Introduction to Inductive Logic Programming

Manoel V. M. França

Department of Computing
City University London

March 26, 2012 / Machine Learning Group Meeting

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming

CITY UNIVERSITY LONDON

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol

CITY UNIVERSITY LONDON

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++

CITY UNIVERSITY LONDON

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Thinking and Explaining

- Imagine a father teaching his little daughter how to drink a glass of water

Thinking and Explaining

- Imagine a father teaching his little daughter how to drink a glass of water
  - He needs to use a proper language to teach her each step involved

Thinking and Explaining

- Imagine a father teaching his little daughter how to drink a glass of water
  - He needs to use a proper language to teach her each step involved
  - It should be as clear as possible

Thinking and Explaining

- Imagine a father teaching his little daughter how to drink a glass of water
  - He needs to use a proper language to teach her each step involved
  - It should be as clear as possible
  - The representation of each object should be good enough for that

Thinking and Explaining

- Imagine a father teaching his little daughter how to drink a glass of water
  - He needs to use a proper language to teach her each step involved
  - It should be as clear as possible
  - The representation of each object should be good enough for that
- Which would fit best?

Thinking and Explaining

- Imagine a father teaching his little daughter how to drink a glass of water
  - He needs to use a proper language to teach her each step involved
  - It should be as clear as possible
  - The representation of each object should be good enough for that
- Which would fit best?
  - "To drink water, you need to grab a glass and use a sink"

Thinking and Explaining

- Imagine a father teaching his little daughter how to drink a glass of water
  - He needs to use a proper language to teach her each step involved
  - It should be as clear as possible
  - The representation of each object should be good enough for that
- Which would fit best?
  - "To drink water, you need to grab a glass and use a sink"
  - water, glass $\in [0, 1]$, threshold = 2,
    drinking = ((water + glass) $\geq$ threshold)

Symbolic Arguments

- “The Neural Networks did perform well (...). However, (...) they consumed enormous amounts of CPU time and they are sometimes equaled by simple symbolic classifiers” (Weiss and Kapouleas, 1989)

Symbolic Arguments

- "The Neural Networks did perform well (...). However, (...) they consumed enormous amounts of CPU time and they are sometimes equaled by simple symbolic classifiers" (Weiss and Kapouleas, 1989)
- "Structured knowledge, is difficult to represent in Neural Networks, contrary to traditional logical models" (Toiviainen, 2000)

Symbolic Arguments

- “The Neural Networks did perform well (...). However, (...) they consumed enormous amounts of CPU time and they are sometimes equaled by simple symbolic classifiers” (Weiss and Kapouleas, 1989)
- “Structured knowledge, is difficult to represent in Neural Networks, contrary to traditional logical models” (Toiviainen, 2000)
- “Attempts have been made to explain the behavior of connectionist networks (...). These explanations are, however, at the level of primitive features of the network (...). Explanations on a higher level of knowledge are difficult to achieve” (Toiviainen, 2000)

Connectionistic Arguments

- “Connectionist networks are robust. (...) They are resistant to noise and gracefully degrade when they are damaged or overloaded with information” (Smolensky et al., 1992)

Connectionistic Arguments

- “Connectionist networks are robust. (...) They are resistant to noise and gracefully degrade when they are damaged or overloaded with information” (Smolensky et al., 1992)
- “Neural Networks are capable of extracting significant features from the training set and using them to process a novel input pattern, thus generalizing better” (Toiviainen, 2000)

Connectionistic Arguments

- "Connectionist networks are robust. (...) They are resistant to noise and gracefully degrade when they are damaged or overloaded with information" (Smolensky et al., 1992)
- "Neural Networks are capable of extracting significant features from the training set and using them to process a novel input pattern, thus generalizing better" (Toiviainen, 2000)
- "Differently from (symbolic) machine learning, (numeric) neural networks perform inductive learning in such a way that the statistical characteristics of the data are encoded in their sets of weights" (Garcez et al., 2009)

Which Means...

- Both paradigms has its own strengths and weaknesses

Which Means...

- Both paradigms has its own strengths and weaknesses
- Both are usually better suited on different kinds of applications

Which Means...

- Both paradigms has its own strengths and weaknesses
- Both are usually better suited on different kinds of applications
- **Both are important and relevant to a complete reasoning model**

The World Without Logic (1)

Knowledge Representation:

With Logic:

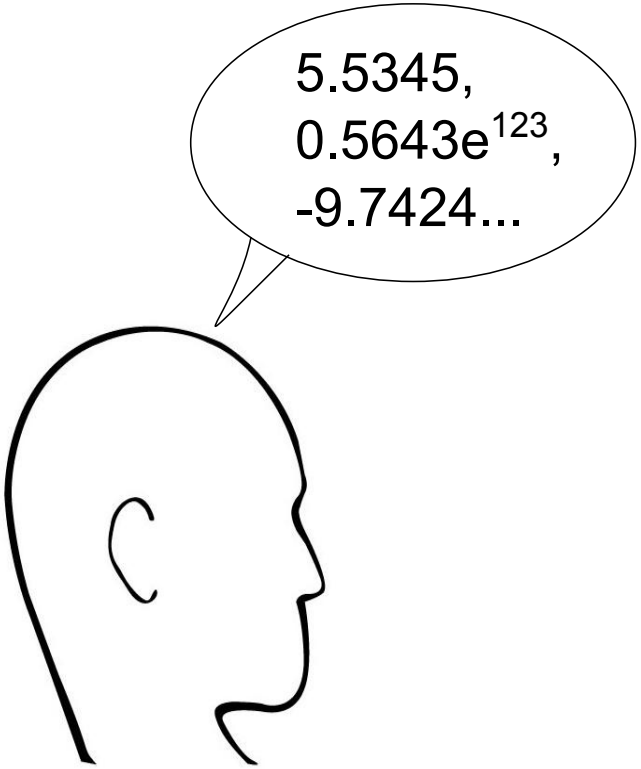
IF (I Study) AND NOT (Evil Teacher)
→ (I Will Pass)

Without Logic (eg. one-layer NN):

$W_{study,\ pass} = 0.989$;
$W_{evilTeacher,\ pass} = -0.966$

The World Without Logic (2)

With Logic



Without Logic

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

This Talk's Goals

- Formally introduce Inductive Logic Programming (ILP) and its theoretical foundations

This Talk's Goals

- Formally introduce Inductive Logic Programming (ILP) and its theoretical foundations
- Give an overall "feeling" of how it works

This Talk's Goals

- Formally introduce Inductive Logic Programming (ILP) and its theoretical foundations
- Give an overall "feeling" of how it works
- Briefly point out some alternative applications of ILP

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Remainder Of The Talk

- Background Knowledge, briefly explaining Propositional and First–Order Logics

Remainder Of The Talk

- Background Knowledge, briefly explaining Propositional and First–Order Logics
- Inductive Logic Programming, which will show the basic ILP concept

Remainder Of The Talk

- Background Knowledge, briefly explaining Propositional and First–Order Logics
- Inductive Logic Programming, which will show the basic ILP concept
- Some Relevant Systems that uses ILP, introducing the Connectionist and Inductive Learning and Logic Programming, C–IL2P

Remainder Of The Talk

- Background Knowledge, briefly explaining Propositional and First–Order Logics
- Inductive Logic Programming, which will show the basic ILP concept
- Some Relevant Systems that uses ILP, introducing the Connectionist and Inductive Learning and Logic Programming, C–IL2P
- Conclusion, to enclose everything that has been presented and add some final remarks

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Types of Reasoning

- Deductive Reasoning: given a background theory, **what is possible to be inferred from it?**

Types of Reasoning

- Deductive Reasoning: given a background theory, **what is possible to be inferred from it?**
- Inductive Reasoning: given a background theory and a set of examples, **what kinds of new theories can be inferred?**

Types of Reasoning

- Deductive Reasoning: given a background theory, **what is possible to be inferred from it?**
- Inductive Reasoning: given a background theory and a set of examples, **what kinds of new theories can be inferred?**
- Abductive Reasoning: given a background theory and a set of examples, **what kinds of facts can explain them?**

Types of Reasoning

- Deductive Reasoning: given a background theory, **what is possible to be inferred from it?**
- Inductive Reasoning: given a background theory and a set of examples, **what kinds of new theories can be inferred?**
- Abductive Reasoning: given a background theory and a set of examples, **what kinds of facts can explain them?**
- Deductive systems are clearly different from the other two, but inductive and abductive ones are somewhat similar

Types of Reasoning

- Deductive Reasoning: given a background theory, **what is possible to be inferred from it?**
- Inductive Reasoning: given a background theory and a set of examples, **what kinds of new theories can be inferred?**
- Abductive Reasoning: given a background theory and a set of examples, **what kinds of facts can explain them?**
- Deductive systems are clearly different from the other two, but inductive and abductive ones are somewhat similar
- In fact, under certain circumstances, an inductive task can be transformed into an abductive one and vice–versa

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:
  - $\wedge$ (and): $P \wedge Q$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:
  - $\wedge$ (and): $P \wedge Q$
  - () (parenthesis): $(P \wedge Q)$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:
  - $\wedge$ (and): $P \wedge Q$
  - () (parenthesis): $(P \wedge Q)$
  - $\vee$ (or): $R \vee (P \wedge Q)$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:
  - $\wedge$ (and): $P \wedge Q$
  - () (parenthesis): $(P \wedge Q)$
  - $\vee$ (or): $R \vee (P \wedge Q)$
  - $\rightarrow$ (implication): $(R \vee (P \wedge Q)) \rightarrow S$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:
  - $\wedge$ (and): $P \wedge Q$
  - () (parenthesis): $(P \wedge Q)$
  - $\vee$ (or): $R \vee (P \wedge Q)$
  - $\rightarrow$ (implication): $(R \vee (P \wedge Q)) \rightarrow S$
  - $\leftrightarrow$ (double–implication): $((R \vee (P \wedge Q)) \rightarrow S) \leftrightarrow T$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:
  - $\wedge$ (and): $P \wedge Q$
  - () (parenthesis): $(P \wedge Q)$
  - $\vee$ (or): $R \vee (P \wedge Q)$
  - $\rightarrow$ (implication): $(R \vee (P \wedge Q)) \rightarrow S$
  - $\leftrightarrow$ (double–implication): $((R \vee (P \wedge Q)) \rightarrow S) \leftrightarrow T$
- Clause: one or more literals connected through zero or more connectives, eg. $((R \vee (P \wedge Q)) \rightarrow S) \leftrightarrow T$

Syntax

- Atom: upper–case letter (P, Q, R, ...), $\perp$ or $\mathcal{T}$
- Logical Negation: $\neg$
- Literal: atom, preceded (**negative literal**) or not (**positive literal**) by $\neg$
- Connectives:
  - $\wedge$ (and): $P \wedge Q$
  - () (parenthesis): $(P \wedge Q)$
  - $\vee$ (or): $R \vee (P \wedge Q)$
  - $\rightarrow$ (implication): $(R \vee (P \wedge Q)) \rightarrow S$
  - $\leftrightarrow$ (double–implication): $((R \vee (P \wedge Q)) \rightarrow S) \leftrightarrow T$
- Clause: one or more literals connected through zero or more connectives, eg. $((R \vee (P \wedge Q)) \rightarrow S) \leftrightarrow T$
- Theory: set of one or more clauses, representing a knowledge domain

CITY UNIVERSITY LONDON

Semantics

- An atom can be *assigned* **true** or **false**

Semantics

- An atom can be *assigned* **true** or **false**
- Given a clause C, a clause interpretation for C consists of truth–value assignments for each of its atoms: I_C: $\mathrm{P}_1^C, \ldots, \mathrm{P}_l^C \mapsto \{true, false\}^{|l|}$

Semantics

- An atom can be *assigned* **true** or **false**
- Given a clause C, a clause interpretation for C consists of truth–value assignments for each of its atoms:
  I_C: $\mathrm{P}_1^C, \ldots, \mathrm{P}_l^C \mapsto \{true, false\}^{|l|}$
- Given a theory B = C_1, ... C_m, an interpretation for B is an assignment of truth values for each of its atoms:
  I_B: $\mathrm{P}_1^{C_1}, \ldots, \mathrm{P}_n^{C_m} \mapsto \{true, false\}^{|n|}$

Semantics

- An atom can be *assigned* **true** or **false**
- Given a clause C, a clause interpretation for C consists of truth–value assignments for each of its atoms:
  I_C: $\mathrm{P}_1^C, \ldots, \mathrm{P}_l^C \mapsto \{true, false\}^{|l|}$
- Given a theory B = C_1, ... C_m, an interpretation for B is an assignment of truth values for each of its atoms:
  I_B: $\mathrm{P}_1^{C_1}, \ldots, \mathrm{P}_n^{C_m} \mapsto \{true, false\}^{|n|}$
- Models are interpretations that assigns truth values to a given clause or theory:
  M(C) = I: I(C) $\mapsto$ *true* :: M(B) = I: I(B) $\mapsto$ *true*

Semantics

- An atom can be *assigned* **true** or **false**
- Given a clause C, a clause interpretation for C consists of truth–value assignments for each of its atoms:
  I_C: $\mathrm{P}_1^C, \ldots, \mathrm{P}_l^C \mapsto \{true, false\}^{|l|}$
- Given a theory B = C_1, ... C_m, an interpretation for B is an assignment of truth values for each of its atoms:
  I_B: $\mathrm{P}_1^{C_1}, \ldots, \mathrm{P}_n^{C_m} \mapsto \{true, false\}^{|n|}$
- Models are interpretations that assigns truth values to a given clause or theory:
  M(C) = I: I(C) $\mapsto$ *true* :: M(B) = I: I(B) $\mapsto$ *true*
- Interpretations are often represented in a *truth–table* format

Connectives Truth Table

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
F	F	V	F	F	V	V
F	V	V	F	V	V	F
V	F	F	F	V	F	F
V	V	F	V	V	V	V

- PS: All “v” symbols inside the table are *true* values

Logical Consequence

- A clause C is a logical consequence of a theory B if and only if $M(B) \subseteq M(C)$ ($B \models C$)

Logical Consequence

- A clause C is a logical consequence of a theory B if and only if $M(B) \subseteq M(C)$ ($B \models C$)
- Logical consequence is also known as entailment: B *entails* C if and only if $M(B) \subseteq M(C)$

Logical Consequence

- A clause C is a logical consequence of a theory B if and only if $M(B) \subseteq M(C)$ ($B \models C$)
- Logical consequence is also known as entailment: B *entails* C if and only if $M(B) \subseteq M(C)$
- C is satisfiable if $M(C) \neq \emptyset$

Logical Consequence

- A clause C is a logical consequence of a theory B if and only if $M(B) \subseteq M(C)$ ($B \models C$)
- Logical consequence is also known as entailment: B *entails* C if and only if $M(B) \subseteq M(C)$
- C is satisfiable if $M(C) \neq \emptyset$
- refutational consequence: $\mathbf{B} \models \mathbf{C}$ if and only if $B \cup \{\neg C\}$ is not satisfiable

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set
  - **R**: a set of inference rules

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set
  - **R**: a set of inference rules
- A proof of a clause C is a set of clauses DB, on the system DS ($DB \vdash C$), if and only if, exists a finite sequence of clauses ($D_1, \ldots, D_n$) which holds:

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set
  - **R**: a set of inference rules
- A proof of a clause C is a set of clauses DB, on the system DS ($DB \vdash C$), if and only if, exists a finite sequence of clauses ($D_1, \ldots, D_n$) which holds:
  1. $\mathbf{D_n = C}$

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set
  - **R**: a set of inference rules
- A proof of a clause C is a set of clauses DB, on the system DS ($DB \vdash C$), if and only if, exists a finite sequence of clauses ($D_1, \ldots, D_n$) which holds:
  1. $\mathbf{D_n = C}$
  2. For each $i \in [1, n]$, one of the following conditions is satisfied:

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set
  - **R**: a set of inference rules
- A proof of a clause C is a set of clauses DB, on the system DS (DB $\vdash$ C), if and only if, exists a finite sequence of clauses ($D_1, \ldots, D_n$) which holds:
  1. $\mathbf{D}_n = \mathbf{C}$
  2. For each $i \in [1, n]$, one of the following conditions is satisfied:
     - $\mathbf{D}_i$ is an instance of AX

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set
  - **R**: a set of inference rules
- A proof of a clause C is a set of clauses DB, on the system DS ($DB \vdash C$), if and only if, exists a finite sequence of clauses ($D_1, \ldots, D_n$) which holds:
  1. $\mathbf{D}_n = \mathbf{C}$
  2. For each $i \in [1, n]$, one of the following conditions is satisfied:
     - $\mathbf{D}_i$ is an instance of AX
     - $\mathbf{D}_i \in DB$

Deductive Systems

- A deductive system DS = (L, AX, R) is composed by:
  - **L**: used language
  - **AX**: logical axioms set
  - **R**: a set of inference rules
- A proof of a clause C is a set of clauses DB, on the system DS (DB $\vdash$ C), if and only if, exists a finite sequence of clauses ($D_1, \ldots, D_n$) which holds:
  1. $\mathbf{D}_n$ **= C**
  2. For each $i \in [1, n]$, one of the following conditions is satisfied:
     - $\mathbf{D}_i$ is an instance of AX
     - $\mathbf{D}_i \in$ DB
     - There exists $j, k < i$ in which D_i can be obtained by applying rules of R

Resolution

- Based on **refutational consequence**

Resolution

- Based on **refutational consequence**
- Definition:

Resolution

- Based on **refutational consequence**
- Definition:
  - L: propositional clauses in conjunctive normal form:
    $(R \vee (P \wedge Q)) \rightarrow S \Rightarrow (\neg R \vee S) \wedge (\neg P \vee S) \wedge (\neg Q \vee S)$

Resolution

- Based on **refutational consequence**
- Definition:
  - L: propositional clauses in conjunctive normal form: $(R \vee (P \wedge Q)) \rightarrow S \Rightarrow (\neg R \vee S) \wedge (\neg P \vee S) \wedge (\neg Q \vee S)$
  - AX: $\emptyset$

Resolution

- Based on **refutational consequence**
- Definition:
  - L: propositional clauses in conjunctive normal form: $(R \vee (P \wedge Q)) \rightarrow S \Rightarrow (\neg R \vee S) \wedge (\neg P \vee S) \wedge (\neg Q \vee S)$
  - AX: $\emptyset$
  - R: **propositional resolution** – $\{A \vee B; C \vee \neg B\} \Rightarrow \{A \vee C\}$

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Description

- Propositional Logic direct extension, to deal with predicates (relations)

Description

- Propositional Logic direct extension, to deal with predicates (relations)
- Able to conclude particular features of general characteristics over elements of a given domain

Description

- Propositional Logic direct extension, to deal with predicates (relations)
- Able to conclude particular features of general characteristics over elements of a given domain
- Able to conclude general features from particularities of single elements of a given domain

Description

- Propositional Logic direct extension, to deal with predicates (relations)
- Able to conclude particular features of general characteristics over elements of a given domain
- Able to conclude general features from particularities of single elements of a given domain
- To achieve that, two quantifiers were included into propositional logic: $\forall$ (**universal**) and $\exists$ (**existential**)

Syntax Modifications

- Terms are consisted of

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)
  - Constants $\Rightarrow$ facts, defined by *functions* of arity 0 (f)

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)
  - Constants $\Rightarrow$ facts, defined by *functions* of arity 0 (f)
  - Variables $\Rightarrow$ represented by upper–case letters (X)

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)
  - Constants $\Rightarrow$ facts, defined by *functions* of arity 0 (f)
  - Variables $\Rightarrow$ represented by upper–case letters (X)
- Predicates $\Rightarrow$ upper–case relation between terms ($P(X)$)

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)
  - Constants $\Rightarrow$ facts, defined by *functions* of arity 0 (f)
  - Variables $\Rightarrow$ represented by upper–case letters (X)
- Predicates $\Rightarrow$ upper–case relation between terms ($P(X)$)
- Quantifiers:

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)
  - Constants $\Rightarrow$ facts, defined by *functions* of arity 0 (f)
  - Variables $\Rightarrow$ represented by upper–case letters (X)
- Predicates $\Rightarrow$ upper–case relation between terms ($P(X)$)
- Quantifiers:
  - $\forall$ (universal): specifies features that are valid for every individual of the domain

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)
  - Constants $\Rightarrow$ facts, defined by *functions* of arity 0 (f)
  - Variables $\Rightarrow$ represented by upper–case letters (X)
- Predicates $\Rightarrow$ upper–case relation between terms ($P(X)$)
- Quantifiers:
  - $\forall$ (universal): specifies features that are valid for every individual of the domain
  - $\exists$ (existential): specifies features that are valid for at least one individual of the domain

Syntax Modifications

- Terms are consisted of
  - Functional symbols $\Rightarrow$ starts with lower–case and can have 0 or more terms ($f(X, a)$)
  - Constants $\Rightarrow$ facts, defined by *functions* of arity 0 (f)
  - Variables $\Rightarrow$ represented by upper–case letters (X)
- Predicates $\Rightarrow$ upper–case relation between terms ($P(X)$)
- Quantifiers:
  - $\forall$ (universal): specifies features that are valid for every individual of the domain
  - $\exists$ (existential): specifies features that are valid for at least one individual of the domain
- Clause: now can have variable quantified by one leftmost quantifier, eg. $\forall x((R(X) \vee (P(X) \wedge q)) \rightarrow s)$

CITY UNIVERSITY LONDON

Semantics Modifications

- An atom can be *assigned* one value of a given domain D of instantiations (an *assigned* atom is called grounded)

Semantics Modifications

- An atom can be *assigned* one value of a given domain D of instantiations (an *assigned* atom is called grounded)
- Given a clause C, a clause interpretation for C consists of assignments for each of its atoms

Semantics Modifications

- An atom can be *assigned* one value of a given domain D of instantiations (an *assigned* atom is called grounded)
- Given a clause C, a clause interpretation for C consists of assignments for each of its atoms
- Given a theory B = $\{C_1, \ldots C_m\}$, an interpretation for B is an assignment values for each of its atoms

Semantics Modifications

- An atom can be *assigned* one value of a given domain D of instantiations (an *assigned* atom is called grounded)
- Given a clause C, a clause interpretation for C consists of assignments for each of its atoms
- Given a theory B = $\{C_1, \ldots C_m\}$, an interpretation for B is an assignment values for each of its atoms
- Models are interpretations that assigns truth values to a given clause or theory

First–Order Resolution

- Each first–order clause passes through a standardization process called **Skolemization**

First–Order Resolution

- Each first–order clause passes through a standardization process called **Skolemization**
- Makes use of unifiers (substitution θ that makes $C_1\theta = C_2\theta$)

First–Order Resolution

- Each first–order clause passes through a standardization process called **Skolemization**
- Makes use of unifiers (substitution θ that makes $C_1\theta = C_2\theta$)
- First–Order propositional resolution rule is identical to the propositional one, but first unifies each term

First–Order Resolution

- Each first–order clause passes through a standardization process called **Skolemization**
- Makes use of unifiers (substitution θ that makes $C_1\theta = C_2\theta$)
- First–Order propositional resolution rule is identical to the propositional one, but first unifies each term
- Besides the proof, first–order resolution returns all used unifications

First–Order Resolution Step

P(A) v Q(A, claire) ¬ P(john) v R(abgail)

θ = {A\john}

Q(john, claire) v R(abgail)

First–Order Induction

- Inverse Resolution: “backwards” resolution, from the leaves to the root(s). Notable system: **Cigol**

First–Order Induction

- Inverse Resolution: “backwards” resolution, from the leaves to the root(s). Notable system: **Cigol**
- New inference rule (given a clause C_1 with an literal A, we want to find C_2 with $\neg$ A):
  $(Resolvent - (C_1 - \{A\})\theta_1)\theta_2^{-1} \cup \{\neg A\theta_1\theta_2^{-1}\}$

First–Order Induction

- Inverse Resolution: "backwards" resolution, from the leaves to the root(s). Notable system: **Cigol**
- New inference rule (given a clause C_1 with an literal A, we want to find C_2 with $\neg$ A):
  $(Resolvent - (C_1 - \{A\})\theta_1)\theta_2^{-1} \cup \{\neg A\theta_1\theta_2^{-1}\}$
- Immediate problem: whereas you are "going up" on inverse resolution, search space increases drastically

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Concepts

- Restriction to the first–order characterization to a more computationally convenient language

Concepts

- Restriction to the first–order characterization to a more computationally convenient language
- Definite clauses of the form $\forall_{x_1,\ldots,x_n}(h \vee b_1 \vee \ldots \vee b_n)$ becomes $\mathrm{h} \leftarrow \mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$

Concepts

- Restriction to the first–order characterization to a more computationally convenient language
- Definite clauses of the form $\forall_{x_1,\ldots,x_n}(h \vee b_1 \vee \ldots \vee b_n)$ becomes $\mathrm{h} \leftarrow \mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$
  - h: *head* of the clause

Concepts

- Restriction to the first–order characterization to a more computationally convenient language
- Definite clauses of the form $\forall_{x_1,\ldots,x_n}(h \vee b_1 \vee \ldots \vee b_n)$ becomes $\mathrm{h} \leftarrow \mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$
  - h: *head* of the clause
  - $\mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$: *bodies* of the clause, each b_i is called *condition*

Concepts

- Restriction to the first–order characterization to a more computationally convenient language
- Definite clauses of the form $\forall_{x_1,\ldots,x_n}(h \vee b_1 \vee \ldots \vee b_n)$ becomes $\mathrm{h} \leftarrow \mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$
  - h: *head* of the clause
  - $\mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$: *bodies* of the clause, each b_i is called *condition*
  - Fact: definite clause with empty body

Concepts

- Restriction to the first–order characterization to a more computationally convenient language
- Definite clauses of the form $\forall_{x_1,\ldots,x_n}(h \vee b_1 \vee \ldots \vee b_n)$ becomes $\mathrm{h} \leftarrow \mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$
  - h: *head* of the clause
  - $\mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$: *bodies* of the clause, each b_i is called *condition*
  - Fact: definite clause with empty body
  - Goal: clause with non–empty body but no head

Concepts

- Restriction to the first–order characterization to a more computationally convenient language
- Definite clauses of the form $\forall_{x_1,\ldots,x_n}(h \vee b_1 \vee \ldots \vee b_n)$ becomes $\mathrm{h} \leftarrow \mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$
  - h: *head* of the clause
  - $\mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$: *bodies* of the clause, each b_i is called *condition*
  - Fact: definite clause with empty body
  - Goal: clause with non–empty body but no head
  - Definite Program: conjunction of definite clauses

Concepts

- Restriction to the first–order characterization to a more computationally convenient language
- Definite clauses of the form $\forall_{x_1,\ldots,x_n}(h \vee b_1 \vee \ldots \vee b_n)$ becomes $\mathrm{h} \leftarrow \mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$
  - h: *head* of the clause
  - $\mathrm{b}_1 \vee \ldots \vee \mathrm{b}_n$: *bodies* of the clause, each b_i is called *condition*
  - Fact: definite clause with empty body
  - Goal: clause with non–empty body but no head
  - Definite Program: conjunction of definite clauses
- The semantical domain used for most logic programs is the Herbrand Universe

Prolog

- PROgramming in LOGic: one of the pioneers of Logic Programming

Prolog

- PROgramming in LOGic: one of the pioneers of Logic Programming
- Makes deduction through a variation of the classic resolution algorithm, called SLD–Resolution

Prolog

- PROgramming in LOGic: one of the pioneers of Logic Programming
- Makes deduction through a variation of the classic resolution algorithm, called SLD–Resolution
- SLD–Resolution differs from classical resolution by defining which clauses will be resolved

Monotonic and Nonmonotonic Logic Programming

- Logical entailment is monotonic (additions into a theory never decreases the amount of possible consequences)

Monotonic and Nonmonotonic Logic Programming

- Logical entailment is monotonic (additions into a theory never decreases the amount of possible consequences)
- An way to add nonmotonicity into a logic program is to use Closed–World Assumption (CWA):
  "If an literal of a goal is not proved by the current theory, then it is false"

Monotonic and Nonmonotonic Logic Programming

- Logical entailment is monotonic (additions into a theory never decreases the amount of possible consequences)
- An way to add nonmotonicity into a logic program is to use Closed–World Assumption (CWA):
  "If an literal of a goal is not proved by the current theory, then it is false"
- This rule is called Negation as Failure

Monotonic and Nonmonotonic Logic Programming

- Logical entailment is monotonic (additions into a theory never decreases the amount of possible consequences)
- An way to add nonmotonicity into a logic program is to use Closed–World Assumption (CWA):
  "If an literal of a goal is not proved by the current theory, then it is false"
- This rule is called Negation as Failure
- An extension to resolution called SLD–NF has been created to deal with this kind of deduction

Monotonic and Nonmonotonic Logic Programming

- Logical entailment is monotonic (additions into a theory never decreases the amount of possible consequences)
- An way to add nonmotonicity into a logic program is to use Closed–World Assumption (CWA):
  "If an literal of a goal is not proved by the current theory, then it is false"
- This rule is called Negation as Failure
- An extension to resolution called SLD–NF has been created to deal with this kind of deduction
- CWA causes nonmotonicity

CWA Example

- Consider the following theory:

$$\mathbf{B} = \begin{cases} \text{Trip(brazil, airplane)} \\ \text{Trip(cambridge, train)} \\ \text{Trip(leeds, car)} \end{cases}$$

CWA Example

- Consider the following theory:

$$\mathbf{B} = \begin{cases} \text{Trip(brazil, airplane)} \\ \text{Trip(cambridge, train)} \\ \text{Trip(leeds, car)} \end{cases}$$

- If a goal literal *Trip(cambridge, train)* is queried with regard to B, a positive answer will be given

CWA Example

- Consider the following theory:

$$\mathbf{B} = \begin{cases} \text{Trip(brazil, airplane)} \\ \text{Trip(cambridge, train)} \\ \text{Trip(leeds, car)} \end{cases}$$

- If a goal literal *Trip(cambridge, train)* is queried with regard to B, a positive answer will be given
- Otherwise, if it is asked for *Trip(paris, airplane)*, two answers can be obtained, depending if CWA is being used

CWA Example

- Consider the following theory:

$$\mathbf{B} = \begin{cases} \text{Trip(brazil, airplane)} \\ \text{Trip(cambridge, train)} \\ \text{Trip(leeds, car)} \end{cases}$$

- If a goal literal *Trip(cambridge, train)* is queried with regard to B, a positive answer will be given
- Otherwise, if it is asked for *Trip(paris, airplane)*, two answers can be obtained, depending if CWA is being used
  - If it is not being used nothing will be answered (in Prolog, a “fail” message would be shown)

CWA Example

- Consider the following theory:

$$\mathbf{B} = \begin{cases} \text{Trip(brazil, airplane)} \\ \text{Trip(cambridge, train)} \\ \text{Trip(leeds, car)} \end{cases}$$

- If a goal literal *Trip(cambridge, train)* is queried with regard to B, a positive answer will be given
- Otherwise, if it is asked for *Trip(paris, airplane)*, two answers can be obtained, depending if CWA is being used
  - If it is not being used nothing will be answered (in Prolog, a “fail” message would be shown)
  - If it is used, this query would return **false**

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning
- Which means: given a set of labeled examples E and a background knowledge B, an ILP system will try to find a hypothesis function H that minimizes a specified loss($B \cup H$, E)

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning
- Which means: given a set of labeled examples E and a background knowledge B, an ILP system will try to find a hypothesis function H that minimizes a specified loss($B \cup H$, E)
- In ILP context:

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning
- Which means: given a set of labeled examples E and a background knowledge B, an ILP system will try to find a hypothesis function H that minimizes a specified loss($B \cup H$, E)
- In ILP context:
  - B: pre–existent definite program

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning
- Which means: given a set of labeled examples E and a background knowledge B, an ILP system will try to find a hypothesis function H that minimizes a specified loss($B \cup H$, E)
- In ILP context:
  - B: pre–existent definite program
  - E: set of grounded atoms of target concept(s), in which labels are truth–values

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning
- Which means: given a set of labeled examples E and a background knowledge B, an ILP system will try to find a hypothesis function H that minimizes a specified loss($B \cup H$, E)
- In ILP context:
  - B: pre–existent definite program
  - E: set of grounded atoms of target concept(s), in which labels are truth–values
  - H: target definite program that entails most examples of E

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning
- Which means: given a set of labeled examples E and a background knowledge B, an ILP system will try to find a hypothesis function H that minimizes a specified loss($B \cup H$, E)
- In ILP context:
  - B: pre–existent definite program
  - E: set of grounded atoms of target concept(s), in which labels are truth–values
  - H: target definite program that entails most examples of E
  - loss(H, E): f(number of examples entailed by $B \cup U$)

Introduction

- Inductive Logic Programming (ILP) is a machine learning technique that conducts supervised inductive concept learning
- Which means: given a set of labeled examples E and a background knowledge B, an ILP system will try to find a hypothesis function H that minimizes a specified loss($B \cup H$, E)
- In ILP context:
  - B: pre–existent definite program
  - E: set of grounded atoms of target concept(s), in which labels are truth–values
  - H: target definite program that entails most examples of E
  - loss(H, E): f(number of examples entailed by $B \cup U$)
- The obtained H not only classifies new examples but can improve an existing theory

CITY UNIVERSITY LONDON

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:
  - E: set of positive and negative literals (examples)

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:
  - E: set of positive and negative literals (examples)
  - B: logic program that defines the background knowledge underlying the task

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:
  - E: set of positive and negative literals (examples)
  - B: logic program that defines the background knowledge underlying the task
  - L: set of logic theories that restricts the search space to find a suitable hypothesis (language bias)

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:
  - E: set of positive and negative literals (examples)
  - B: logic program that defines the background knowledge underlying the task
  - L: set of logic theories that restricts the search space to find a suitable hypothesis (language bias)
- If *learning from interpretation*, it is also defined as <E, B, L>, where:

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:
  - E: set of positive and negative literals (examples)
  - B: logic program that defines the background knowledge underlying the task
  - L: set of logic theories that restricts the search space to find a suitable hypothesis (language bias)
- If *learning from interpretation*, it is also defined as <E, B, L>, where:
  - E: set of Herbrand Interpretations of B, labeled positive and negative as well

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:
  - E: set of positive and negative literals (examples)
  - B: logic program that defines the background knowledge underlying the task
  - L: set of logic theories that restricts the search space to find a suitable hypothesis (language bias)
- If *learning from interpretation*, it is also defined as <E, B, L>, where:
  - E: set of Herbrand Interpretations of B, labeled positive and negative as well
  - B: logic program that defines the background knowledge underlying the task

Task Formalization

- An ILP task can be defined in two ways, depending on the type of learning
- If *learning from entailment*, it is defined as a tuple <E, B, L>, where:
  - E: set of positive and negative literals (examples)
  - B: logic program that defines the background knowledge underlying the task
  - L: set of logic theories that restricts the search space to find a suitable hypothesis (language bias)
- If *learning from interpretation*, it is also defined as <E, B, L>, where:
  - E: set of Herbrand Interpretations of B, labeled positive and negative as well
  - B: logic program that defines the background knowledge underlying the task
  - L: language bias

CITY UNIVERSITY LONDON

Hypothesis Search Types

- ILP systems can build hypothesis following two directions:

Hypothesis Search Types

- ILP systems can build hypothesis following two directions:
  - Bottom–Up: starting with the most specific clause ($\perp$), generalizations can be made to make it cover the positive examples while keeping most negative ones out

Hypothesis Search Types

- ILP systems can build hypothesis following two directions:
  - Bottom–Up: starting with the most specific clause ($\perp$), generalizations can be made to make it cover the positive examples while keeping most negative ones out
  - Top–Down: starting with the most general clause ($h \leftarrow$), specializations can be made to eliminate negative examples from the coverage set while maintaining positive ones covered

ILP General Algorithm

- Most ILP systems are based in a Sequential–Covering algorithm:

Algorithm 1 Sequential–Covering Algorithm

Require: E, B, L
Ensure: H

```
E_cur = E
H = ∅
while generalization stopping criterion is satisfied do
    c = h ←
    while specialization stopping criterion is satisfied do
        c = REFINE(c, L)
    end while
    H = H ∪ c
    E_cov = e ∈ E_cur : B ∪ H ⊨ e
    E_cur = E_cur − E_cov
end while
```

ILP General Algorithm

- Most ILP systems are based in a Sequential–Covering algorithm:

Algorithm 1 Sequential–Covering Algorithm

Require: E, B, L
Ensure: H

```
E_cur = E
H = ∅
while generalization stopping criterion is satisfied do
    c = h ←
    while specialization stopping criterion is satisfied do
        c = REFINE(c, L)
    end while
    H = H ∪ c
    E_cov = e ∈ E_cur : B ∪ H ⊨ e
    E_cur = E_cur − E_cov
end while
```

- REFINE is a function that varies between ILP systems and chooses or removes bodies from a candidate hypothesis

Quick Note

- ILP systems relies on specialization and generalization operators to build hypothesis

Quick Note

- ILP systems relies on specialization and generalization operators to build hypothesis
- The way they work and are used depends on the ILP system being used, as well as the language bias L

Quick Note

- ILP systems relies on specialization and generalization operators to build hypothesis
- The way they work and are used depends on the ILP system being used, as well as the language bias L
- For convenience, *Progol* has been chosen as the system which will illustrate how these dependencies works

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

A Brief Summary of Progol

- Progol is an ILP system and a Machine Learning algorithm, based on *Inverse Entailment* and *Sequential Covering*, which searches for suited hypothesis in a search space bounded by the most general clause (h $\leftarrow$) and a *Bottom–Clause* ($\perp$)

A Brief Summary of Progol

- Progol is an ILP system and a Machine Learning algorithm, based on *Inverse Entailment* and *Sequential Covering*, which searches for suited hypothesis in a search space bounded by the most general clause ($h \leftarrow$) and a *Bottom–Clause* ($\perp$)
  - Inverse Entailment: $B \cup H \models E \Rightarrow B \models H \rightarrow E \Rightarrow B \models \neg E \rightarrow \neg H \Rightarrow B \cup \neg E \models \neg H$ ($H \models \perp$)

A Brief Summary of Progol

- Progol is an ILP system and a Machine Learning algorithm, based on *Inverse Entailment* and *Sequential Covering*, which searches for suited hypothesis in a search space bounded by the most general clause ($h \leftarrow$) and a *Bottom–Clause* ($\perp$)
  - Inverse Entailment: $B \cup H \models E \Rightarrow B \models H \rightarrow E \Rightarrow B \models \neg E \rightarrow \neg H \Rightarrow B \cup \neg E \models \neg H$ ($H \models \perp$)
  - Sequential Covering: the search for an hypothesis starts from ($h \leftarrow$) and adds iteratively literals that covers the positive examples and do not covers most of the negative ones

A Brief Summary of Progol

- Progol is an ILP system and a Machine Learning algorithm, based on *Inverse Entailment* and *Sequential Covering*, which searches for suited hypothesis in a search space bounded by the most general clause ($h \leftarrow$) and a *Bottom–Clause* ($\perp$)
  - Inverse Entailment: $B \cup H \models E \Rightarrow B \models H \rightarrow E \Rightarrow B \models \neg E \rightarrow \neg H \Rightarrow B \cup \neg E \models \neg H$ ($H \models \perp$)
  - Sequential Covering: the search for an hypothesis starts from ($h \leftarrow$) and adds iteratively literals that covers the positive examples and do not covers most of the negative ones
  - Bottom–Clause: most–specific clause of a restricted space, defined by a single example and L

A Brief Summary of Progol

- Progol is an ILP system and a Machine Learning algorithm, based on *Inverse Entailment* and *Sequential Covering*, which searches for suited hypothesis in a search space bounded by the most general clause (h $\leftarrow$) and a *Bottom–Clause* ($\perp$)
  - Inverse Entailment: $B \cup H \models E \Rightarrow B \models H \rightarrow E \Rightarrow B \models \neg E \rightarrow \neg H \Rightarrow B \cup \neg E \models \neg H$ ($H \models \perp$)
  - Sequential Covering: the search for an hypothesis starts from (h $\leftarrow$) and adds iteratively literals that covers the positive examples and do not covers most of the negative ones
  - Bottom–Clause: most–specific clause of a restricted space, defined by a single example and L
- It uses the following loss function: $loss^{progol}(E, B, H) = + \sum_{r \in H} |r| - |\{e \in E^+ : B \cup H \models e\}| + |\{e' \in E^- : B \cup H \models e'\}|$

Language Bias Structure

- Progol has two language bias structures:

Language Bias Structure

- Progol has two language bias structures:
  - Mode Declarations: can be *head declarations* – modeh(recall, s), or body declarations – modeb(recall, s), where *recall* controls the number of instantiations of a literal and *s* is a ground positive or negative literal, with *placemarkers* which defines if it is an input (+), output (–) or a constant (#), and its type

Language Bias Structure

- Progol has two language bias structures:
  - Mode Declarations: can be *head declarations* – modeh(recall, s), or body declarations – modeb(recall, s), where *recall* controls the number of instantiations of a literal and *s* is a ground positive or negative literal, with *placemarkers* which defines if it is an input (+), output (–) or a constant (#), and its type
  - Determinations: used to specify which bodies can be used to build a clause with a given head

Language Bias Structure

- Progol has two language bias structures:
  - Mode Declarations: can be *head declarations* – modeh(recall, s), or body declarations – modeb(recall, s), where *recall* controls the number of instantiations of a literal and *s* is a ground positive or negative literal, with *placemarkers* which defines if it is an input (+), output (–) or a constant (#), and its type
  - Determinations: used to specify which bodies can be used to build a clause with a given head
  - Examples:
    $modeh(1, mother\_in\_law(+woman, -man))$
    $modeb(*, progenitor\_of(+woman, -woman))$
    $modeb(1, wife\_of(+woman, -man))$
    $determination(mother\_in\_law/2, progenitor\_of/2)$
    $determination(mother\_in\_law/2, wife\_of/2)$

Specialization and Generalization Operators

- Uses $\theta - subsumption$: a clause C $\theta - subsumes$ D (C $\prec_\theta$ D) if exists a substitution θ in which $C\theta \subseteq D$ holds, eg.:
  C: f(A, B) $\leftarrow$ p(B, G), q(G, A) $\theta - subsumes$
  D: f(a, b) $\leftarrow$ p(b, g), q(g, a), t(a, d) through θ = {A/a, B/b, G/g}

Specialization and Generalization Operators

- Uses $\theta - subsumption$: a clause C $\theta - subsumes$ D (C $\prec_\theta$ D) if exists a substitution θ in which $C\theta \subseteq D$ holds, eg.:
  C: f(A, B) $\leftarrow$ p(B, G), q(G, A) $\theta - subsumes$
  D: f(a, b) $\leftarrow$ p(b, g), q(g, a), t(a, d) through θ = {A/a, B/b, G/g}
- If C $\prec_\theta$ D, then C $\models$ D

Specialization and Generalization Operators

- Uses $\theta - subsumption$: a clause C $\theta - subsumes$ D (C $\prec_\theta$ D) if exists a substitution θ in which $C\theta \subseteq D$ holds, eg.:
  C: f(A, B) $\leftarrow$ p(B, G), q(G, A) $\theta - subsumes$
  D: f(a, b) $\leftarrow$ p(b, g), q(g, a), t(a, d) through θ = {A/a, B/b, G/g}
- If C $\prec_\theta$ D, then C $\models$ D
- This means that for a specialization or generalization of a clause C, C $\prec_\theta \perp$ ensures that the search space bounds still holds

Variable Chaining

- In Progol, every body input variable needs to be an input of a head literal or an output of a previous body literal

Variable Chaining

- In Progol, every body input variable needs to be an input of a head literal or an output of a previous body literal
- An example:

```
modeh(*, mult(+real, +real,-real))
modeb(*, dec(+real,-real))
modeb(*, plus(+real, +real,-real))
determination(mult/3, mult/3)
determination(mult/3, dec/2)
determination(mult/3, plus/3)
```

C = mult(E,F,G) $\leftarrow$ dec(E, H), mult(F, H, I), plus(F, I, G).

Bottom–Clause

- In Progol, the search space for a candidate hypothesis starts from (h $\leftarrow$) and specializations are added in order to minimize its loss function.

Bottom–Clause

- In Progol, the search space for a candidate hypothesis starts from (h $\leftarrow$) and specializations are added in order to minimize its loss function.
- This search is limited by a special *bottom–clause*

Bottom–Clause

- In Progol, the search space for a candidate hypothesis starts from (h $\leftarrow$) and specializations are added in order to minimize its loss function.
- This search is limited by a special *bottom–clause*
- It is built from a chosen example by applying iteratively substitutions to match all possible *modeb* literals, in top–down order, according to its *modeh* definition, background knowledge and *determination* restrictions

Bottom–Clause

- In Progol, the search space for a candidate hypothesis starts from (h $\leftarrow$) and specializations are added in order to minimize its loss function.
- This search is limited by a special *bottom–clause*
- It is built from a chosen example by applying iteratively substitutions to match all possible *modeb* literals, in top–down order, according to its *modeh* definition, background knowledge and *determination* restrictions
- It uses a deepness (d) input parameter, which controls the amount of cycles through *modeb* literals

Bottom–Clause

- In Progol, the search space for a candidate hypothesis starts from (h $\leftarrow$) and specializations are added in order to minimize its loss function.
- This search is limited by a special *bottom–clause*
- It is built from a chosen example by applying iteratively substitutions to match all possible *modeb* literals, in top–down order, according to its *modeh* definition, background knowledge and *determination* restrictions
- It uses a deepness (d) input parameter, which controls the amount of cycles through *modeb* literals
- After built, it is then variabilized to be used as a specialization boundary

Example

Input: example: **mult(2, 4, 8)**, maxDeepness = 1

Modes:
modeh(*, mult(+real, +real, -real))
modeb(*, dec(+real, -real))
modeb(*, plus(+real, +real, -real))

Background Knowledge:
dec(2, 4); dec(2, 5); plus(2, 2, 4); plus(2, 4, 6);

Possible terms to use: $\emptyset$
Other terms: $\emptyset$

$\perp = \emptyset$
Current deepness: 0

Example

Input: example: mult(2, 4, 8), maxDeepness = 1

Modes:
modeh(*, mult(+real, +real, -real))
modeb(*, dec(+real, -real))
modeb(*, plus(+real, +real, -real))

Background Knowledge:
dec(2, 4); dec(2, 5); plus(2, 2, 4); plus(2, 4, 6);

Possible terms to use: A(2), B(4)
Other terms: C(8)

$\perp$ = mult(A, B, C) $\leftarrow$
Current deepness: 0

Example

Input: example: mult(2, 4, 8), maxDeepness = 1

Modes:
modeh(*, mult(+real, +real, -real))
modeb(*, dec(+real, -real))
modeb(*, plus(+real, +real, -real))

Background Knowledge:
dec(2, 4); **dec(2, 5)**; plus(2, 2, 4); plus(2, 4, 6);

Possible terms to use: A(2), B(4), D(5)
Other terms: C(8)

$\perp$ = mult(A, B, C) $\leftarrow$ dec(A, B), dec(A, D)
Current deepness: 0

Example

Input: example: mult(2, 4, 8), maxDeepness = 1

Modes:
modeh(*, mult(+real, +real, -real))
modeb(*, dec(+real, -real))
modeb(*, plus(+real, +real, -real))

Background Knowledge:
dec(2, 4); dec(2, 5); **plus(2, 2, 4)**; plus(2, 4, 6);

Possible terms to use: A(2), B(4), D(5)
Other terms: C(8)

$\perp$ = mult(A, B, C) $\leftarrow$ dec(A, B), dec(A, D), plus(A, A, B)
Current deepness: 0

Example

Input: example: mult(2, 4, 8), maxDeepness = 1

Modes:
modeh(*, mult(+real, +real, -real))
modeb(*, dec(+real, -real))
modeb(*, plus(+real, +real, -real))

Background Knowledge:
dec(2, 4); dec(2, 5); plus(2, 2, 4); **plus(2, 4, 6)**;

Possible terms to use: A(2), B(4), D(5), E(6)
Output terms: C(8)

$\perp$ = mult(A, B, C) $\leftarrow$ dec(A, B), dec(A, D), plus(A, A, B), plus(A, B, E)
Current deepness: 1 (algorithm stop)

CITY UNIVERSITY LONDON

Specialization Operator

- An order on $\perp$ is assumed and let $\perp(k)$ be the k–th element in $\perp$

Specialization Operator

- An order on $\perp$ is assumed and let $\perp$(k) be the k–th element in $\perp$
- Since H $\prec_\theta \perp$, there must be a substitution θ such that for each literal *h* in H, there is a literal *l* such that $h\theta = l$

Specialization Operator

- An order on $\perp$ is assumed and let $\perp$(k) be the k–th element in $\perp$
- Since H $\prec_\theta \perp$, there must be a substitution θ such that for each literal *h* in H, there is a literal *l* such that $h\theta = l$
- The substitution operator δ can be defined as:

$< P(v_1, \ldots, v_m, \theta_m) > \in \delta(\theta, k)$ if and only if
– $P(u_1, \ldots, u_m)$ is the k–th literal in $\perp$
– $\theta_0 = \theta$
– if $v_j / u_j \in \theta_{j-1}$ then $\theta_j = \theta_{j-1}$ for $0 < j \leq m$

Specialization Operator

- An order on $\perp$ is assumed and let $\perp$(k) be the k–th element in $\perp$
- Since H $\prec_\theta \perp$, there must be a substitution θ such that for each literal *h* in H, there is a literal *l* such that $h\theta = l$
- The substitution operator δ can be defined as:

  $< P(v_1, \ldots, v_m, \theta_m) > \in \delta(\theta, k)$ if and only if
  – $P(u_1, \ldots, u_m)$ is the k–th literal in $\perp$
  – $\theta_0 = \theta$
  – if $v_j/u_j \in \theta_{j-1}$ then $\theta_j = \theta_{j-1}$ for $0 < j \leq m$
- The specialization operator ρ can be defined as:

  $< r', \theta', k' > \in \rho(< r, \theta, k >)$ if and only if either
  – $r' = c \cup \{l\}, k' = k, < l, \theta' > \in \delta(\theta, k), r' \in \{Rulespace\}$ or
  – $r' = c, k' = k + 1, \theta' = \theta$

  for $1 \leq k \leq |\perp|$

Specialization Choosing

- From all possible substitutions using ρ, the one that minimizes $loss^{progol}(E, B, H)$. The search method used is A*, which tracks the best path regarding $loss^{progol}$ with a priority list for other options:

mult(A, B, C) ←

mult(A, B, C) ← dec(A, B) mult(A, B, C) ← dec(A, C)

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Neural–Symbolic Integration

- Symbolical systems, such as Progol, holds very powerful representative power through First–Order logics briefly explaining Propositional and First–Order Logics

Neural–Symbolic Integration

- Symbolical systems, such as Progol, holds very powerful representative power through First–Order logics briefly explaining Propositional and First–Order Logics
- Connectionistic systems, being neural networks its main representative, have great noise–robustness capabilities and can model knowledge as probabilities through their weights

Neural–Symbolic Integration

- Symbolical systems, such as Progol, holds very powerful representative power through First–Order logics briefly explaining Propositional and First–Order Logics
- Connectionistic systems, being neural networks its main representative, have great noise–robustness capabilities and can model knowledge as probabilities through their weights
- As seen during introduction, each one of those paradigms holds advantages and advantages, in a quite “complementary” way

Neural–Symbolic Integration

- Symbolical systems, such as Progol, holds very powerful representative power through First–Order logics briefly explaining Propositional and First–Order Logics
- Connectionistic systems, being neural networks its main representative, have great noise–robustness capabilities and can model knowledge as probabilities through their weights
- As seen during introduction, each one of those paradigms holds advantages and advantages, in a quite “complementary” way
- Why not combine both paradigms?

Neural–Symbolic Integration

	Symbolical	Connectionistic
Multiple Learning	Slow learning of multiple concepts	**Efficient parallel learning of multiple concepts**
Noise Robustness	Limited, artificial noise–robustness capabilities	**Natural, method–inherent noise treatment**
Concept Clarity	**Learned concepts formally represented**	Concepts are tangled inside numerical data
Background Data	**Makes partial or total use of background data**	Only empirical examples are used

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Idea

- Union two of the most popular symbolic and connectionistic representatives: ILP and Neural Networks

Idea

- Union two of the most popular symbolic and connectionistic representatives: ILP and Neural Networks
- Enhance their advantages, while supressing their flaws

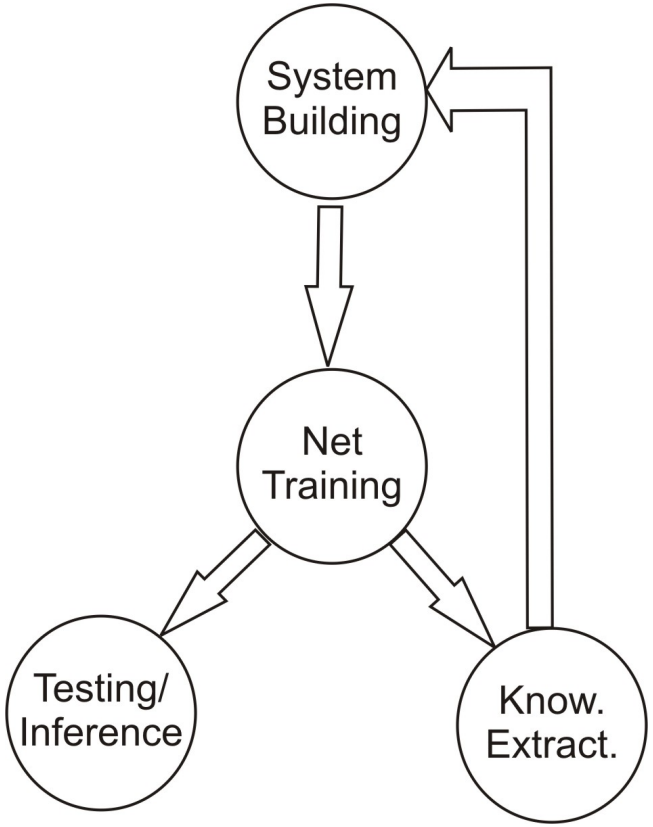
Idea

- Union two of the most popular symbolic and connectionistic representatives: ILP and Neural Networks
- Enhance their advantages, while supressing their flaws
- A relational (recursive) three–layer network is built from a propositional logic program and it is then used to train examples

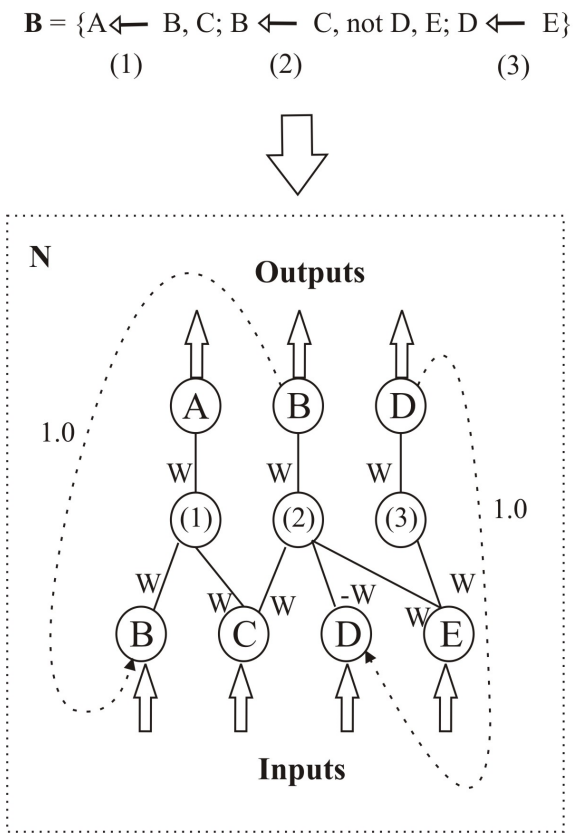
Idea

- Union two of the most popular symbolic and connectionistic representatives: ILP and Neural Networks
- Enhance their advantages, while supressing their flaws
- A relational (recursive) three–layer network is built from a propositional logic program and it is then used to train examples
- This way, a knowledge–based neural network is created (which solves the main problem of classical neural networks) which is capable of using “almost” full capabilities of back–propagation training regarding noise robustness and incomplete data (which are the two main ILP weakpoints)

C–IL2P Knowledge Flow



C–IL2P Structure



Applications

- As expected of a hybrid system, C–IL2P can be used in any problem in which any one of its components would be eligible to be applied.

Applications

- As expected of a hybrid system, C–IL2P can be used in any problem in which any one of its components would be eligible to be applied.
- Additionally (as the knowledge flow has shown before), the way C–IL2P deals with information processing can allow it to be applied in some unique applications, such as fault diagnosis and multi–instance learning problems

Applications

- As expected of a hybrid system, C–IL2P can be used in any problem in which any one of its components would be eligible to be applied.
- Additionally (as the knowledge flow has shown before), the way C–IL2P deals with information processing can allow it to be applied in some unique applications, such as fault diagnosis and multi–instance learning problems
- If extended to work with First–Order logic programs, its applicability would be hugely enhanced

Outline

1. Introduction
   - Motivation
   - Objectives
   - Overview
2. Background Knowledge
   - Propositional (Classical) Logic
   - First–Order Logic
   - Logic Programming
3. Inductive Logic Programming
   - Definitions
   - Progol
4. Neural–Symbolic Systems
   - Introduction
   - C–IL2P
   - CILP++
5. Conclusion

CITY UNIVERSITY LONDON

Main Concepts

- CILP++ is the results of my work on C–IL2P to allow it to work with First–Order logics

Main Concepts

- CILP++ is the results of my work on C–IL2P to allow it to work with First–Order logics
- Different ways of using First–Order logics are being studied:

Main Concepts

- CILP++ is the results of my work on C–IL2P to allow it to work with First–Order logics
- Different ways of using First–Order logics are being studied:
  - Using Bottom–Clauses as examples;

Main Concepts

- CILP++ is the results of my work on C–IL2P to allow it to work with First–Order logics
- Different ways of using First–Order logics are being studied:
  - Using Bottom–Clauses as examples;
  - Different kinds of propositionalizations;

Main Concepts

- CILP++ is the results of my work on C–IL2P to allow it to work with First–Order logics
- Different ways of using First–Order logics are being studied:
  - Using Bottom–Clauses as examples;
  - Different kinds of propositionalizations;
  - Applying the language bias on the building step of C–IL2P

Main Concepts

- CILP++ is the results of my work on C–IL2P to allow it to work with First–Order logics
- Different ways of using First–Order logics are being studied:
  - Using Bottom–Clauses as examples;
  - Different kinds of propositionalizations;
  - Applying the language bias on the building step of C–IL2P
- It uses the same building process of C–IL2P, but differs in the network training, depending on the kind of First–Order information that is being given to it

What Has Been Done

- Enhancements in the underlying neural network to optimize it

What Has Been Done

- Enhancements in the underlying neural network to optimize it
- Testings using *Bottom–Clauses* and RSA propositionalization

What Has Been Done

- Enhancements in the underlying neural network to optimize it
- Testings using *Bottom–Clauses* and RSA propositionalization
- A friendly GUI using *wxWidgets* to let other people uses the basic C–IL2P functionality (an open–source C–IL2P project is already active at SourceForge: http://sourceforge.net/projects/cil2p/)

What Will Be Done

In priority order:

- Fine–tuning the system to work with bottom–clauses and propositionalizated datasets

What Will Be Done

In priority order:

- Fine–tuning the system to work with bottom–clauses and propositionalizated datasets
- Study how knowledge extraction will take place in this new system

What Will Be Done

In priority order:

- Fine–tuning the system to work with bottom–clauses and propositionalizated datasets
- Study how knowledge extraction will take place in this new system
- Test other ways of using First–Order logics into C–IL2P without major structural changes

What Will Be Done

In priority order:

- Fine–tuning the system to work with bottom–clauses and propositionalizated datasets
- Study how knowledge extraction will take place in this new system
- Test other ways of using First–Order logics into C–IL2P without major structural changes
- Analyze structural changes on CILP++ to allow better suitability for First–Order logic processing

Applications

- CILP++, working with First–Order logics, will be able to completely explore domain–theory and classification problems. It will be the first hybrid system to achieve that

Applications

- CILP++, working with First–Order logics, will be able to completely explore domain–theory and classification problems. It will be the first hybrid system to achieve that
- Specifically about bottom–clauses usage, as it is relatively simple, it will allow quick training and online inference of First–Order logics, which can make way to some interesting applications:

Applications

- CILP++, working with First–Order logics, will be able to completely explore domain–theory and classification problems. It will be the first hybrid system to achieve that
- Specifically about bottom–clauses usage, as it is relatively simple, it will allow quick training and online inference of First–Order logics, which can make way to some interesting applications:
  - Web–semantics

Applications

- CILP++, working with First–Order logics, will be able to completely explore domain–theory and classification problems. It will be the first hybrid system to achieve that
- Specifically about bottom–clauses usage, as it is relatively simple, it will allow quick training and online inference of First–Order logics, which can make way to some interesting applications:
  - Web–semantics
  - Intelligent Agents

Final Remarks

- Connectionism and symbolism are two paradigms that were kept separated for a long time, but this is coming to an end

Final Remarks

- Connectionism and symbolism are two paradigms that were kept separated for a long time, but this is coming to an end
- Symbolism, supported by its very successful machine learning algorithm called ILP, is a necessary tool to express knowledge in a formal and clear way and to represent complex and hierarchical relations

Final Remarks

- Connectionism and symbolism are two paradigms that were kept separated for a long time, but this is coming to an end
- Symbolism, supported by its very successful machine learning algorithm called ILP, is a necessary tool to express knowledge in a formal and clear way and to represent complex and hierarchical relations
- Neural–Symbolic Integration is one way–to–go in going one step further in learning, reasoning and expressing knowledge

Recommended Reading

- Tom M. Mitchell. **Machine Learning.** McGraw−Hill, New York, 1997

Recommended Reading

- Tom M. Mitchell. **Machine Learning.** McGraw−Hill, New York, 1997
- Stuart J. Russell and Peter Norvig. **Artificial Intelligence − A Modern Approach** (3. internat. ed.). Pearson Education, 2010

Recommended Reading

- Tom M. Mitchell. **Machine Learning.** McGraw−Hill, New York, 1997
- Stuart J. Russell and Peter Norvig. **Artificial Intelligence − A Modern Approach** (3. internat. ed.). Pearson Education, 2010
- Artur S. D. Garcez and Luis C. Lamb and Dov M. Gabbay. **Neural-Symbolic Cognitive Reasoning**. Springer−Verlag, 2009

Recommended Reading

- Tom M. Mitchell. **Machine Learning.** McGraw−Hill, New York, 1997
- Stuart J. Russell and Peter Norvig. **Artificial Intelligence − A Modern Approach** (3. internat. ed.). Pearson Education, 2010
- Artur S. D. Garcez and Luis C. Lamb and Dov M. Gabbay. **Neural-Symbolic Cognitive Reasoning**. Springer−Verlag, 2009
- Petri Toiviainen. **Symbolic AI Versus Connectionism in Music Research**. *Readings in Music and Artificial Intelligence*. Amsterdam: Harwood Academic Publishers, 2000

Thank You!

Manoel.Franca.1@city.ac.uk