Survey: Logic, Logic Programming, Ontology, Rules

Ontolog Mini-Series: Ontology, Rules, and Logic Programming for Reasoning and Applications (RulesReasoningLP)

Dr. Leo Obrst
Information Semantics
Center for Connected Government
MITRE
October 24, 2013

Agenda

- Logic: propositional logic, predicate logic
- Logic Programming
- Ontology as Logical Theory about World
- Ontology Reasoning and Rules
- Issues: Decidability, Complexity

What is Logic?

- Here, our focus is on deductive formal logic
- Formal Logic as a Formal Language
- A Formal Language, just like NL, has a syntax & semantics
- We are interested in formal logics because we can represent our knowledge in these, machines can interpret these, and then perform automated inference (proofs using inference rules)
- The same inferences that humans make

Propositional Logic: Syntax

- PL is a Language having a Syntax & a Semantics
  - A set of symbols:
    - Logical Constants: True, False (or T, F)
    - Logical Variables (or propositional symbols): p, q, r, …
    - Logical Operators (or connectives): $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$, (,)
  - Formulas (Well-formed Formulas or WFFs) of PL (we will call these ***propositions***)
    - Any propositional symbol is a WFF of PL
    - If α and β are WFFs, then so are $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \rightarrow \beta)$, $(\alpha \leftrightarrow \beta)$, and $(\neg\alpha)$ [and note that we will dispense with parentheses where we can]
    - Nothing else is a WFF.
  - So the following are WFFs: p, $\neg$p, p $\vee$ q, p $\wedge$ q, (p $\wedge$ q) $\rightarrow$ r
  - Propositions are things that are true or false

Propositions in English:

If John is a management employee, then John manages an organization.

John is a management employee.

John manages an organization (MP)

Propositions in PL:

p $\rightarrow$ q

p

---

q (MP: Modus Ponens)

Still Need Semantics!

Proof Theory (Syntax) vs. Model Theory (Semantics)

- **Proof Theory**: deductive apparatus of a language
  - ***Axioms***: declaring by fiat certain formulas of L
  - ***Rules of Inference***: determines which relations between formulas of L are relations of *immediate consequence* of L
    - i.e., from $\alpha \Rightarrow \beta$ in one step
    - More generally, ***syntactic consequence*** is: iff there is a derivation in PL of the set of formulas β from the set of formulas α, written α **|-** β
  - Apply rules to Axioms to derive Theorems
  - ***Theorem***: a formula of a formal language that satisfies purely syntactic requirements and has no meaning
- **Formal Model:** a model of a formula of L is an interpretation of L for which the formula comes out true (a proposition)
- **Model Theory**: the theory of interpretations of languages
  - ***Logical Validity:*** **'|= α'** means that α is a logically valid formula of PL iff α is true for every interpretation of PL
  - ***Semantic consequence:*** **'α |= β'** means β is a semantic consequence of α iff there is no interpretation of PL for which α is true and β is false

Semantics: Interpretation

- Interpretation:
  - An *interpretation* of a formal language is an assignment of meanings to its symbols and/or formulas [Hunter, 1973, p.6-7]
  - "An interpretation of PL is an assignment to each propositional symbol (logical variable) of one or other (but not both) of the truth values truth and falsity" [Hunter, 1973, p. 57-58]
- Truth tables: $\neg p \vee (q \wedge r) \equiv (p \rightarrow q) \wedge (p \rightarrow r)$

p	q	r	$\neg p$	$(q \wedge r)$	$\neg p \vee (q \wedge r)$	$(p \rightarrow q)$	$(p \rightarrow r)$	$(p \rightarrow q) \wedge (p \rightarrow r)$
T	T	T	F	T	T	T	T	T
T	T	F	F	F	F	T	F	F
T	F	T	F	F	F	F	T	F
T	F	F	F	F	F	F	F	F
F	T	T	T	T	T	T	T	T
F	T	F	T	F	T	T	T	T
F	F	T	T	F	T	T	T	T
F	F	F	T	F	T	T	T	T

Some Examples

English Example	Logical Translation
Implication: **(1) If pigs can fly, then dogs can talk.**	**(1') P → Q**
Consequence: **(2.A) If pigs can fly, then dogs can talk.** **(2.B) Pigs can't fly.** **(2.C) Therefore** **(2.D) Dogs can't talk.** **INVALID Conclusion!**	**(2.A') P → Q** **(2.B') ¬P** **(2.C') ————————————** **(2.D) ¬Q**
Consequence: **(3.A) If pigs can fly, then dogs can talk.** **(3.D) Dogs can't talk.** **(3.C) Therefore** **(3.B) Pigs can't fly** **VALID Conclusion!**	**(2.A') P → Q** **(2.B') ¬Q** **(2.C') ————————————** **(2.D) ¬P** **This inference rule is Modus Tollens**

Predicate Logic: To Propositional Logic, Add Predicates, Individuals, Quantifiers

Propositions & Predicates in English:

If John is a management employee, then John manages an organization.

John is a management employee.

John manages an organization (MP)

Propositions & Predicates in First Order Predicate Logic:

$p(x) \rightarrow q(x)$

$p(john)$

$q(john)$ **(MP: Modus Ponens)**

Propositions & Predicates in English:

Everyone who is a management employee manages some organization.

Or:

For everyone who is a management employee, there is some organization that that person manages.

John is a management employee.

There is some organization that John manages.

Propositions & Predicates in First Order Predicate Logic:

$\forall x.\ [p(x) \rightarrow \exists y.\ [q(y) \wedge r(x,y)]]$

"For all x, if x is a p, then there is some y such that y is a q, and x is in the r relation to y"

$p(john)$

$\exists y.\ [q(y) \wedge r(john,y)]]$

(MP: Modus Ponens)

Still Need Semantics!

Semantics of Predicate Calculus

- Generalization of Truth Tables
- From Tarski, “truth with respect to a model”
- So Semantic Evaluation is Generalized
- Define the Semantics of FOL expressions (formulae)
  - Interpretation: Mapping of symbols of the formal language (predicates, functions, variables, constants) onto the modeled domain (formal: Domain, relational Structure, or Universe)
  - Valuation: Determine the bindings of variables
  - Compositional (Constructive) Semantics: Determine the semantics of complex expressions inductively based on the definition of the semantics of basic expressions

Logic Programming

- Good Properties:
  - Least Herbrand Model
  - Fixpoint characterization
  - Goal-directed proof procedures
  - These features make logic a real programming language with a clear and complete operational semantics with respect to its declarative semantics [Lloyd, 1984]
- *Horn formulas:
  - Literals: p, $\neg$p
  - Horn Clause: clause with at most one positive literal, e.g., $p_1 \wedge \ldots \wedge p_k \rightarrow q$
  - Horn Formula: conjunction of Horn clauses
  - DavisPutnam procedure (which is a heuristics for SAT) is a polynomial time algorithm for HORNSAT
  - Head :- Body, where Head is positive
  - Negation as failure (see CWA) makes logic programs “nonmonotonic”
  - But: Prolog can have classical negation, just can’t have explicit reference to it
  - Uses SLD Calculus: “Selection”, “Linear”, “Definite”, resolution method
  - **Dowling and Gallier's result: satisfiability of Horn clauses can be tested in linear time

*Vardi, Moshe. 1998. Logic in Computer Science: An Algorithmic Approach. Course notes. http://www.cs.rice.edu/~vardi/sigcse/mv2.ps.gz
**Dowling, W., Gallier, J., Linear Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, Journal of Logic Programming 3 (1984), 267-284

Prolog Fundamentals

- Horn Logic: Horn clauses & Horn rules
- Substitution, Unification: Compute compatible information
- Resolution Theorem-Proving (resolution as an inference rule)
  - Goes back to Robinson (1966), who discovered resolution inference rule & shone light on its efficiency
- Depth-first proving/search
- Implicit quantifiers: Universal, though can have existentials
- Aside: For most theorem-provers, you use “skolemization” for existential quantification
  - Also applies to universal quantification, i.e., substitute a skolem-constant (for unary predicates) or a skolem-function (for n-ary predicates)
  - Quantifier elimination & normalization of expressions helps for efficiency

Declarative vs. Procedural Aspects of Prolog

- Prolog is mostly a declarative language, vs. procedural (operational)
- Mostly declarative semantics, like logic
- Operational (procedural) semantics:
  - Cut operator: developer puts a search-terminator operator in his/her program where it is known that a search down/across, based on additional understanding of the domain, will never be appropriate. This is human knowledge as a kind of implicit meta-rule!
  - Negation as finite failure: if something is not found in the KB, assert that it's not the case, i.e., is false. Note: this is NOT true logical negation!
  - Prolog is order-dependent, i.e., solutions will be based on the order of facts in the knowledge base AND the order of rules
  - Rule order, in particular, is important because rules can have alternatives and can be recursive
- Prolog and most Logic Programming languages assume Closed World Assumption, i.e., if it is not in the KB, it doesn't count
- Logic and Semantic Web languages assume Open World Assumption and true logical negation

Facts, Rules in Prolog vs. Logic

- Facts: instantiated relations (properties)
  - Could have great flexibility in how you implement logical relations; maybe too much!
  - subclass(human, mammal) [human is a subclass of mammal], or
  - class(human) [human is a class]
  - parent(mary,john).
  - female(mary).
  - parent(ann,mary).
- Rules: inference steps; consider the proof syntax generalized MP
  - mother(X,Y) :- parent(X,Y), female(X).
  - grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
- Compare to Logic:

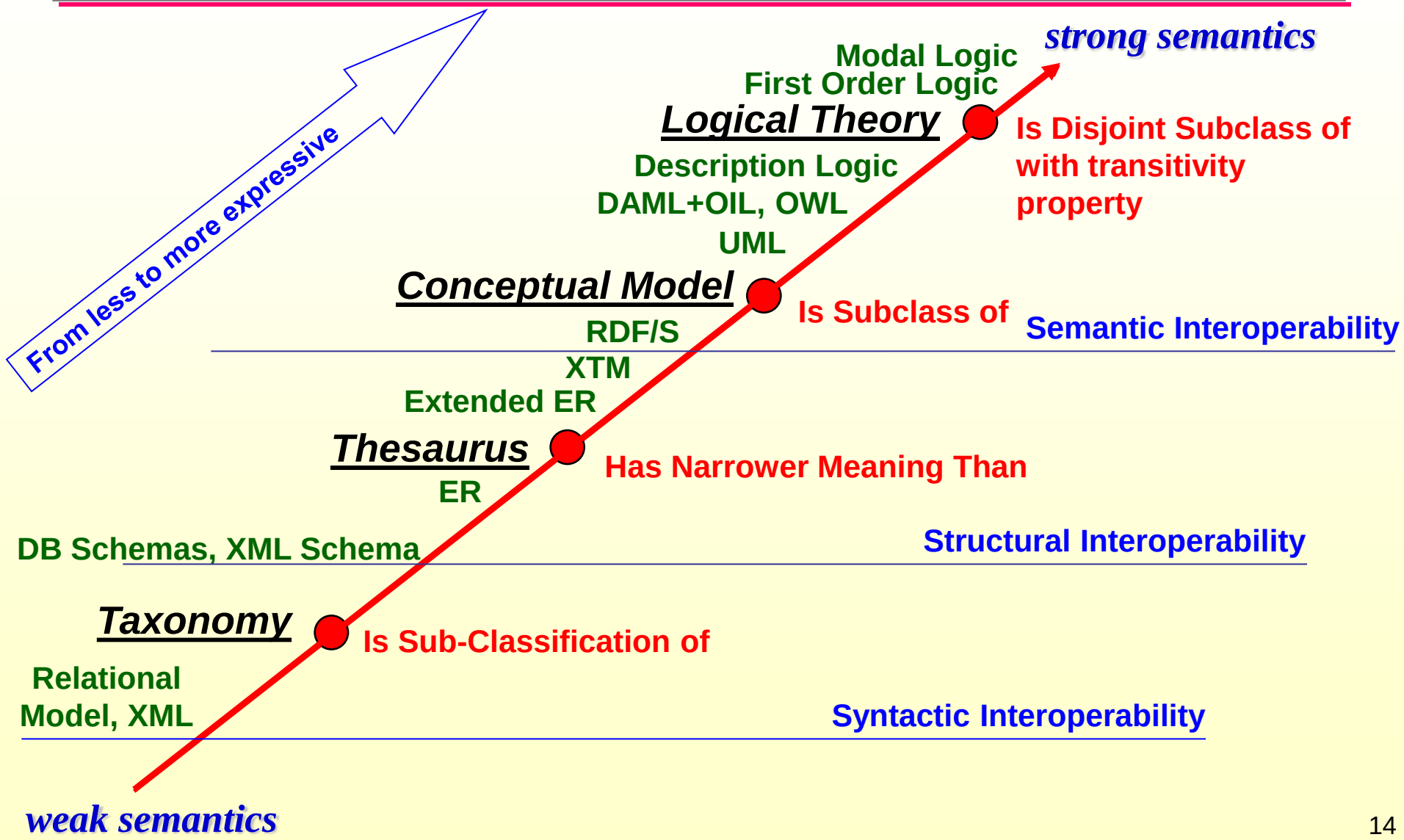
  $\forall X: \forall Y: \text{parent}(X,Y) \wedge \text{female}(X) \rightarrow \text{mother}(X,Y)$.

  $\forall X: \forall Y: \forall Z: \text{parent}(X,Y) \wedge \text{parent}(Y,Z) \rightarrow \text{grandparent}(X,Z)$.

Remember: This is Generalized Modus Ponens, i.e., *consequence* NOT *implication!*

- Compared to Logic, the conclusion is given first, i.e., it's on the left side, it's the Head of the Rule; the Clauses you need to prove (the individual conjuncts or disjuncts) are on the right side.
- These structures are called Horn Clauses & Horn Rules

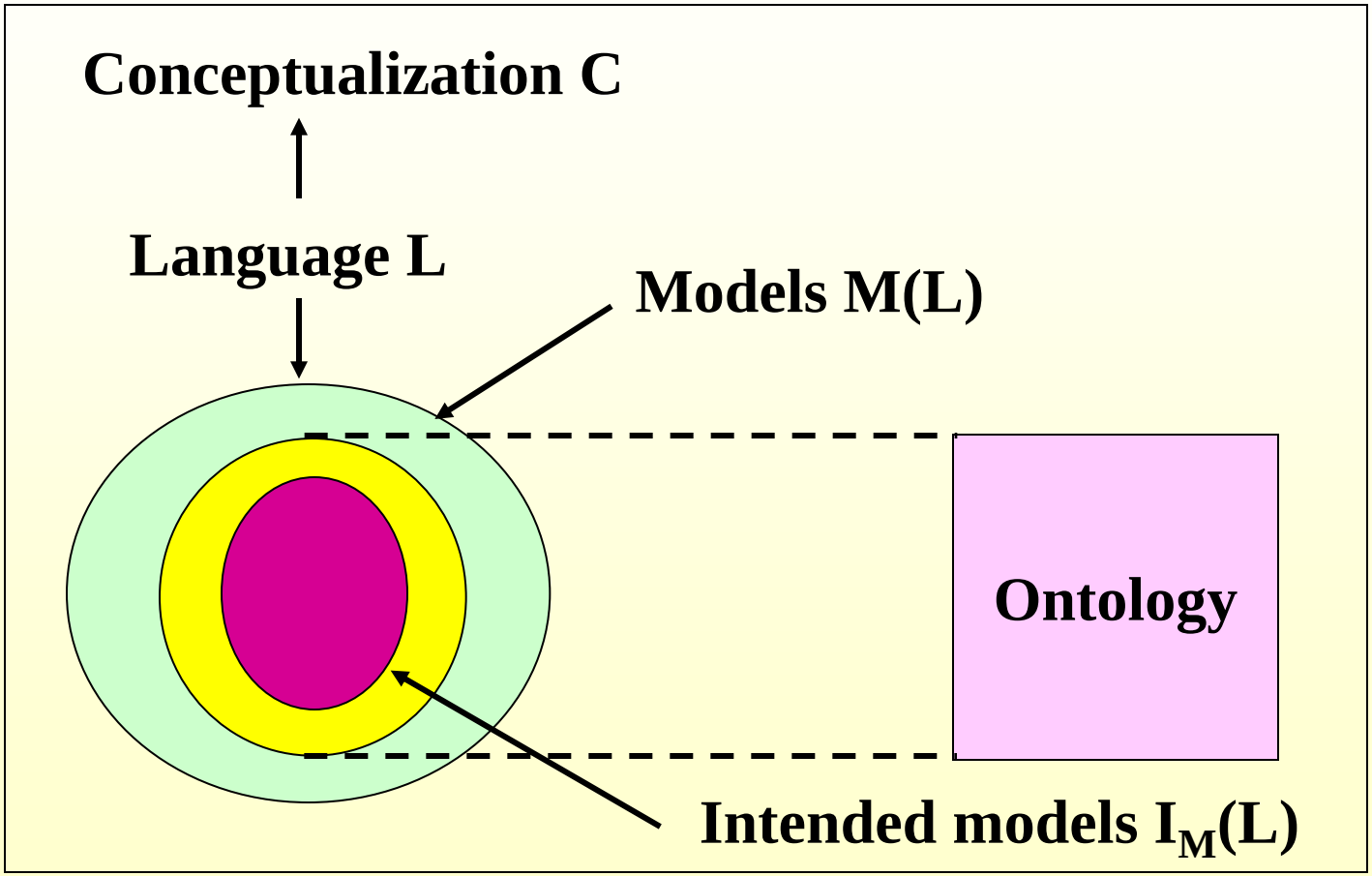
Ontology Spectrum: One View



Ontology as Logical Theory

- A set of axioms designed to account for the intended meaning of a vocabulary
- Axioms are designed such that the set of its models approximates the set of intended models of L according to ontological commitment K
- So, an ontology is a logical theory accounting for the intended meaning of a formal vocabulary (ontological commitment to a particular conceptualization of the world)
- Axioms, inference rules, theorems, theory

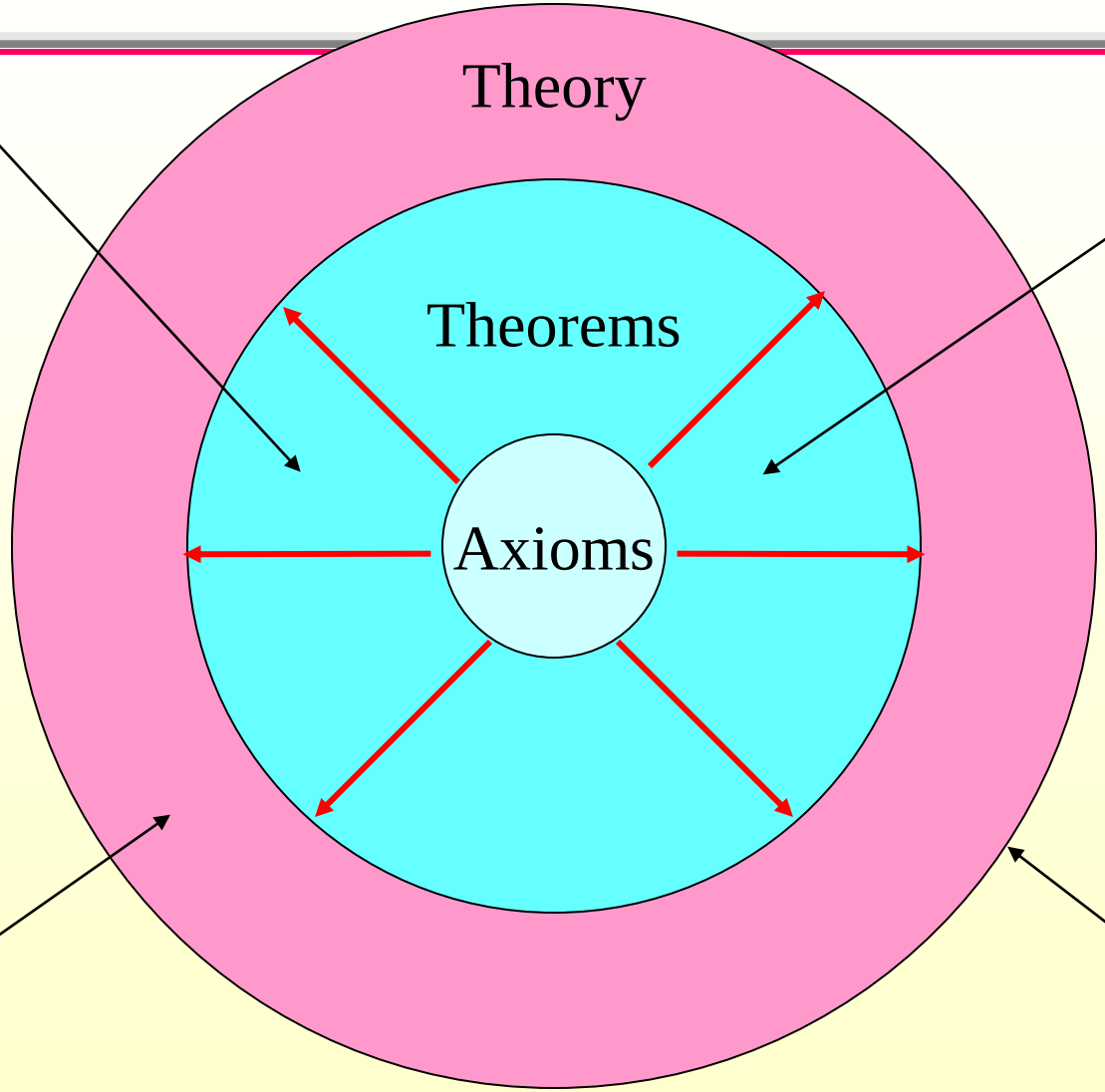
Logical Theories: More Formally



* N. Guarino. 1998. Formal ontology in information systems, pp. 3-15. In Formal Ontology in Information Systems, N. Guarino, ed., Amsterdam: IOS Press. Proceedings of the First International Conference (FOIS'98), June 6-8, Trent, Italy. p. 7

Axioms, Inference Rules, Theorems, Theory

MITRE



(1) Theorems are licensed by a valid proof using ***inference rules*** such as Modus Ponens

(2) Theorems proven to be true can be added back in, to be acted on subsequently like axioms by inference rules

(3) Possible other theorems (as yet unproven)

(4) Ever expanding theory

Axioms

Class(Thing)

Class(Person)

Class(Parent)

Class(Child)

If SubClass(X, Y) then X is a subset of Y. This also means that if A is a member of Class(X), then A is a member of Class(Y)

SubClass(Person, Thing)

SubClass(Parent, Person)

SubClass(Child, Person)

ParentOf(Parent, Child)

NameOf(Person, String)

AgeOf(Person, Integer)

If X is a member of Class (Parent) and Y is a member of Class(Child), then $\neg$ (X =Y)

Inference Rules

And-introduction: given P, Q, it is valid to infer P $\wedge$ Q.

Or-introduction: given P, it is valid to infer P $\vee$ Q.

And-elimination: given P $\wedge$ Q, it is valid to infer P.

Excluded middle: P $\vee$ $\neg$P (i.e., either something is true or its negation is true)

Modus Ponens: given P $\rightarrow$ Q, P, it is valid to infer Q

Theorems

If P $\wedge$ Q are true, then so is P $\vee$ Q.

If X is a member of Class(Parent), then X is a member of Class(Person).

If X is a member of Class(Child), then X is a member of Class(Person).

If X is a member of Class(Child), then NameOf(X, Y) and Y is a String.

If Person(JohnSmith), then $\neg$ ParentOf(JohnSmith, JohnSmith).

Ontology Representation Levels

Level	Example Constructs
Knowledge Representation (KR) Language (Ontology Language) Level: Meta Level to the Ontology Concept Level	Class, Relation, Instance, Function, Attribute, Property, Constraint, Axiom, Rule
Ontology Concept (OC) Level: Object Level to the KR Language Level, Meta Level to the Instance Level	Person, Location, Event, Parent, Hammer, River, FinancialTransaction, BuyingAHouse, Automobile, TravelPlanning, etc.
Ontology Instance (OI) Level: Object Level to the Ontology Concept Level	Harry X. Landsford III, Ralph Waldo Emerson, Person560234, PurchaseOrderTransactionEvent6117090, 1995-96 V-6 Ford Taurus 244/4.0 Aerostar Automatic with Block Casting # 95TM-AB and Head Casting 95TM

Meta-Level to Object-Level

Meta-Level to Object-Level

Language

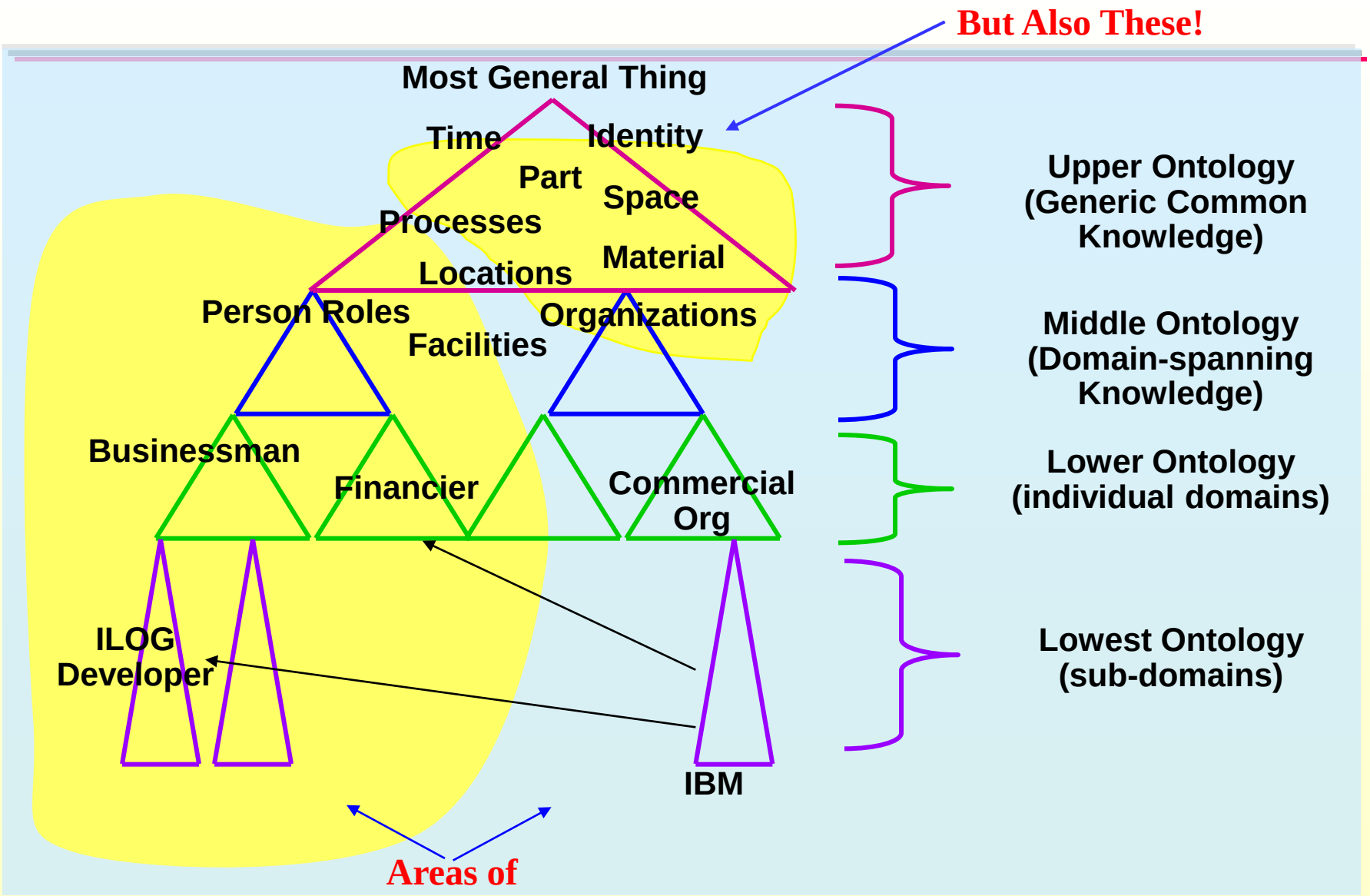
Ontology (General)

Knowledge Base (Particular)

Ontology Example from Electronic Commerce: the general domain of machine tooling & manufacturing; note that these are expressed in English, but usually would be in expressed in a logic-based language

Concept	Example
Classes (general things)	Metal working machinery, equipment and supplies, metal-cutting machinery, metal-turning equipment, metal-milling equipment, milling insert, turning insert, etc.
Instances (particular things)	An instance of metal-cutting machinery is the "OKK KCV 600 15L Vertical Spindle Direction, 1530x640x640mm 60.24"x25.20"x25.20 X-Y-Z Travels Coordinates, 30 Magazine Capacity, 50 Spindle Taper, 20kg 44 lbs Max Tool Weight, 1500 kg 3307 lbs Max Loadable Weight on Table, 27,600 lbs Machine Weight, CNC Vertical Machining Center"
Relations: subclass-of, (kind_of), instance-of, part-of, has-geometry, performs, used-on, etc.	A kind of metal working machinery is metal cutting machinery, A kind of metal cutting machinery is milling insert.
Properties	Geometry, material, length, operation, ISO-code, etc.
Values:	1; 2; 3; "2.5", inches"; "85-degree-diamond"; "231716"; "boring"; "drilling"; etc.
Rules (constraints, axioms)	If milling-insert(X) & operation(Y) & material(Z)=HG_Steel & performs(X, Y, Z), then has-geometry(X, 85-degree-diamond). [Meaning: if you need to do milling on High Grade Steel, then you need to use a milling insert (blade) which has a 85-degree diamond shape.]

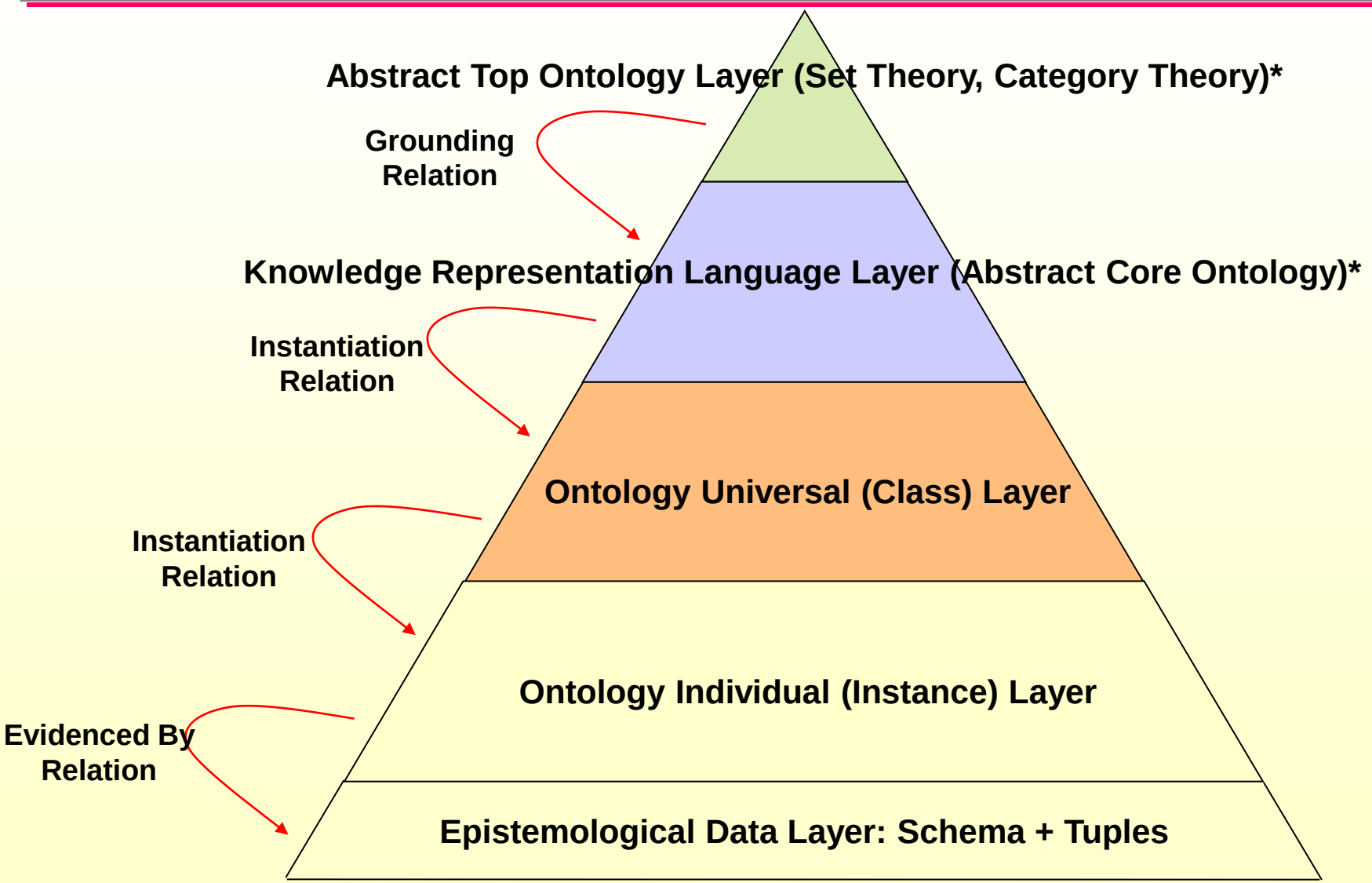
Upper, Middle, Domain Ontologies



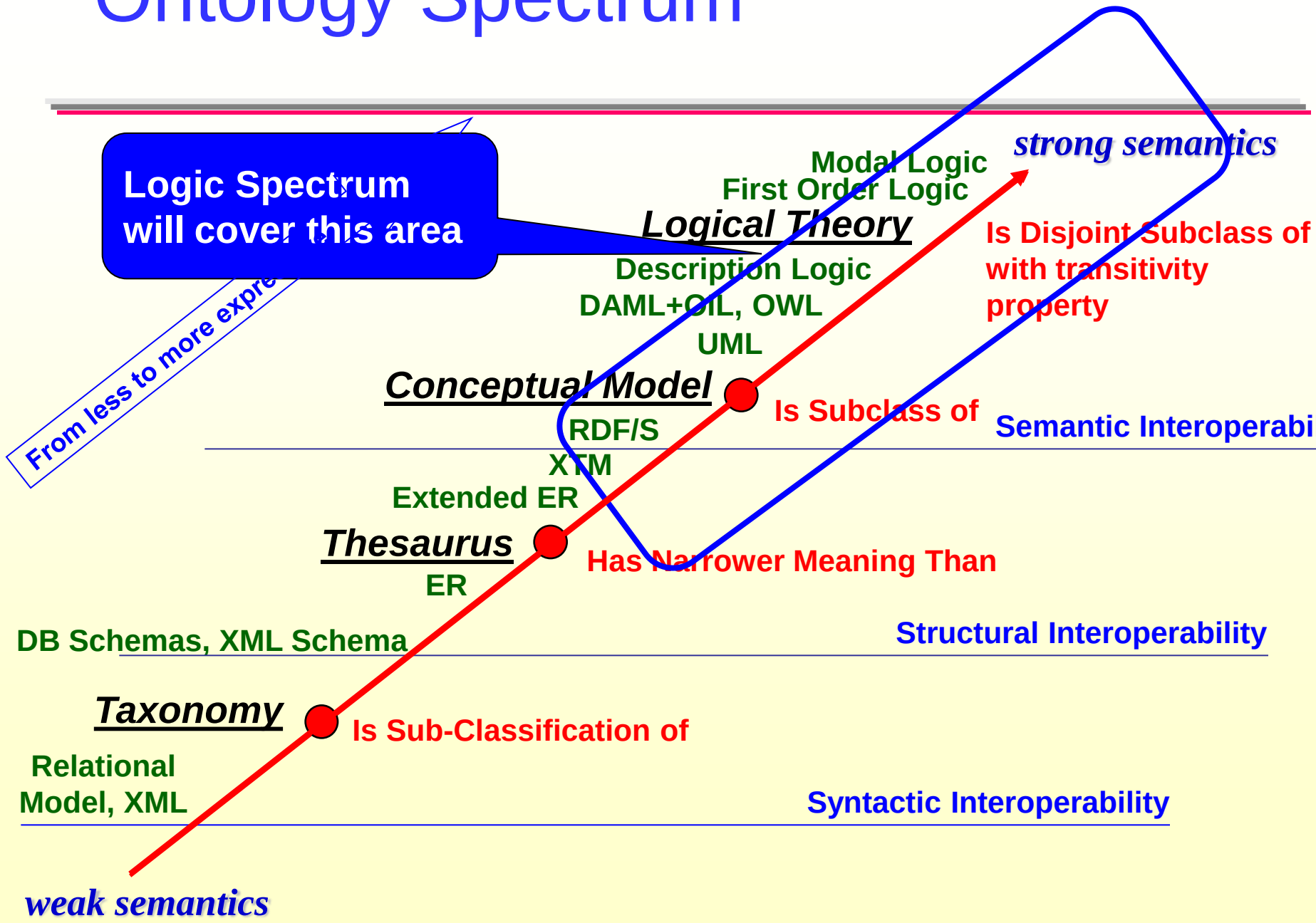
Ontology Content Architecture: More Complex View

MITRE

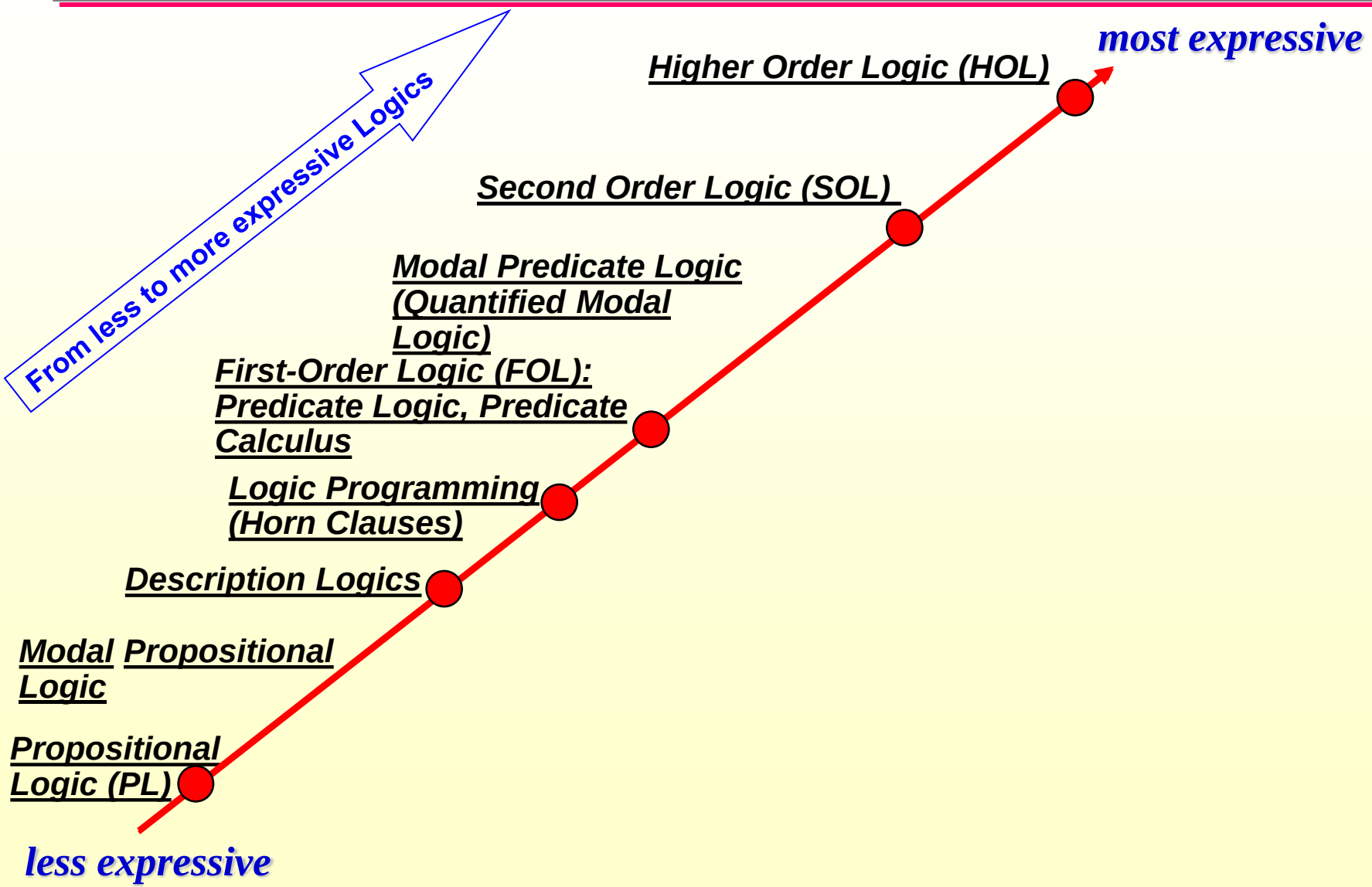
* Adapted from: Herre, Heinrich, and Frank Loebe. 2005. A Meta-ontological Architecture for Foundational Ontologies. In: R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1398–1415, 2005. Springer-Verlag Berlin Heidelberg.



Ontology Spectrum

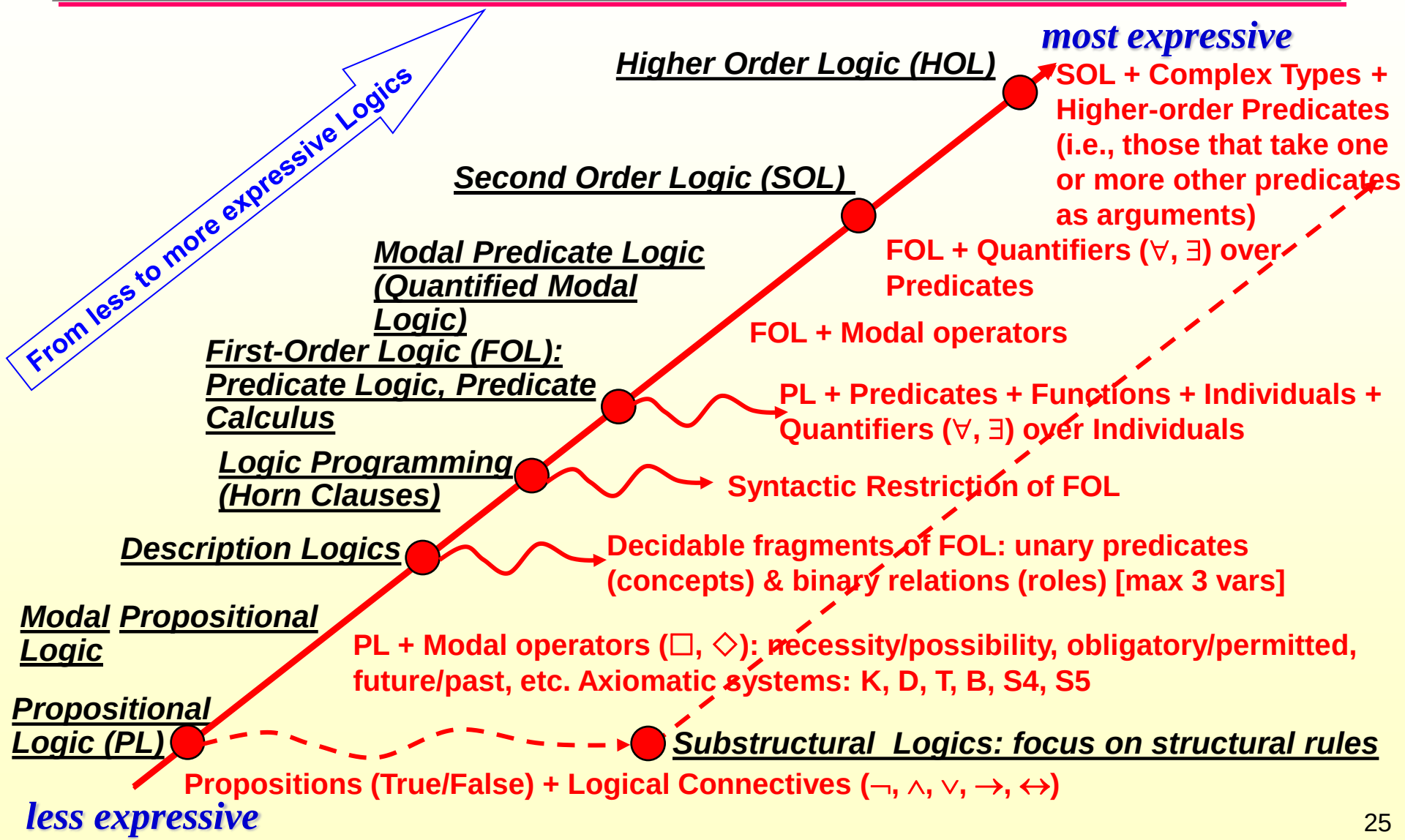


Logic Spectrum

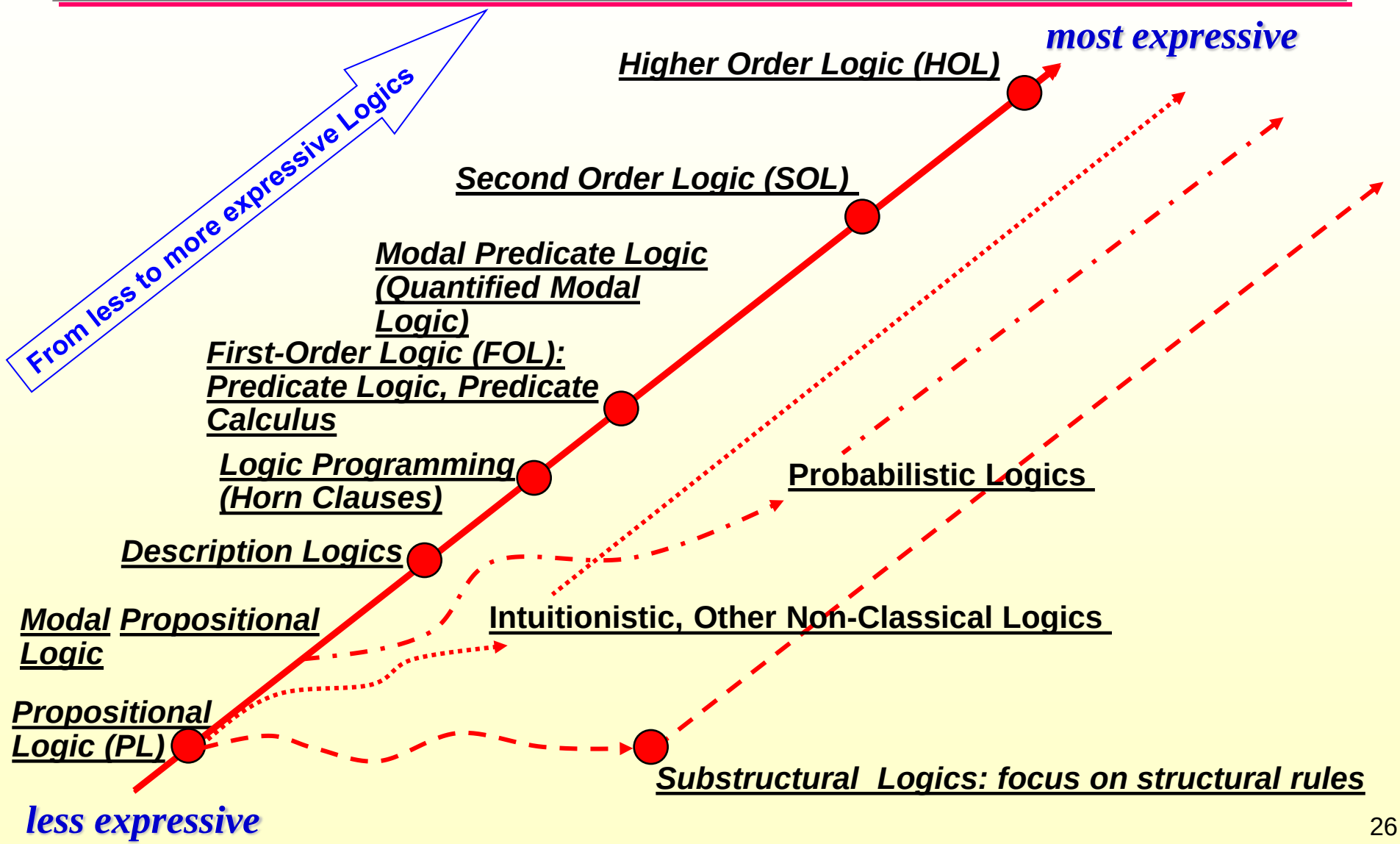


Logic Spectrum: Classical Logics: PL to HOL

MITRE

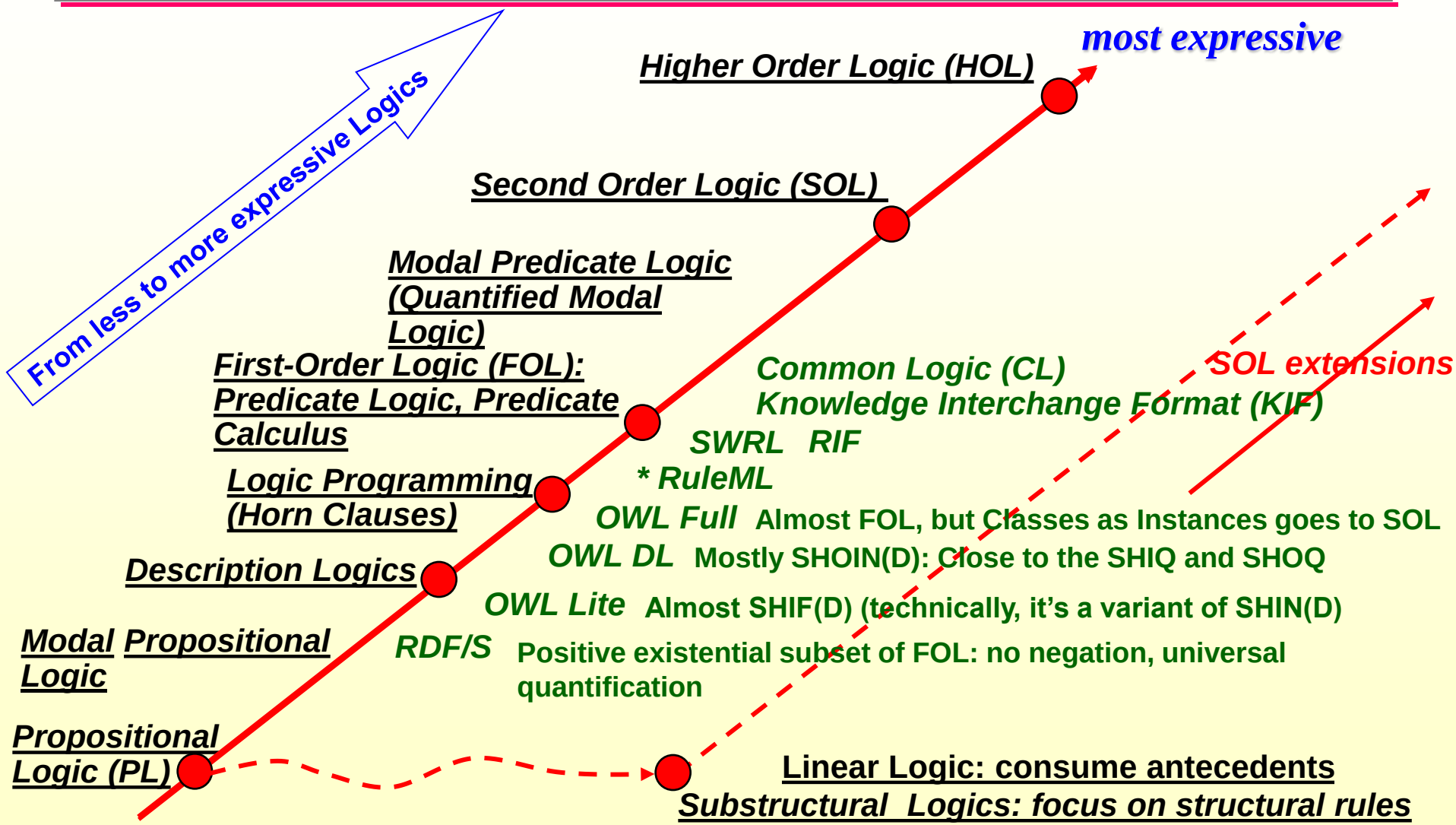


Logic Spectrum: Extending Logic



Logic Spectrum: Languages: Ontologies & Rules

MITRE

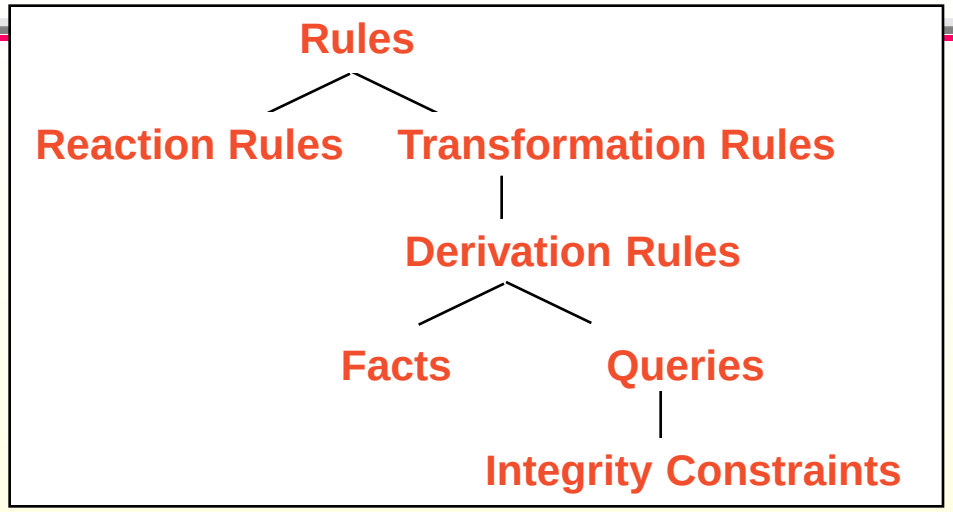


* Expressed syntactically in XML, requires binding to a logic, ranges over all logics

Semantic Web Rules: RuleML, SWRL (RuleML + OWL), RIF

MITRE

RuleML Rule Taxonomy*



*Adapted from Harold Boley, Benjamin Grosof, Michael Sintek, Said Tabet, Gerd Wagner. 2003. RuleML Design, 2002-09-03: Version 0.8. http://www.ruleml.org/indesign.html

- ***Reaction rules*** can be reduced to general rules that return no value. Sometimes these are called "condition-action" rules. Production rules in expert systems are of this type
- ***Transformation rules*** can be reduced to general rules whose 'event' trigger is always activated. A Web example of transformation rules are the rules expressed in XSLT to convert one XML representation to another. "Term rewrite rules" are transformation rules, as are ontology-to-ontology mapping rules
- ***Derivation rules*** can be reduced to transformation rules that like characteristic functions on success just return true. Syntactic $A \mid\!-_P B$ and Semantic Consequence $A \mid\!=_P B$ are derivation rules
- ***Facts*** can be reduced to derivation rules that have an empty (hence, 'true') conjunction of premises. In logic programming, for example, facts are the ground or instantiated relations between "object instances"
- ***Queries*** can be reduced to derivation rules that have – similar to refutation proofs – an empty (hence, 'false') disjunction of conclusions or – as in 'answer extraction' – a conclusion that captures the derived variable bindings
- ***Integrity constraints*** can be reduced to queries that are 'closed' (i.e., produce no variable bindings)

So Which Rules Are Useful, Good, Bad, Ugly?

MITRE

☺ Good

- Logical rules are declarative, confirmable by human beings, machine semantically-interpretable, non-side-effecting
- Logical rules can express everything that production (expert system) rules, procedural rules can
- Logical rules can express business, policy rules, static/dynamic rules

☠ Bad

- Rules expressed in procedural code if-then-else case statements are non-declarative, inspectable by human beings, confirmable with documentation and observance of conformance to documentation, side-effecting (ultimate side-effect: negating a value and returning true for that value)

☹ Ugly

- Expert systems rules “simulate” inference, are pre-logical, have side-effects, tend toward non-determinism, force all knowledge levels to the same level (this is why ontologies and ontological engineering came about), are horrible to debug

Issues: Expressivity

- What do you want to do with your KR language?
  - Build an ontology, build a knowledge base
  - Check consistency of your knowledge
  - Check completeness of your knowledge
  - I.e., Model checking, model finding
  - Automatically classify new concepts, assertions
  - Query the KB (search & navigation)
  - Perform other inference
    - Deduction
    - Induction
    - Abduction
  - Add probabilistic reasoning

Issues: Negation

- Example: First-order positive existential conjunctive logic restricted to binary relations: e.g., RDF/S
- Negation
  - Open World Assumption: can always add more facts
  - Closed World Assumption (negation by finite failure): whatever is in the database, possibly extended by dynamic (but finite) assertions up to some point determines what is the case
    - If X is not provable from theory/found in the database, then NOT X is true
    - One problem: Theory could be inconsistent (from Cadoli & Eiter, 1998, p. 63): T={a $\vee$ b}, CWA(T)={a $\vee$ b, $\neg$a, $\neg$b}
  - Generalized CWA: inference from minimal models
    - T={a $\vee$ b}, CWA(T)={a $\vee$ b}
  - Negation (by finite failure) makes logic programs “nonmonotonic”
    - Classical logic: add an axiom to a 1st order theory, can derive new theorems, no previously proved theorems need to be reproved
    - But adding a rule to a logic program may force some retractions

Issues: Tractability (Complexity)

- Descriptive Complexity: part of Finite Model Theory (MT of finite structures) a branch of Logic and Computer Science
- Decidable, Semi-decidable:
  - **Decidable: there is an effective method for telling whether or not each formula of a system is a theorem of that system or not**
  - **Semi-decidable: If a formula really is a theorem of a system, eventually will be able to prove it is, but not if it is not: may never terminate**
- Complexity Classes

RE (semi-decidable): Recursively enumerable

EXPSPACE: Exponential space — **Intractable**

NEXP: Nondeterministic exponential time

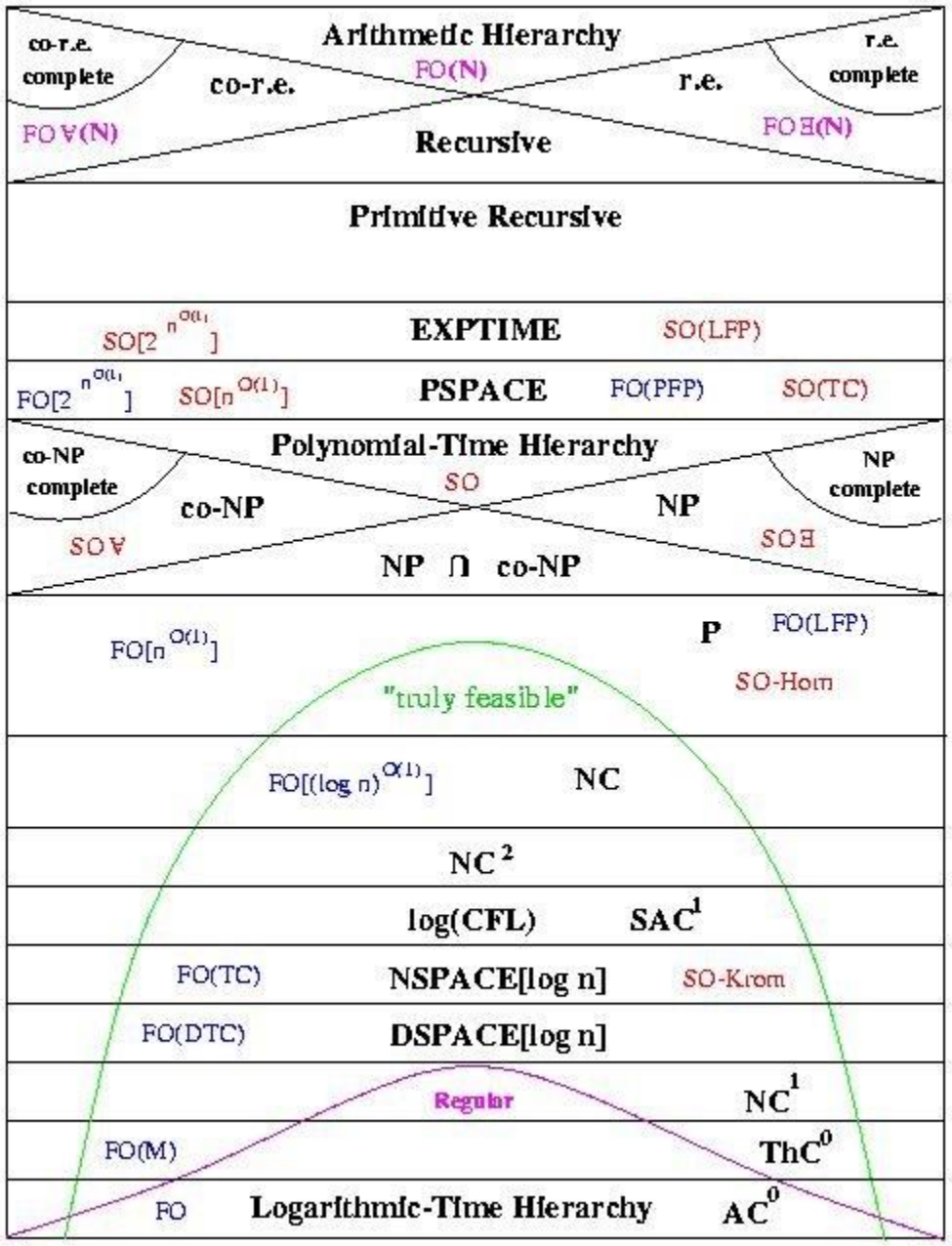
EXP: Exponential time

PSPACE: Polynomial space — **Probably Intractable**

NP: Nondeterministic polynomial time

P: Polynomial time — **Tractable**

Issues: Tractability (Complexity)



*http://www.cs.umass.edu/~immerman/descriptive_complexity.html

Issues: Formal Properties

- Satisfiability: whether a representation is consistent (Note: inference in the formalism has this cost)
- Entailment: whether a formula follows from another formula
- Soundness: any expression that can be derived from the KB is logically implied by that KB
- Completeness: any expression that is logically implied by the KB can be derived
- Decidability: can a sound and complete algorithm be constructed?
- Complexity: is it tractable (worst-case polynomial time) or intractable?
- Expressivity: roughly: expressivity and tractability are inversely proportional
  - some expressive formalisms may be intractable or even undecidable
- Model Checking: whether a state is consistent with the knowledge
- Model Finding: can find a coherent state of the knowledge?

Thanks!

- Next: Benjamin Grosof will provide more detail

Backup

Equivalences

MITRE

Law of Negation:

$\neg \neg p \equiv p$

Combining a Variable with itself:

$p \vee \neg p \equiv TRUE$ Excluded Middle

$p \wedge \neg p \equiv FALSE$ Contradiction

$p \vee p \equiv p$ Idempotence of $\vee$

$p \wedge p \equiv p$ Idempotence of $\wedge$

Properties of Constants:

$p \vee TRUE \equiv TRUE$

$p \vee FALSE \equiv p$

$p \wedge TRUE \equiv p$

$p \wedge FALSE \equiv FALSE$

Commutativity:

$p \wedge q \equiv q \wedge p$

$p \vee q \equiv q \vee p$

Associativity:

$p \vee (q \vee r) \equiv (p \vee q) \vee r$

$p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$

Distributivity:

$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

DeMorgan's Laws:
(Distributing negation over a complex expression, with change of the operator of that expression)

$\neg(p \wedge q) \equiv \neg p \vee \neg q$

$\neg (p \vee q) \equiv \neg p \wedge \neg q$

Subsumption:

$p \wedge (p \vee q) \equiv p$

Conditional Law:

$p \rightarrow q \equiv \neg p \vee q$

Biconditional Law:

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Contrapositive Law:

$p \rightarrow q \equiv \neg q \rightarrow \neg p$

Example Prolog: Transitive Closure (is Subsumed By)

MITRE

- Facts:

```
superClass(a, b).
superClass(a1, b).
superClass(b, c).
superClass(c, d).
superClass(d, e).
superClass(e, f).
superClass(b, c2).
superClass(e, c2).
superClass(c2, d2).
superClass(d2, e2).
superClass(e2, f2).
superClass(a, b2).
superClass(b2, c3).
superClass(c3, d3).
```

- Rules:

```
ancestor(X,Y) :- superClass(X,Y).
ancestor(X,Y) :- superClass(X,Z), ancestor(Z,Y).
```

- We assert the above by either consulting a file (batch mode) or entering these in interactively (former is preferred!)

Example: Transitive Closure (is Subsumed By) Results

MITRE

- Results can be written to a file/database or displayed on screen
- Assume the previous file was called transClosure.pro was loaded (consulted).
- Then the query was executed:

```
test1(L);
Consulting Source Files: 'transClosure.pro'
Type 'quit.' to End
?- test1(L).
L = [[a, b], [a1, b], [b, c], [c, d], [d, e], [e, f], [b, c2], [e, c2], [c2, d2], [d2, e2],
    [e2, f2], [a, b2], [b2, c3], [c3, d3], [a, c], [a, c2], [a, d], [a, e], [a, f], [a, c2],
    [a, d2], [a, e2], [a, f2], [a, d2], [a, e2], [a, f2], [a1, c], [a1, c2], [a1, d], [a1, e],
    [a1, f], [a1, c2], [a1, d2], [a1, e2], [a1, f2], [a1, d2], [a1, e2], [a1, f2], [b, d],
    [b, e], [b, f], [b, c2], [b, d2], [b, e2], [b, f2], [c, e], [c, f], [c, c2], [c, d2], [c, e2],
    [c, f2], [d, f], [d, c2], [d, d2], [d, e2], [d, f2], [b, d2], [b, e2], [b, f2], [e, d2], [e,
    e2], [e, f2], [c2, e2], [c2, f2], [d2, f2], [a, c3], [a, d3], [b2, d3]] ;
no
?-
```

Rules of Inference:

Note that P & Q in the following can each be a simple or complex expression

MITRE

Modus Ponens

$p \rightarrow q$

p

———

q

$\leftrightarrow$Introduction

$p \rightarrow q$

$q \rightarrow p$

———

$p \leftrightarrow q$

Case Analysis (sometimes called $\vee$ Elimination)

$p \vee q$

$p \rightarrow r$

$q \rightarrow r$

———

r

$\wedge$ Introduction

p

q

———

$p \wedge q$

$\vee$ Introduction

p

———

$p \vee q$

Substitution (when α=β)

α

———

β

Modus Tollens

$p \rightarrow q$

$\neg q$

———

$\neg p$

Contrapositive

$p \rightarrow q$

———

$\neg q \rightarrow \neg p$

Vacuous Proof

$\neg p$

———

$p \rightarrow q$

$\wedge$ Elimination

$p \wedge q$

———

p

Contradiction

p

$\neg p$

———

FALSE

Tautology (when α= TRUE)

———

α

NOTE: the following two rules have assumptions and so indentation is necessary above the line.

Reduction to Absurdity

[p]

FALSE

———

$\neg p$

$\rightarrow$Introduction

[p]

q

———

$p \rightarrow q$

Prolog vs. Logic

- Prolog does backward-chaining
  - Go from the theorem you want to prove & prove its dependent components
  - As opposed to forward-chaining, in which you prove the components & those solutions are posted on some global space (data structure), which other forward rules then can have at
- Prolog assumes Universal Quantification: builds it in, so every query/rule is universally quantified, by default
- Prolog will return the first satisfying values, then if you request more, all the remainders
  - There are also Prolog constructs to just do all the solutions & find all the values (FINDALL)
- Remember: queries are like theorems
  - The difference? Proven theorems return T or F; queries return the bindings which make those true theorems true

Example: Axioms on Social Roles 1

Masolo, Claudio; Laure Vieu; Emanuele Bottazzi; Carola Catenacci; Roberta Ferrario; Aldo Gangemi; Nicola Guarino. 2004. Social Roles and their Descriptions, Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning. AAAI Press, 2004.

- Descriptions and Concepts

  (A1) $DS(x) \rightarrow NASO(x)$

  (A2) $CN(x) \rightarrow NASO(x)$

  (A3) $DS(x) \rightarrow \neg CN(x)$

- Concept Use and Definition

  (A4) $US(x, y) \rightarrow (CN(x) \wedge DS(y))$

  (A5) $DF(x, y) \rightarrow US(x, y)$

  (A6) $CN(x) \rightarrow \exists y(DF(x, y))$

  (A7) $DS(x) \rightarrow \exists y(US(y, x))$

  (A8) $(DF(x, y) \wedge DF(x, z)) \rightarrow y = z$

  (A9) $US(x, y) \rightarrow (PRE(y, t) \rightarrow PRE(x, t))$

  (A10) $DF(x, y) \rightarrow (PRE(x, t) \rightarrow PRE(y, t))$

  (T1) $DF(x, y) \rightarrow (CN(x) \wedge DS(y))$ *(A4),(A5)*

  (T2) $CN(x) \rightarrow \exists! y(DF(x, y))$ *(A6),(A8)*

  (T3) $DF(x, y) \rightarrow (PRE(x, t) \leftrightarrow PRE(y, t))$ *(A5),(A9),(A10)*

Example: Axioms on Social Roles 2

- Classification

  (A11) $CF(x, y, t) \rightarrow (ED(x) \wedge CN(y) \wedge TL(t))$

  (A12) $CF(x, y, t) \rightarrow PRE(x, t)$

  (A13) $(CF(x, y, t) \wedge DS(x)) \rightarrow \neg US(y, x)$

  (A14) $CF(x, y, t) \rightarrow \neg\, CF(y, x, t)$

  (A15) $(CF(x, y, t) \wedge CF(y, z, t)) \rightarrow \neg CF(x, z, t)$

- Anti-Rigidity

  (D1) $AR(x) \equiv_{df} \forall y,t(CF(y, x, t) \rightarrow \exists t'(PRE(y, t') \wedge \neg CF(y, x, t')))$

- Founded

  (D2) $FD(x)\ ) \equiv_{df} \exists y,d(DF(x, d) \wedge US(y, d)$ Ÿ
  $\forall z,t(CF(z, x, t) \rightarrow$
  $\exists z'(CF(z', y, t) \wedge \neg P(z, z', t) \wedge \neg P(z', z, t)))$

- Role

  (D3) $RL(x)\ ) \equiv_{df} AR(x) \wedge FD(x)$

Reaction Rules (Event-Condition-Action)

Reaction rules **can be reduced to general rules that return no value. Sometimes these are called "condition-action" rules. Production rules in expert systems are of this type. Note that these are similar to logical derivation rules but are side-effecting (state-changing), i.e., non-monotonic**

General rule form: Condition → Conclusion, i.e., If Condition Then Conclusion

Reaction rule form: Event-Condition-Action
Event: some occurrence triggers or invokes the rule
Condition: the rule fires and the condition gets evaluated. The condition can be anything but in general represents a particular state, i.e., if a property or set of properties hold.
Action: if the condition is met, then the action is performed

In general, the action can arbitrarily change the state of the rule environment, thus is *non-monotonic*

Example: If (boiler.pressure > 1000 lbs/sq in) then (boiler.state = shutdown)
If the boiler pressure is greater than 1000 lbs/sq in, shut the boiler down

Transformation Rules

Transformation rules **can be reduced to general rules whose 'event' trigger is always activated. A Web example of transformation rules are the rules expressed in XSLT to convert one XML representation to another. "Term rewrite rules" are transformation rules, as are ontology-to-ontology mapping rules**

General rule form: LeftHandSide $\rightarrow$ RightHandSide
LeftHandSide: a formal language expression
RightHandSide: a formal language expression

Context Free grammar rules (also [Extended] Backus-Naur, etc.) are like this:
S $\rightarrow$ a b
S $\rightarrow$ a S
Equivalence rules in Logic, Mathematics: e.g., DeMorgan's Laws:
$\neg$ (P $\vee$ Q) = ($\neg$ P) $\wedge$ ($\neg$ Q) **2 * (3 + 4) = (2 * 3) + (2 * 4)**
$\neg$ (P $\wedge$ Q) = ($\neg$ P) $\vee$ ($\neg$ Q)

General Rewrite Rules:
A, B $\rightarrow$ B A

Derivation Rules

Derivation rules **can be reduced to transformation rules that like characteristic functions on success just return true. Syntactic A $|-_P$ B and Semantic Consequence A $|=_P$ B are derivation rules**

General rule form: Antecedent |– Consequent (syntactic consequence)

Example: Modus Ponens
X, X $\rightarrow$ Y |– Y

Facts, Queries, Integrity Constraints

- ***Facts*** **can be reduced to derivation rules that have an empty (hence, 'true') conjunction of premises. In logic programming, for example, facts are the ground or instantiated relations between "object instances"**

- **General rule form: |– Consequent (syntactic consequence)**
- **Example: father (johnSmith, marySmith)**
- **|– Y**

- ***Queries*** **can be reduced to derivation rules that have – similar to refutation proofs – an empty (hence, 'false') disjunction of conclusions or – as in 'answer extraction' – a conclusion that captures the derived variable bindings**

- **General rule form: Antecedent |– ∅ (syntactic consequence)**
- **Example: person (X), person (Y), father (X, Y)?**
- **X |–**

- ***Integrity constraints*** **can be reduced to queries that are 'closed' (i.e., produce no variable bindings)**

Example: Inference and Proof

Proof Using Inference Rule of Modus Ponens

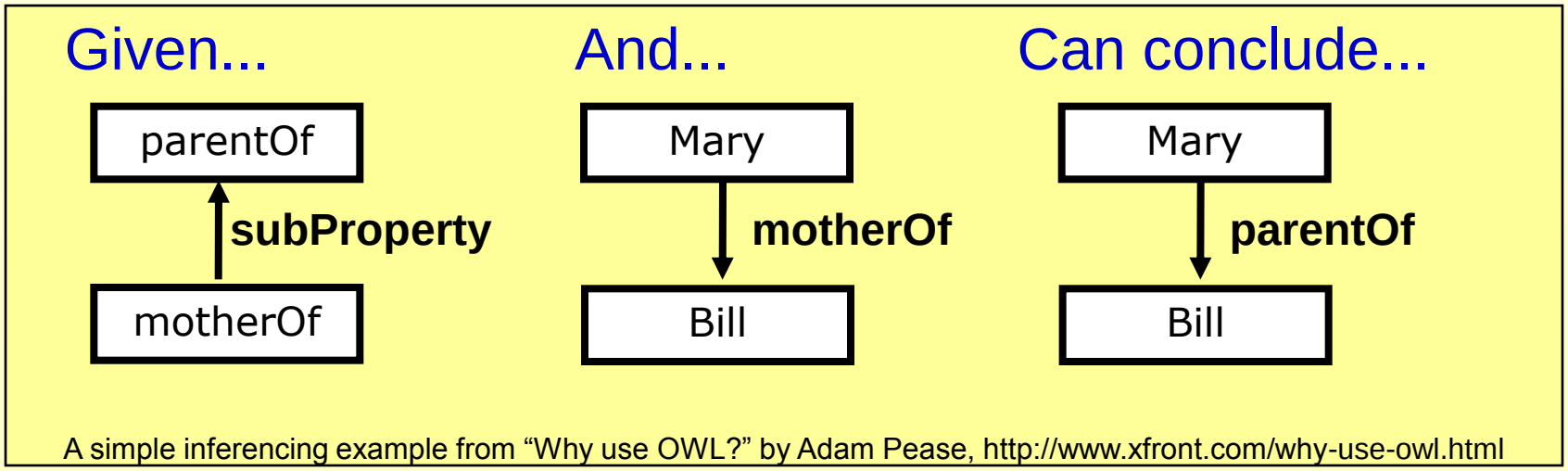
Given: **If motherOf is a subProperty of parentOf, and Mary is the mother of Bill, then Mary is the parentOf Bill**

motherOf is a subProperty of parentOf

Mary is the motherOf Bill

Infer: **Mary is the parentOf Bill**

Deduction A method of reasoning by which one infers a conclusion from a set of sentences by employing the axioms and rules of inference for a given logical system.



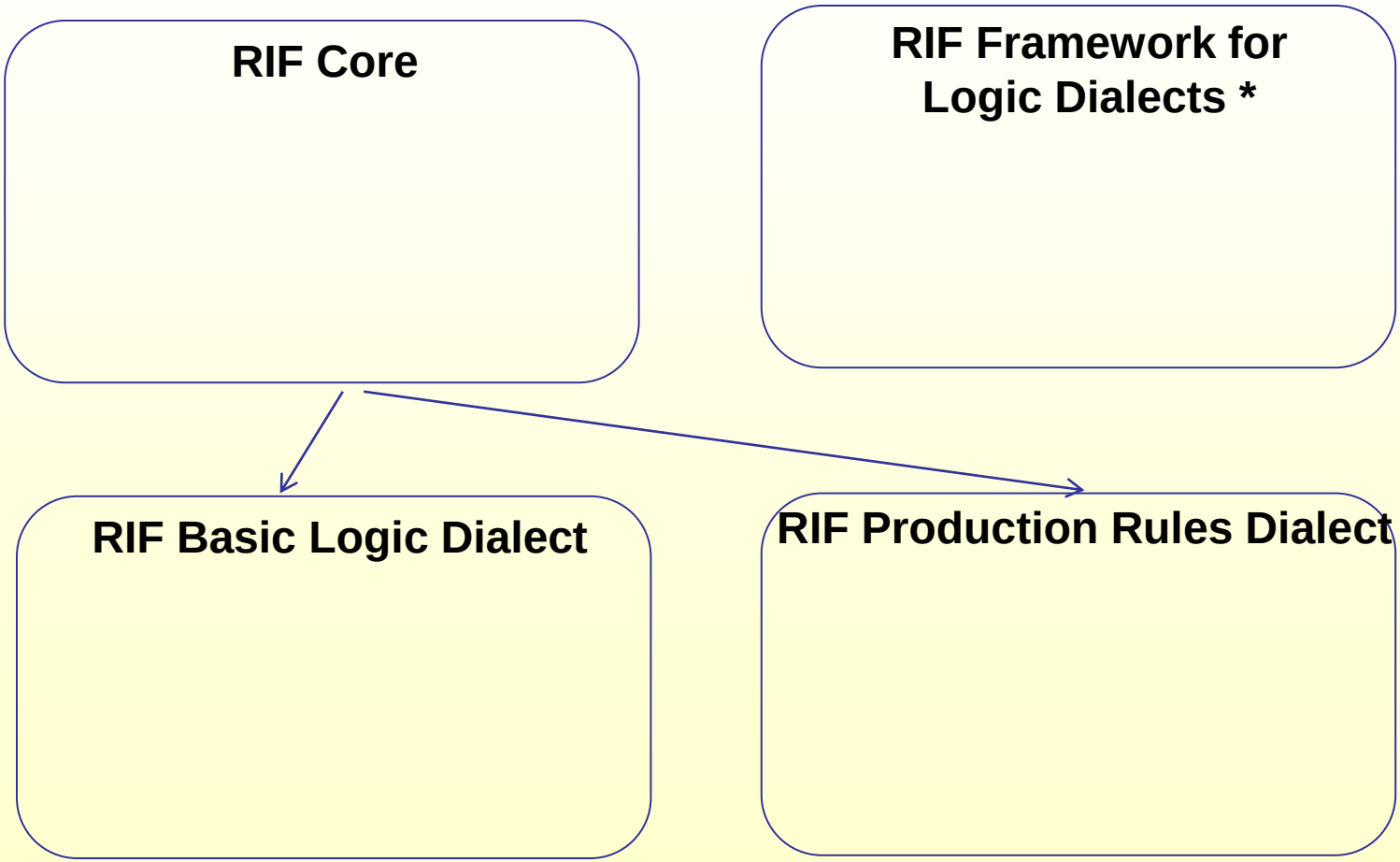
A simple inferencing example from "Why use OWL?" by Adam Pease, http://www.xfront.com/why-use-owl.html

Rule Interchange Format (RIF)*

- RIF is a rule language based on XML syntax
- RIF provides multiple versions, called *dialects:*
  - **Core:** the fundamental RIF language, and a common subset of most rule engines (It provides "safe" positive datalog with builtins)
  - **BLD (Basic Logic Dialect**): adds to Core: logic functions, equality in the *then*-part, and named arguments (This is positive Horn logic, with equality and builtins)
  - **PRD (Production Rules Dialect):** adds a notion of forward-chaining rules, where a rule *fires* and then performs some action, such as adding more information to the store or *retracting* some information (This is comparable to production rules in expert systems, sometimes called condition-action, event-condition-action, or reaction rules)

•http://www.w3.org/TR/rif-overview/
•http://www.w3.org/2005/rules/wiki/RIF_Working_Group
•http://www.w3.org/2005/rules/wiki/RIF_FAQ

RIF Dialects



* http://www.w3.org/TR/rif-fld/

Description Logics

- **What is a Description Logic?**
  - Synonyms: Terminological Logic, Concept Logic, based on Concept Language, Term Subsumption Language
  - A declarative formalism for the representation and expression of knowledge and sound, tractable reasoning methods founded on a firm theoretical (logical) basis
  - Expressive, sound & complete, decidable, classical semantics, tractable reasoning
  - Function-free FOL using at most 3 variables (basic)
- ***A description*:** an expression in a formal language that defines a set of instances or tuples
- **DL**: a syntax for constructing descriptions and a semantics that defines the meaning of each description
- ***Components***
  - ***T-box:*** *Terminological box – concepts, classes, predicates*
    - One or more subsumption hierarchies/taxonomies of descriptions
    - Terminological axioms: introduce names of concepts, roles
    - Concepts: denote entities
    - Roles: denote properties (binary predicates, relations)
    - Subsumption: comparable to matching or unification in other systems
  - ***A-box:*** *Assertional box – individuals, constants*
    - Instances in the OO world, tuples in the DB world

First Order & Higher Order Logics

- **FOL semi-decidable**
  - **Decidable: there is an effective method for telling whether or not each formula of a system is a theorem of that system or not**
  - **Semi-decidable: If a formula really is a theorem of a system, eventually will be able to prove it is, but not if it is not: may never terminate**
- **Second Order: sometimes used in linguistics**
  - **"Tall", "Most", etc.**
  - **Quantification over Individual & Predicate variables**
  - **$\exists\phi$ (ϕ (a) $\wedge$ F(ϕ)): "John has an unusual property"**
- **CYCL has some constrained 2nd order reasoning**
- **Theorem-provers**
  - **HOL, Prover9, etc.**
- **Prolog & Cousins**
  - **Restricted FOL: Horn Clauses (only 1 un-negated term in a formula, resolution method proves the contradiction of the negation of a term)**
  - **Non-standard negation: negation by finite failure**
  - **Closed World Assumption**
  - **Declarative + Operational Semantics: use of Cut**
- **Other: Conceptual Graphs, UML, Expert System Shells, Modal Logics**

Limitations of FOL, Other Logics

- Expressive limitations of FOL:
  - Possible non-monotonicity:

    All birds can fly
    All ostriches are birds
    Tweety is an ostrich
    _____
    *Tweety can fly NO!
- Quantifiers: Existential, Universal
- Negation: what kind?
- Other:
  - Generalized Quantifiers
  - Types & Sorts
  - Probabilistic Reasoning
  - Possibilistic Reasoning
  - Etc.

Issues: Tractability (Complexity)

- Intractable: exponential in the worse case (meaning some are hard to solve)
  - What is more typical (average?) case?
- Satisfiability decision process (SAT): NP-complete
  - NP-complete: can be solved by a nondeterministic Turing Machine in polynomial time, with the additional property that it is also NP-hard (solving it in polynomial time would make it possible to solve all problems in class NP in polynomial time)
  - FO SAT is co-RE-complete (co: complementary)
- Syntactic restrictions:
  - Horn Clause formulae: SAT is polynomial
  - $\exists x,\ldots, \forall y,\ldots. \phi$ with no functions symbols: NEXP-complete
  - Some relations on finite models expressible in FOL
    - Graphs: symmetric, transitive, ok
    - But not: Is graph A the transitive closure of graph B?