Semantic Web

— to Artificial Intelligence

Yue Ma

Laboratoire de Recherche en Informatique (LRI)
Université Paris Sud
ma@lri.fr

About me

- 2008 PhD in Mathematics, Peking University, China
- 2008-2012 Postdoc in Computer Science, Université Paris Nord, France
- 2012-2014 Research Assistant in Computer Science, TU Dresden, Germany
- since September 2014 : Associate Professor in Computer Science, University Paris Sud, France

Research topics :

Semantic Web, Ontology, Automated Reasoning

My expectation :

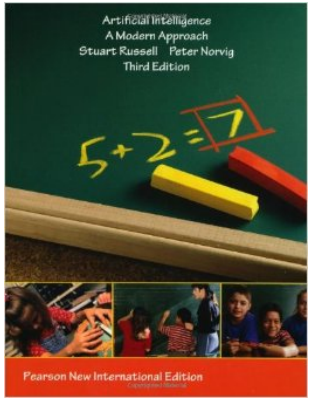
give you a solid foundation on Semantic Web

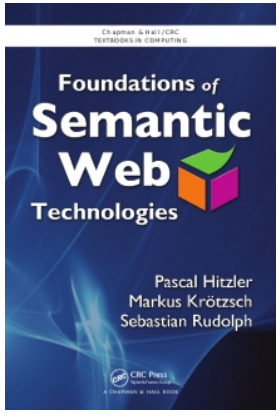
help your D2K career

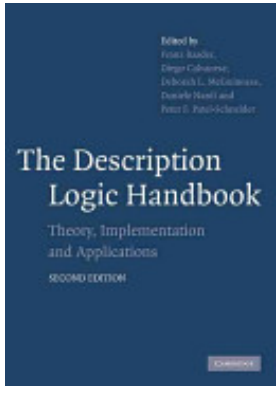
General Information

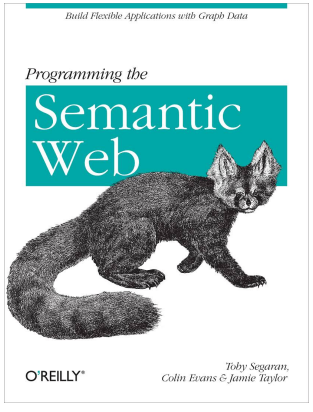
- This lecture is inspired by the following courses :
  - Knowledge Representation for the Semantic Web
    Pascal Hitzler, Wright State University, US
  - Foundations of Semantic Web Technologies
    Sebastian Rudolph, TU Dresden, Germany
  - Introduction à l'intelligence artificielle
    Laurent Simon, Polytech Paris Sud

- Reference books :









General Information

- Course materials : `www.lri.fr/~ma/M2DK`
- Organizational matters :
  - 4.5 lectures + exercices, 2.5 labs
  - Initial plans :
    - 1st week : introduction+ first deductive reasoning
    - 2nd week : implementation of the deductive reasoning
    - 3rd week : semantic web standards, description logics (DLs)
    - 4th week : exercises on DLs + building your first ontology
    - 5th week : more DLs
    - 6th week : working with ontology reasoners
    - 7th week : extensions and homework presentation
- Grading : 25% Project + 25% Homework + 50% Exam
  - Project : implementation of a reasoning algorithm (individual)
  - Homework : design a semantic web application (teamwork)
  - Exam : written

General Information

- Course materials : `www.lri.fr/~ma/M2DK`
- Organizational matters :
  - 4.5 lectures + exercices, 2.5 labs
  - Initial plans :
    - 1st week : introduction+ first deductive reasoning
    - 2nd week : implementation of the deductive reasoning
    - 3rd week : semantic web standards, description logics (DLs)
    - 4th week : exercises on DLs + building your first ontology
    - 5th week : more DLs
    - 6th week : working with ontology reasoners
    - 7th week : extensions and homework presentation
- Grading : 25% Project + 25% Homework + 50% Exam
  - Project : implementation of a reasoning algorithm (individual)
  - Homework : design a semantic web application (teamwork)
  - Exam : written

Outline

Why Semantic Web ?

Semantics and Knowledge Representation

- Principle
- Syntax and Semantics
- Propositional (logic) language

Inference Algorithms for a special PL

- A special PL sub-language : Horn rules
- Forward Chaining
- Backforward Chaining

MIT News

Tim Berners-Lee was honored with the Turing Award for his work inventing the World Wide Web, the first web browser, and "the fundamental protocols and algorithms [that allowed] the web to scale."

Photo: Henry Thomas

Tim Berners-Lee wins $1 million Turing Award

CSAIL researcher honored for inventing the web and developing the protocols that spurred its global use.

Adam Conner-Simons | CSAIL
April 4, 2017

The Current Web

- WWW is penetrating our society (immensely successful)
- Huge amounts of data
- Syntax standards for transfer of structured data
- Machine-processable, human-readable documents

BUT

- Content/knowledge cannot be accessed by machines.
- Meaning (semantics) of transferred data is not accessible.

Limits of the current Web

- Too much information with too little structure made for human consumption
  - Content search is very simplistic
  - future requires better methods
- Web content is heterogeneous
  - in terms of content
  - in terms of structure
  - in terms of character encoding
  - future requires intelligent information integration
- Humans can derive new (implicit) information from given pieces of information but on the current Web we can only deal with syntax.
  - requires automated reasoning techniques

Limits of the current Web

- Too much information with too little structure made for human consumption
  - Content search is very simplistic
  - future requires better methods
- Web content is heterogeneous
  - in terms of content
  - in terms of structure
  - in terms of character encoding
  - future requires intelligent information integration
- Humans can derive new (implicit) information from given pieces of information but on the current Web we can only deal with syntax.
  - requires automated reasoning techniques

Limits of the current Web

- Too much information with too little structure made for human consumption
  - Content search is very simplistic
  - future requires better methods
- Web content is heterogeneous
  - in terms of content
  - in terms of structure
  - in terms of character encoding
  - future requires intelligent information integration
- Humans can derive new (implicit) information from given pieces of information but on the current Web we can only deal with syntax.
  - requires automated reasoning techniques

Examples

- Find that landmark article on data integration written by an Indian researcher in the 1990s.

  [How can you get the answer ?]

- Are lobsters spiders ?

  [This is getting easier these days, but was impossible a few years ago. It still needs finding and integrating over different websites, as well as some background knowledge.]

- Which car is called a "duck" in German ?

  [This needs some intelligent integration of content from different websites plus background knowledge.]

Examples

- Find that landmark article on data integration written by an Indian researcher in the 1990s.

  [How can you get the answer ?]

- Are lobsters spiders ?

  [This is getting easier these days, but was impossible a few years ago. It still needs finding and integrating over different websites, as well as some background knowledge.]

- Which car is called a "duck" in German ?

  [This needs some intelligent integration of content from different websites plus background knowledge.]

Examples

- Find that landmark article on data integration written by an Indian researcher in the 1990s.

  [How can you get the answer ?]

- Are lobsters spiders ?

  [This is getting easier these days, but was impossible a few years ago. It still needs finding and integrating over different websites, as well as some background knowledge.]

- Which car is called a “duck” in German ?

  [This needs some intelligent integration of content from different websites plus background knowledge.]

Examples

- Find that landmark article on data integration written by an Indian researcher in the 1990s.

  [How can you get the answer ?]

- Are lobsters spiders ?

  [This is getting easier these days, but was impossible a few years ago. It still needs finding and integrating over different websites, as well as some background knowledge.]

- Which car is called a “duck” in German ?

  [This needs some intelligent integration of content from different websites plus background knowledge.]

Examples

- Find that landmark article on data integration written by an Indian researcher in the 1990s.

  [How can you get the answer ?]

- Are lobsters spiders ?

  [This is getting easier these days, but was impossible a few years ago. It still needs finding and integrating over different websites, as well as some background knowledge.]

- Which car is called a “duck” in German ?

  [This needs some intelligent integration of content from different websites plus background knowledge.]

Examples

- Find that landmark article on data integration written by an Indian researcher in the 1990s.

  [How can you get the answer ?]

- Are lobsters spiders ?

  [This is getting easier these days, but was impossible a few years ago. It still needs finding and integrating over different websites, as well as some background knowledge.]

- Which car is called a “duck” in German ?

  [This needs some intelligent integration of content from different websites plus background knowledge.]

Another Example

Identify congress members, who have voted "No" on pro environmental legislation in the past four years, with high-pollution industry in their congressional districts.

In principle, all the required knowledge is on the Web – most of it even in machine-readable form.

However, without automated processing and reasoning we cannot obtain a useful answer.

Academic questions :

Which is the defective protein causing the lysosomal storage disease Fabry ?
Answer : [‘alpha-galactosidase A’]

“Anderson-Fabry disease (referred to as Fabry disease) is an X-linked disorder characterized by a deficiency of the lysosomal enzyme alpha-galactosidase A and the subsequent accumulation in various tissues of globotriaosylceramide (Gb(3)), the main substrate of the defective enzyme.”

Daily life questions :

What treatment should be taken for a baby with eczema ? WHY ?

Given the evidences/justifications to a user !!!

Homework

Design a system to answer such questions by exploring web data

- Specify your task
- Analyze the task (theoretical and practical issues)
- Propose a method to solve it (completely or partially)
- No need to implement each step, but you need to be precise why your proposal can be automatised by computers
- Teamwork : 2 persons

Basic ingredients for the Semantic Web

- Open Standards for describing information on the Web
- Methods for obtaining further information from such descriptions

We'll talk about these matters in this course.

Basic ingredients for the Semantic Web

- Open Standards for describing information on the Web
- Methods for obtaining further information from such descriptions

Main approach : Logical deduction (aka automated reasoning)
E.g.,

D.C. is a capital
Every capital is a city

Hence : D.C. is a city

Based on predicate logic. – it needs to be specified which conclusions are valid. Plus, we need algorithms for these.


Semantic Web

Outline

The main problems

We need to store/represent information :

- for a domain of interest
- to be possibly queried by users

Inference algorithms

Answers to queries should be computed by the algorithm independent on the application domain :

- It takes as input a knowledge base K and a query ;
- It can correctly answer the query according to the information given in K.

Main studies on :

- the expressivity of a formal language
- the soundness / completeness / complexity of inference algorithms for a chosen representation language

The main problems

We need to store/represent information :

- for a domain of interest
- to be possibly queried by users

Inference algorithms

Answers to queries should be computed by the algorithm independent on the application domain :

- It takes as input a knowledge base K and a query ;
- It can correctly answer the query according to the information given in K.

Main studies on :

- the expressivity of a formal language
- the soundness / completeness / complexity of inference algorithms for a chosen representation language

The main problems

We need to store/represent information :

- for a domain of interest
- to be possibly queried by users

Inference algorithms

Answers to queries should be computed by the algorithm independent on the application domain :

- It takes as input a knowledge base K and a query ;
- It can correctly answer the query according to the information given in K.

Main studies on :

- the expressivity of a formal language
- the soundness / completeness / complexity of inference algorithms for a chosen representation language

Formal Knowledge Representation Langauge

To be processable by machines, the language needs to be

- Formal (syntactically well defined)
- Precise (semantically well defined)

- like C, Java, Python, ...
- unlike ... ?

Formal Knowledge Representation Langauge

To be processable by machines, the language needs to be

- Formal (syntactically well defined)
- Precise (semantically well defined)

- like C, Java, Python, ...
- unlike ... ?

Syntax and Semantics

Syntax

is to represent the knowledge and queries in a machine readable format (similar to sentences and words in a book, but it's the object and variable symbols in the computer memory.)

Semantics

is to link the symbols to the real world, without which syntax is meaningless.

In short

- Computers manipulate the syntactical objects (i.e., the process of automatic inference)
- The manipulation is meaningful because of the meaning associated to the syntax.

Syntax and Semantics

Syntax

is to represent the knowledge and queries in a machine readable format (similar to sentences and words in a book, but it's the object and variable symbols in the computer memory.)

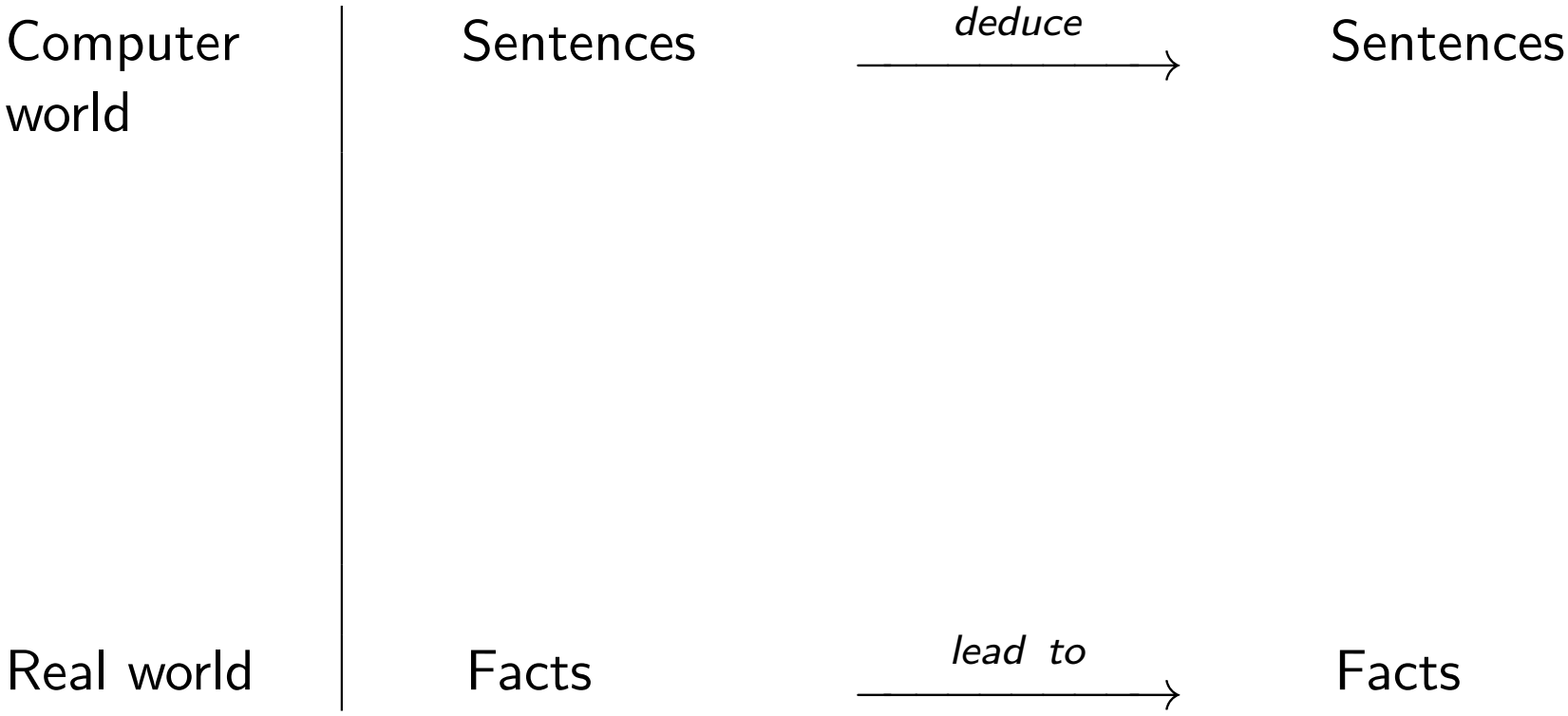
Semantics

is to link the symbols to the real world, without which syntax is meaningless.

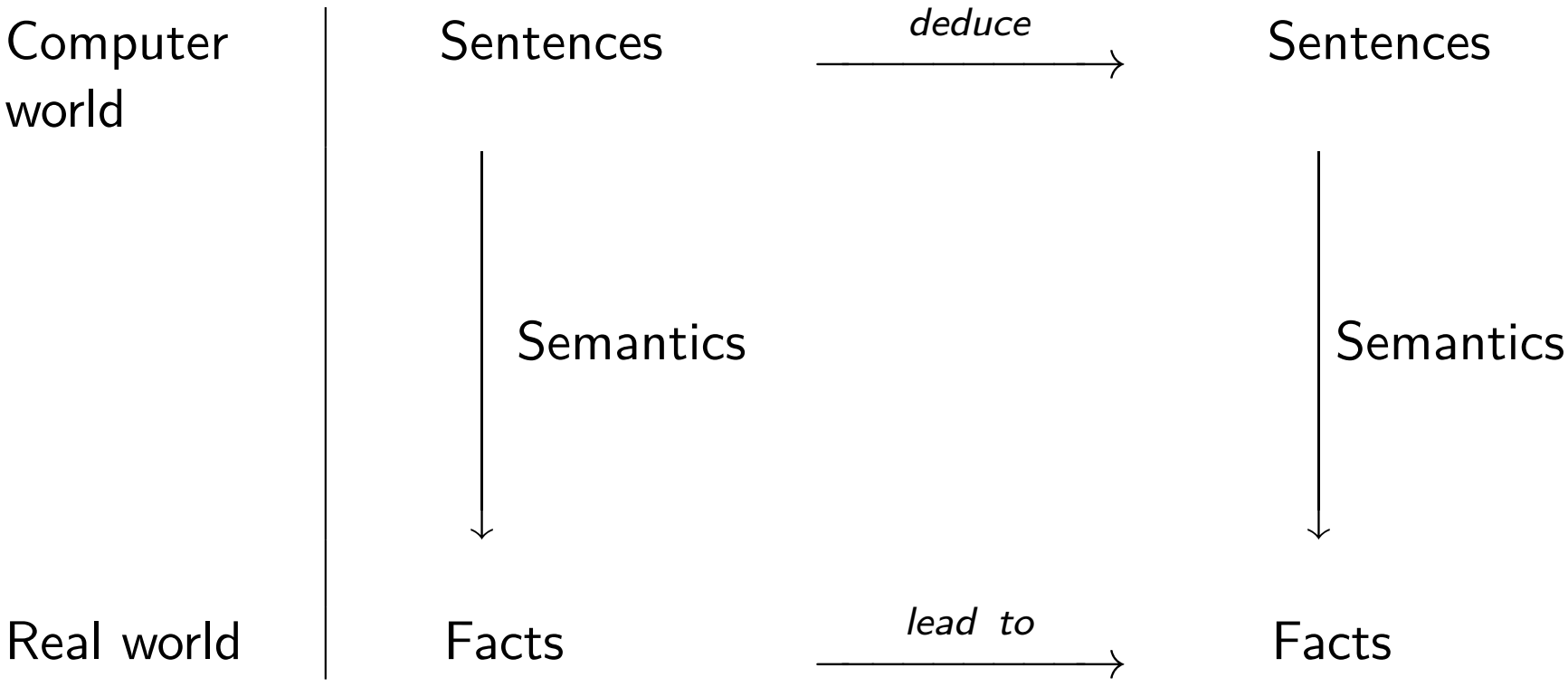
In short

- Computers manipulate the syntactical objects (i.e., the process of automatic inference)
- The manipulation is meaningful because of the meaning associated to the syntax.

Syntax / Semantics / Inference



Syntax / Semantics / Inference



Propositional Language (Syntax)

Let $\mathcal{V} = \{x_1, \ldots, x_n\}$ be a set of (propositional) variables, the langage $\mathcal{LP}$ of **propositions** is constructed inductively by applying the following rules :

- $\top, \bot \in \mathcal{LP}$;
- $x \in \mathcal{LP}$ for all $x \in \mathcal{V}$;
- $\neg f \in \mathcal{LP}$ for all $f \in \mathcal{LP}$;
- $f \vee g$, $f \wedge g$, $f \to g$, $\in \mathcal{LP}$ for all f, $g \in \mathcal{LP}$.

An equivalent notation :

The propositional formulas φ are built by the following grammar :

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \quad \text{with } p \in \mathcal{V}$$

Example

- $x_1 : \mathit{Capital}, x_2 : \mathit{City}$ in $\mathcal{V}$, so

  $\mathit{Capital} \to \mathit{City} \in \mathcal{LP}$

- $\mathcal{V} = \{x, w, \mathit{capital} \to x_2, x_3\}$ ok

Propositional Language (Syntax)

Let $\mathcal{V} = \{x_1, \ldots, x_n\}$ be a set of (propositional) variables, the langage $\mathcal{LP}$ of **propositions** is constructed inductively by applying the following rules :

- $\top, \bot \in \mathcal{LP}$;
- $x \in \mathcal{LP}$ for all $x \in \mathcal{V}$;
- $\neg f \in \mathcal{LP}$ for all $f \in \mathcal{LP}$;
- $f \vee g$, $f \wedge g$, $f \to g$, $\in \mathcal{LP}$ for all f, $g \in \mathcal{LP}$.

An equivalent notation :

The propositional formulas φ are built by the following grammar :

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \quad \text{with } p \in \mathcal{V}$$

Example

- $x_1 : \mathit{Capital}, x_2 : \mathit{City}$ in $\mathcal{V}$, so $\mathit{Capital} \to \mathit{City} \in \mathcal{LP}$
- $\mathcal{V} = \{v, w, x_1, y_1, x_2, y_2\}$, so

Propositional Language (Syntax)

Let $\mathcal{V} = \{x_1, \ldots, x_n\}$ be a set of (propositional) variables, the langage $\mathcal{LP}$ of **propositions** is constructed inductively by applying the following rules :

- $\top, \bot \in \mathcal{LP}$;
- $x \in \mathcal{LP}$ for all $x \in \mathcal{V}$;
- $\neg f \in \mathcal{LP}$ for all $f \in \mathcal{LP}$;
- $f \vee g$, $f \wedge g$, $f \to g$, $\in \mathcal{LP}$ for all f, $g \in \mathcal{LP}$.

An equivalent notation :

The propositional formulas φ are built by the following grammar :

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \quad \text{with } p \in \mathcal{V}$$

Example

- $x_1 : \mathit{Capital}, x_2 : \mathit{City}$ in $\mathcal{V}$, so

  $\mathit{Capital} \to \mathit{City} \in \mathcal{LP}$

- $\mathcal{V} = \{v, w, x_1, y_1, x_2, y_2\}$, so

Propositional Language (Syntax)

Let $\mathcal{V} = \{x_1, \ldots, x_n\}$ be a set of (propositional) variables, the langage $\mathcal{LP}$ of **propositions** is constructed inductively by applying the following rules :

- $\top, \bot \in \mathcal{LP}$;
- $x \in \mathcal{LP}$ for all $x \in \mathcal{V}$;
- $\neg f \in \mathcal{LP}$ for all $f \in \mathcal{LP}$;
- $f \vee g$, $f \wedge g$, $f \to g$, $\in \mathcal{LP}$ for all f, $g \in \mathcal{LP}$.

An equivalent notation :

The propositional formulas φ are built by the following grammar :

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \quad \text{with } p \in \mathcal{V}$$

Example

- $x_1 : \mathit{Capital}, x_2 : \mathit{City}$ in $\mathcal{V}$, so

  $\mathit{Capital} \to \mathit{City} \in \mathcal{LP}$

- $\mathcal{V} = \{v, w, x_1, y_1, x_2, y_2\}$, so

$$(((v \vee w) \vee \neg x_1) \wedge (v \vee \neg x_1 \vee \neg y_1)) \to (((y_1 \wedge y_2) \wedge \neg w) \wedge \neg x_2) \in \mathcal{LP}$$

Propositional Knowledge Bases

A propositional knowledge base K is a set of propositional formulae.
Example. $K = \{Paris \rightarrow Capital, Capital \rightarrow City\}$

Semantics for the propositional language

- The semantics : if $x =$"Paris is a capital", x is a true fact.
- Interpretations of a formula : assigning true/false to each variable

A formula with n variables, we obtain 2^n possible Interpretations, which can be generated by truth tables.

The truth tables (Semantics)

A propositional valuation/interpretation $I : \Pi \to \{0, 1\}$ is function mapping a propositional variable $p \in \Pi$ to a truth value $I(p) \in \{0, 1\}$. Let p and q be two proposition lettres, the semantics of a propositional formula is defined inductively by the following tables :

Négation

p	¬p
vrai	faux
faux	vrai

Conjonction

p	q	p ∧ q
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

Disjonction

p	q	P ∨ q
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

Implication

p	q	p → q
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	vrai
faux	faux	vrai

Equivalence

p	q	P ↔ q
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	vrai

Example : all interpretations of $(p \rightarrow (q \wedge r))$

interpretations	p	q	r	$(q \wedge r)$	$(p \rightarrow (q \wedge r))$
I_1	true	true	true	true	true
I_2	true	true	false	false	false
I_3	true	false	true	false	false
I_4	true	false	false	false	false
I_5	false	true	true	true	true
I_6	false	true	false	false	true
I_7	false	false	true	false	true
I_8	false	false	false	false	true

Example : all interpretations of $(p \rightarrow (q \wedge r))$

interpretations	p	q	r	$(q \wedge r)$	$(p \rightarrow (q \wedge r))$
I_1	true	true	true	true	true
I_2	true	true	false	false	false
I_3	true	false	true	false	false
I_4	true	false	false	false	false
I_5	false	true	true	true	true
I_6	false	true	false	false	true
I_7	false	false	true	false	true
I_8	false	false	false	false	true

Model of a formula

I is a model if I makes the formula α true, i.e. $I(\alpha) = \mathit{true}$.

Example : all interpretations of $(p \rightarrow (q \wedge r))$

interpretations	p	q	r	$(q \wedge r)$	$(p \rightarrow (q \wedge r))$
I_1	true	true	true	true	true
I_2	true	true	false	false	false
I_3	true	false	true	false	false
I_4	true	false	false	false	false
I_5	false	true	true	true	true
I_6	false	true	false	false	true
I_7	false	false	true	false	true
I_8	false	false	false	false	true

Model of a formula

I is a model if I makes the formula α true, i.e. $I(\alpha) = \mathit{true}$.

Example.

- I_5, I_6, I_7, I_8 are the models of $(p \rightarrow (q \wedge r))$.
- Let $K = \{x, \neg x\}$. Is K satisfiable ?

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. .... ?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. .... ?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. .... ?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. ....?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. .... ?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. .... ?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. ....?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

- Inconsistency : a formula A is called inconsistent or unsatisfiable if A does not have a model.

  Example : $p \wedge \neg p$ is inconsistent
- Consistency : a formula A is called consistent or satisfiable if the formula A has un model.

  Example : $p \vee q$ is a consistent formula.
- Tautology : a formula A is called valid (tautology) if A is true for all interpretations of their variables. We denote $\models A$

  Example : $(p \vee \neg p)$ is valid.
- logically equivalence : α is logically equivalent to β iff. ....?

  Example : $(p \rightarrow q)$ and $(\neg p \vee q)$ are logically equivalent. (Check by yourself)

Semantic Notions

Semantic entailment

A propositional knowledge base K implies a formula ϕ, written $K \models \phi$, iff. each model of K is a model of ϕ.

$K1 = \{x1 \rightarrow x2, x2 \rightarrow x3\}$, does $K \models x3$?

Unsatisfiability Theorem. $K \models \phi$ iff. $K \cup \{\neg\phi\}$ is .... ?

Semantic Notions

Semantic entailment

A propositional knowledge base K implies a formula ϕ, written $K \models \phi$, iff. each model of K is a model of ϕ.

$K1 = \{x1 \rightarrow x2, x2 \rightarrow x3\}$, does $K \models x3$?

Unsatisfiability Theorem. $K \models \phi$ iff. $K \cup \{\neg\phi\}$ is .... ?

Semantic Notions

Semantic entailment

A propositional knowledge base K implies a formula ϕ, written $K \models \phi$, iff. each model of K is a model of ϕ.

$K1 = \{x1 \rightarrow x2, x2 \rightarrow x3\}$, does $K \models x3$?

Unsatisfiability Theorem. $K \models \phi$ iff. $K \cup \{\neg\phi\}$ is .... ?

$K2 = \{x1 \rightarrow x2, x2 \wedge x3 \wedge \neg x1 \rightarrow x4, x4 \rightarrow \neg x1, x1...\}$, does $K2 \models x4$?

How to check $K \models \phi$?

- Using truth table (by definition)
- Using syntactic inference systems (Resolution principle, Nature deduction system, ...).

We write $K \vdash \phi$ if ϕ follows from K syntactically.

- $\vdash$ is called sound : if $K \vdash \phi$ then $K \models \phi$.
- $\vdash$ is called complete : if $K \models \phi$ $K \vdash \phi$.

$K2 = \{x1 \rightarrow x2, x2 \wedge x3 \wedge \neg x1 \rightarrow x4, x4 \rightarrow \neg x1, x1...\}$, does $K2 \models x4$?

How to check $K \models \phi$?

- Using truth table (by definition)
- Using syntactic inference systems (Resolution principle, Nature deduction system, ...).

We write $K \vdash \phi$ if ϕ follows from K syntactically.

- $\vdash$ is called sound : if $K \vdash \phi$ then $K \models \phi$.
- $\vdash$ is called complete : if $K \models \phi$ $K \vdash \phi$.

Tutorial 1 - ex1

Importance in computational complexity theory

Definition (Litteral, Clause)

- A **litteral** is eithert x (positive) or $\neg x$ (negative litteral) with $x \in \mathcal{V}$. x and $\neg x$ are called **complementary**.
- A **clause** is a disjunction of litterals.

Example (Clause)

Sont des clauses :

- $a \vee \neg b \vee c$
- $(a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee c)$

SAT problem : Given a propositional CNF formula, decide if this formula is satisfiable (i.e., has a model).

Importance in computational complexity theory

Definition (Litteral, Clause)

- A **litteral** is eithert x (positive) or $\neg x$ (negative litteral) with $x \in \mathcal{V}$. x and $\neg x$ are called **complementary**.
- A **clause** is a disjunction of litterals.

Example (Clause)

Sont des clauses :

- $a \vee \neg b \vee c$
- $(a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee c)$

SAT problem : Given a propositional CNF formula, decide if this formula is satisfiable (i.e., has a model).
NP-complete Problems : we don't know any polynomial algorithm to solve it.

3-SAT and 2-SAT

Complexities :

- NP-completeness of 3-SAT : 3-SAT and general SAT are a NP-C problems by Cook Theorem
- 2-SAT is in the complexity class $\mathcal{P}$: There are several polynomial algorithms to solve the 2-SAT problem.

Every decision problem in the complexity class NP can be reduced to the SAT problem !
Many problems are proven to be NP-hard due to a polynomial deduction from SAT problem (i.e., deciding whether a given graph has a 3-coloring)

3-SAT and 2-SAT

Complexities :

- NP-completeness of 3-SAT : 3-SAT and general SAT are a NP-C problems by Cook Theorem
- 2-SAT is in the complexity class $\mathcal{P}$: There are several polynomial algorithms to solve the 2-SAT problem.

Every decision problem in the complexity class NP can be reduced to the SAT problem !

Many problems are proven to be NP-hard due to a polynomial deduction from SAT problem (i.e., deciding whether a given graph has a 3-coloring)

3-SAT and 2-SAT

Complexities :

- NP-completeness of 3-SAT : 3-SAT and general SAT are a NP-C problems by Cook Theorem
- 2-SAT is in the complexity class $\mathcal{P}$: There are several polynomial algorithms to solve the 2-SAT problem.

Every decision problem in the complexity class NP can be reduced to the SAT problem !
Many problems are proven to be NP-hard due to a polynomial deduction from SAT problem (i.e., deciding whether a given graph has a 3-coloring)

Solving puzzles by propositional logics

Given the following facts :

- Jean says : If Bernard is guilty, Sophie is too.
- Bernard says : Jean is guilty and Sophie is not.
- Sophie says : she is not guilty but at least one of the others are

We assume that each one lies if and only if he/she is guilty.

- Modeling the knowledge in propositional logic.
- Can we deduce who is guilty ?

Try yourselves.

Outline

Inference Algorithms

General algorithm for PL inference can be as expensive as truth table enumeration ! (exponential)

Syntax restrictions of formulae, Horn

Only formulae of the form are allowed in a knowledge base :

$$l_1 \wedge l_2 \wedge l_3 \wedge ... \wedge l_n \rightarrow l$$

Inference Algorithms

General algorithm for PL inference can be as expensive as truth table enumeration ! (exponential)

Syntax restrictions of formulae, Horn

Only formulae of the form are allowed in a knowledge base :

$$l_1 \wedge l_2 \wedge l_3 \wedge ... \wedge l_n \rightarrow l$$

Chaining from two sides

Two chaining algorithms

Forward Chaining

Principles : starting from a base of initial facts, deduce new facts according to rules in the base

- **Releasable rule** : all of its conditions are satisfiable
- **New facts** : the conclusions of releasable rules until saturation or the queried goal is achieved

Backward Chaining

Principles : starting from the goal, replace the goal by sub-goals.
Sub-goals : the conditions of a rule whose conclusion is a goal until that all the sub-goals are either facts or there is not sub-goal can be replaceable (not a conclusion of any rules)

Chaining from two sides

Two chaining algorithms

Forward Chaining

Principles : starting from a base of initial facts, deduce new facts according to rules in the base

- **Releasable rule** : all of its conditions are satisfiable
- **New facts** : the conclusions of releasable rules until saturation or the queried goal is achieved

Backward Chaining

Principles : starting from the goal, replace the goal by sub-goals.
Sub-goals : the conditions of a rule whose conclusion is a goal until that all the sub-goals are either facts or there is not sub-goal can be replaceable (not a conclusion of any rules)

Forward Chaining Algorithm

Iterative Evaluation of Rules over a base of facts

FB : Fact base ; RB : Rule base
$ForwardChaining(FB_{init}, RB)$ computes the smallest fix point of the operator $Cons$ of immediate consequences from FB_{init}

- Cons is applied over FB if there exists a releasable rule over FB
- The result $Cons(FB)$ is obtained by adding FB the conclusion of the rule

Evaluation of a rule

Let FB be a fact base, and the rule R : **if** I_1 **and** ... **and** I_n **then** $I\!I$.

- $match(I, FB) = true$ iff $I \in FB$
- $eval(R, FB) = true$ iff for all i, $match(I_i, FB) = true$

Forward Chaining (ctd.)

```
1 : function ForwardChaining(FB_init, RB)
2 :     s(FB) ← FB_init
3 :     repeat
4 :         FB ← s(FB)
5 :         for all Rule R is not yet applied do
6 :             if eval(R, FB) then s(FB) ← s(FB) ∪ conclusions(R)
7 :             end if
8 :         end for
9 :     until s(FB) = FB
10 :     return s(FB)
11 : end function
```

Forward Chaining (ctd.)

Evaluation of a query Q by forward chaining

Q is satisfiable iff Q appears in the result of $\mathit{ForwardChaining}(FB_{init}, RB)$

Result : the algorithm $\mathit{ForwardChaining}(FB_{init}, RB)$ is polynomial w.r.t. the size of sFB_{init}, the size of RB, and the size of a rule.

Forward Chaining (example)

Tutorial 1, ex2-Q1

Optimisation of the forward chaining algorithm

Idea : don't repeat the operation of making correspondences between facts and conditions

- from one rule to another
- from one iteration to another

We proceed with the propagation of facts over the rules where the facts appear as condition.

- need indexing of atom-conditions regarding rules : $\mathit{InRuleConditions}(I) = \{R \mid I \text{ is an atom appearing in the condition of } R\}$
- for each rule, we keep memorizing the conditions satisfied in previous iterations.
  We update the rules by keeping only their conditions that are not yet satisfied while the propagation of a fact f :

$$\mathit{conditions}(R) \leftarrow \mathit{conditions}(R) - f$$

Optimisation of the forward chaining algorithm

Idea : don't repeat the operation of making correspondences between facts and conditions

- from one rule to another
- from one iteration to another

We proceed with the propagation of facts over the rules where the facts appear as condition.

- need indexing of atom-conditions regarding rules : $\mathit{InRuleConditions}(l) = \{R \mid l \text{ is an atom appearing in the condition of } R\}$
- for each rule, we keep memorizing the conditions satisfied in previous iterations.
  We update the rules by keeping only their conditions that are not yet satisfied while the propagation of a fact f :

$$\mathit{conditions}(R) \leftarrow \mathit{conditions}(R) - f$$

Algorithm of Forward Chaining with Propagation

function *ForwardChainingBis*(FB_{init}, RB)
    $FB \leftarrow FB_{init}$
    for all fact $F \in FB_{init}$ **do**
        $FB \leftarrow FB \cup$ *Propagate*(F, RB)
    end for
    return FB
end function

function *Propagate*(F, RB)
    $\Delta F \leftarrow \emptyset$
    for all rule $R \in$ *InRuleConditions*(F) **do**
        Delete from *conditions*(R) the atom I such that *match*(I, *condition*(R))
        if *conditions*(R) $= \emptyset$ **then**
            $RB \leftarrow RB \setminus \{R\}$
            $\Delta F \leftarrow \Delta F \cup$ *conclusions*(R)
        end if
    end for
    $FB \leftarrow \Delta F$
    for all fact $F \in \Delta F$ **do**
        $FB \leftarrow FB \cup$ *Propagate*(F, RB)
    end for
    return FB
end function

Algorithm of Forward Chaining with Propagation

function $ForwardChainingBis(FB_{init}, RB)$
    $FB \leftarrow FB_{init}$
    for all fact $F \in FB_{init}$ **do**
        $FB \leftarrow FB \cup Propagate(F, RB)$
    end for
    return FB
end function

function $Propagate(F, RB)$
    $\Delta F \leftarrow \emptyset$
    for all rule $R \in InRuleConditions(F)$ **do**
        Delete from $conditions(R)$ the atom I such that $match(I, condition(R))$
        if $conditions(R) = \emptyset$ **then**
            $RB \leftarrow RB \setminus \{R\}$
            $\Delta F \leftarrow \Delta F \cup conclusions(R)$
        end if
    end for
    $FB \leftarrow \Delta F$
    for all fact $F \in \Delta F$ **do**
        $FB \leftarrow FB \cup Propagate(F, RB)$
    end for
    return FB
end function

Example

Toturail 1, ex2-Q2/3

Backforward Chaining

Principle

Algorithm of replacing a goal by sub-goals, by matching a current goal (the user's query at beginning) with the conclusion atoms of rules.

When to use it ?

To guide the questions proposed by users in the case of incomplete information from FB to satisfy the query.

Generally, it is combined with the forward chaining :

- forward chaining to saturate the FB
- backward chaining to guide the interaction with users of a AND/OR tree.

Backforward Chaining

Principle

Algorithm of replacing a goal by sub-goals, by matching a current goal (the user's query at beginning) with the conclusion atoms of rules.

When to use it ?

To guide the questions proposed by users in the case of incomplete information from FB to satisfy the query.

Generally, it is combined with the forward chaining :

- forward chaining to saturate the FB
- backward chaining to guide the interaction with users of a AND/OR tree.

Backforward Chaining

Principle

Algorithm of replacing a goal by sub-goals, by matching a current goal (the user's query at beginning) with the conclusion atoms of rules.

When to use it ?

To guide the questions proposed by users in the case of incomplete information from FB to satisfy the query.

Generally, it is combined with the forward chaining :

- forward chaining to saturate the FB
- backward chaining to guide the interaction with users of a AND/OR tree.

Backward Chaining Algorithm

```
1 : function BackwardChaining(Q, FB_init, RB)
2 :     FB ← FB_init
3 :     if match(Q, FB) then
4 :         return true
5 :     else
6 :         for all Rule r : If l_1 And l_2 And ... And l_n Then Conc t.q.
    match(Q, Conc) do
7 :             bool ← true ; i ← 1
8 :             while bool And i ≤ n do
9 :                 bool ← BackwardChaining(l_i, FB, RB)
10 :                i ← i + 1
11 :            end while
12 :            if bool then return true
13 :            end if
14 :        end for
15 :        return false
16 :    end if
17 : end function
```

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with questions to users

Principles of the questions

To decide the questions that can be asked to users.

- *demandable*(l) : we can ask a question over l.Generally, if l doesn't appear in conclusions of rules, it is demandable.
- *question*(l) : boolean returning the answer of the user to the question : if l is *true*.

Modification Principles : If we cannot find a fact, we can ask the user as last resort.

Backward Chaining with interactive questions

function *BackwardChaining2*(Q, FB_{init}, RB)
  $FB \leftarrow FB_{init}$
  if *match*(Q, FB) **then**
    return *true*
  else
    for all rgle r : **if** l_1 **and** l_2 **and** ... **and** l_n **then** *Conc* t.q. *match*$(Q, Conc)$
do
      $bool \leftarrow true$; $i \leftarrow 1$
      while *bool* Et $i \leq n$ **do**
        $bool \leftarrow BackwardChaining(l_i, FB, RB)$
        $i \leftarrow i + 1$
      end while
      if *bool* **then return** *true*
      end if
    end for
    if *demandable*(Q) **then return** *question*(Q)
    elsereturn *false*
    end if
  end if
end function

Exemple

Tutorial 1, ex3.

Recursive Rules

Problem of loopback risk

Solution : don't re-apply the algorithm over the sub-goals that have been examined.

```
function BackwardChaining3(Q, FB_init, RB)
    FB ← FB_init
    if match(Q, FB) then
        return true
    else
        AlreadyTried ← ∅
        for all Rule r : if l_1 and l_2 and ... and l_n then Conc t.q. match(Q, Conc) do
            bool ← true ; i ← 1
            while bool And i ≤ n do
                if l_i ∉ AlreadyTried then
                    AlreadyTried ← AlreadyTried ∪ {l_i}
                    bool ← BackwardChaining(l_i, FB, RB)
                end if
                i ← i + 1
            end while
            if bool then return true
            end if
        end for
        if demandable(Q) then return question(Q)
        elsereturn false
        end if
    end if
end function
```

Recursive Rules

Problem of loopback risk

Solution : don't re-apply the algorithm over the sub-goals that have been examined.

function *BackwardChaining*3(Q, FB_{init}, RB)
  $FB \leftarrow FB_{init}$
  if *match*(Q, FB) **then**
    return *true*
  else
    AlreadyTried $\leftarrow \emptyset$
    for all Rule r : **if** l_1 **and** l_2 **and** ... **and** l_n **then** *Conc* t.q. *match*(Q, *Conc*) **do**
      bool $\leftarrow$ *true* ; $i \leftarrow 1$
      while *bool* And $i \leq n$ **do**
        if $l_i \notin$ *AlreadyTried* **then**
          AlreadyTried $\leftarrow$ *AlreadyTried* $\cup \{l_i\}$
          bool $\leftarrow$ *BackwardChaining*(l_i, FB, RB)
        end if
        $i \leftarrow i + 1$
      end while
      if *bool* **then return** *true*
      end if
    end for
    if *demandable*(Q) **then return** *question*(Q)
    elsereturn *false*
    end if
  end if
end function