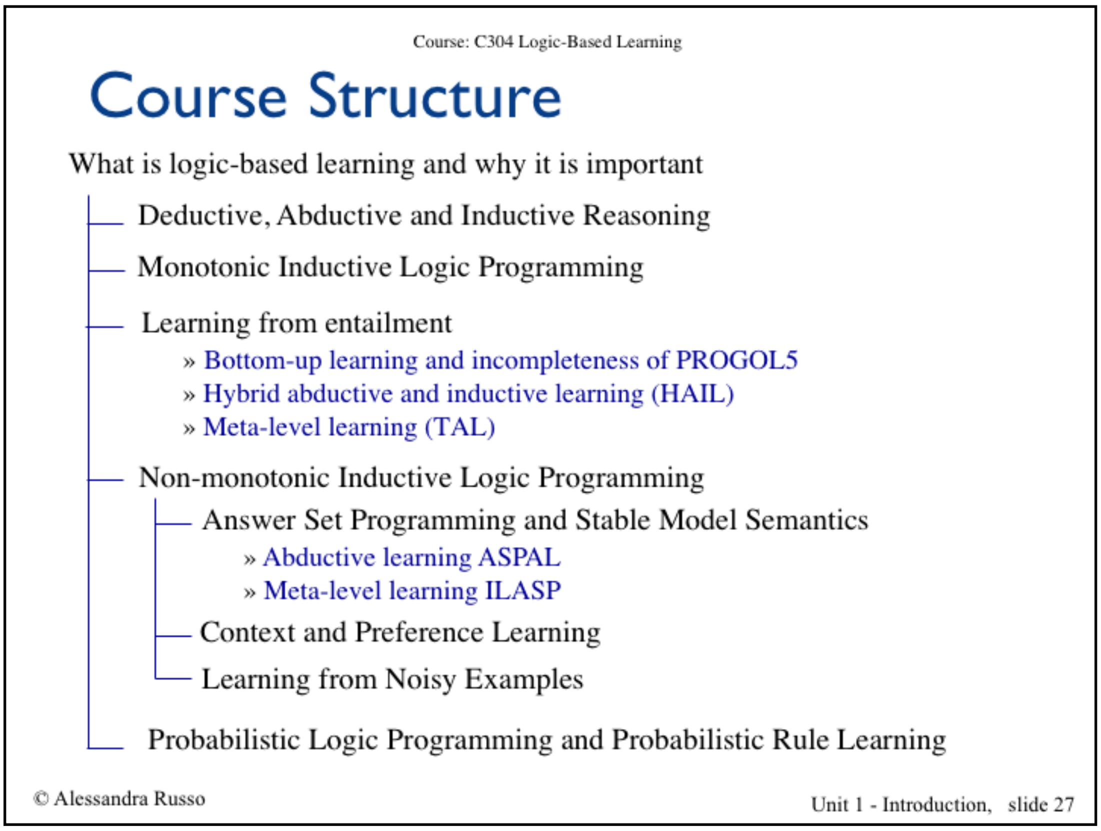
The second textbook provides a nice summary in the first few chapters of why logic-based learning is different from data mining and other machine learning classification approaches. This lecture summarise some of the content in the first chapter. The second chapter is a clear simple introduction to logic. Third year students would find this chapter and the next lecture easy, whereas MSc conversion students and MSc specialists who don't have much experience with logic should read this chapter. The next lecture is an extended version of the second chapter of this book.

In the forth teaching week we will be introducing a particular type of computational logic environment called Answer Set Programming (ASP). This is a wide area in computational logic and Potsdam university has developed one of the mostly used programming environments for solving problems written in Answer Set Programming. The program is called "Clingo" and we expect students to familiarise with it as some of the logic-based learning approaches that we will be covering are based on ASP. The second link in the slide provides a simple introduction to ASP.

Some of the learning algorithms presented in this course have recently been developed in the research community of Inductive Logic Programming and most comprehensive explanations can be found in related PhD thesis.

Learning systems will be accessible through the following Web-based system https://teaching.doc.ic.ac.uk/ilp/frameworks/. You will be able to use it in order to practice with these systems to better understand the type of learning output that different systems would generate for given different learning tasks.

The part on probabilistic logic programming and probabilistic rule learning we will based on key research papers. I will indicate the relevant papers in the notes of the related slides.


Course: C304 Logic-Based Learning

Course Structure

What is logic-based learning and why it is important

- Deductive, Abductive and Inductive Reasoning
- Monotonic Inductive Logic Programming
- Learning from entailment
  - » Bottom-up learning and incompleteness of PROGOL5
  - » Hybrid abductive and inductive learning (HAIL)
  - » Meta-level learning (TAL)
- Non-monotonic Inductive Logic Programming
  - Answer Set Programming and Stable Model Semantics
    - » Abductive learning ASPAL
    - » Meta-level learning ILASP
  - Context and Preference Learning
  - Learning from Noisy Examples
- Probabilistic Logic Programming and Probabilistic Rule Learning

© Alessandra Russo

Unit 1 - Introduction, slide 27

Note that a variety of projects could be considered for undergraduate and Master students in both theoretical advances of logic-based framework and real-world problems, as the ones described in the past few slides.