Semantic Web

— to Artificial Intelligence

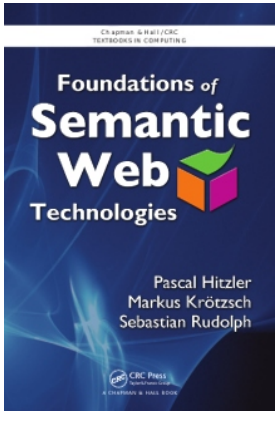
Yue Ma

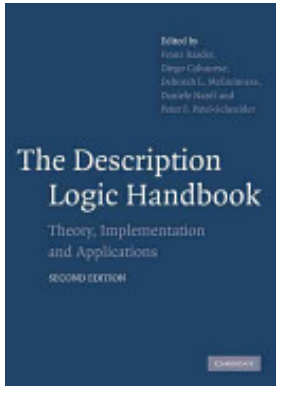
Laboratoire de Recherche en Informatique (LRI)
Université Paris Sud
ma@lri.fr

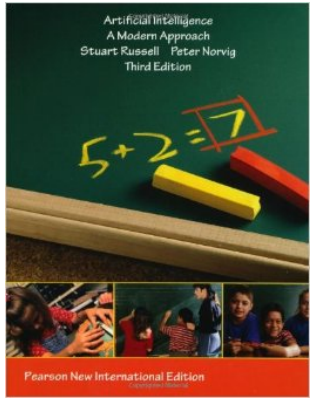
General Information

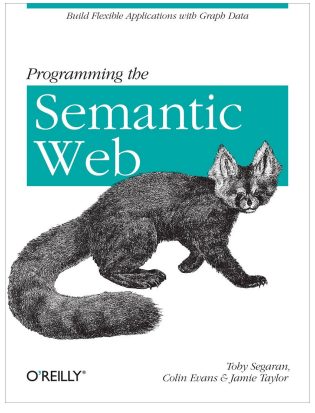
- This lecture is inspired by the following courses :
  - Knowledge Representation for the Semantic Web
    Pascal Hitzler, Wright State University, US
  - Foundations of Semantic Web Technologies
    Sebastian Rudolph, TU Dresden, Germany
  - Knowledge Graph Analysis
    Jens Lehmann, University of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

- Reference books :









General Information

- Course materials : `www.lri.fr/~ma/M2DK`
- Organizational matters :
  - Initial plans :
    - Part 1 : Semantic Web Standards (RDF, RDFS, OWL, SPARQL)
    - Part 2 : Semantics and Reasoning for Semantic Web
    - Projects : Querying Semantic Data, Ontology Reasoning
  - Grading : average(Projects, Exam)

(Semi) Decidability, Complexities, and DL Families

Outline

Solving Problems

Decidability, Semi-decidability, Complexties

Expressive and Lightweight DLs

Computability and Problems

In TCS and math, the main interest in the machinery of computability comes from the desire to solve certain discrete problems. Here are the most important types :

- Decision Problems : Return a Yes/No answer.
- Counting Problems : Count objects of a certain kind.
- Function Problems : Calculate a certain function.
- Search Problems : Select one particular solution.

Decision Problems

A decision problem consists of a set of instances and a subset of Yes-instances.

Problem : **Primality**
Instance : A natural number x.
Question : Is x a prime number ?

Here the set of all instances is $\mathbb{N}$ and the set of Yes-instances is $\{p \in \mathbb{N} \mid p \text{ is prime}\}$.
The answer (solution) to any decision problem is just one bit (true or false).

Counting Problems

A counting problem consists of a set of instances, each instance I is associated with a set of "solutions" sol(I). We need to calculate the cardinality of sol(I).

Problem : **Prime Counting**
Instance : A natural number x.
Question : The number of primes $p \leq x$.

Here the set of all instances is $\mathbb{N}$ and the "solutions" for x are $\{p \leq x \mid p \text{ is prime } \}$. In the literature, this function is denoted $\pi(x)$.
The answer to a counting problem is always a natural number.

Function Problems

A function problem consists of a set of instances and, for each instance I, a unique solution sol(I).

Problem : **Next Prime**
Instance : A natural number x.
Question : The least prime $p > x$.

Instances are again $\mathbb{N}$ and the solution for $x \in \mathbb{N}$ is $\text{sol}(x) = p$ where p is the appropriate prime (uniquely determined).
We insist that there always is a solution (otherwise sol would be a partial function).

Search Problems

A search problem consists of a set of instances and, for each instance I, a set of solutions sol(I).
In this case, "the" solution is not required to by unique. And, we allow for sol(I) to be empty. This turns out to be very convenient in practice.

Problem : **Factor**
Instance : A natural number x.
Question : A natural number $z, 1 < z < x$, dividing x.

Note that the set of solutions is empty if x is prime.

Solving a Problem

- To solve a decision problem, a computable function has to accept each instance of the problem as input, and return "Yes" or "No" depending on whether the instance is a Yes-instance.
- To solve a counting problem, a computable function has to accept each instance x of the problem as input, and return the appropriate count $|\mathrm{sol}(x)|$ as answer.
- To solve a function problem, a computable function has to accept each instance x of the problem as input, and return the unique $\mathrm{sol}(x)$.
- To solve a search problem, a computable function has to accept each instance x of the problem as input, and return either an element of $\mathrm{sol}(x)$ or "No" if $\mathrm{sol}(x)$ is empty (the instance has no solutions).

Connections

Note that **Primality, Prime Counting** and **Next Prime** are closely connected : an algorithm for one problem can be turned into an algorithm for the other, plus a modest bit of overhead (a white lie). This idea of a reduction is critical in computability theory and in complexity theory.

Factor is a bit different, though : there are primality tests that provide no insight into factors of a composite number.

In fact, we now know that **Primality** is easy in the sense that there is a polynomial time algorithm for it, but we fervently hope that **Factor** will turn out to be hard (certain cryptographic methods will fail otherwise).

Our examples have $\mathbb{N}$ as the set of instances. In general, for some general decision problem Π we may have

- a set I_Π of instances, and
- a set $Y_\Pi \subseteq I_\Pi$ of Yes-instances.

For example, I_Π might be the collection of all undirected graphs, and Y_Π could be the collection of all connected undirected graphs.

More Instances

One might wonder what an algorithm is supposed to do with input that is not an instance of the problem in question.
There are two choices :

- We don't care : the behavior of the algorithm can be arbitrary.
- More realistic : the algorithm rejects bad input. This is perfectly reasonable since the collection of all inputs is always trivially decidable (in fact very low in the p.r. hierarchy). Something very fishy is going on if it is not.

Algorithms versus Computability

We will often informally talk about algorithms without offering any definition of this concept ; every CS person knows what an algorithm is. We could cheat and say that an algorithm is just a computable function.

Alas, that is just not right. Certainly, every algorithm expresses a computable function, but it is surprisingly difficult to give a coherent and useful definition of what constitutes an algorithm. If you want to get an idea of how one might think about this problem, look at the paper by Moschovakis on the web.

You can think of an algorithm as :
a computable function, taking into account implementation issues and efficiency.

Outline

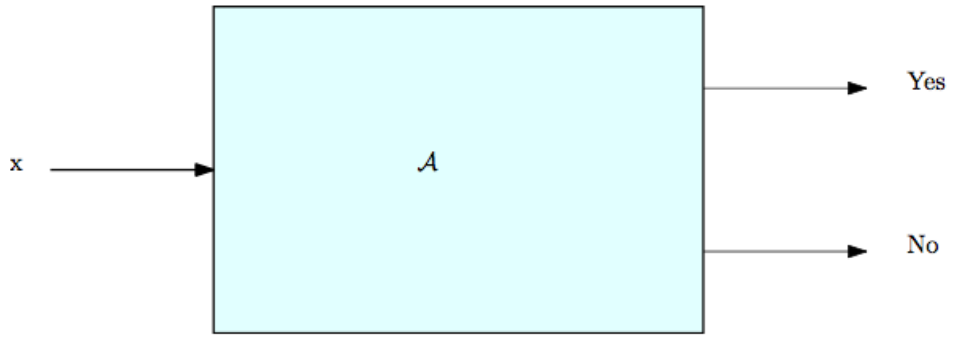
Solving Problems

Decidability, Semi-decidability, Complexties

Expressive and Lightweight DLs

Decidability

Informally, a problem is decidable if there is a decision algorithm $\mathcal{A}$ that returns Yes or No depending on whether the input has the property in question.



Decision Algorithms

Decision problems have been around since the day of the flood : one is interested in checking whether a number is prime, whether a polynomial is irreducible, whether a polygon is convex, and so on. Gauss certainly understood the computational difficulty of primality checking.

The first big splash came in 1900, when Hilbert presented his famous list of 23 open problems at the International Congress of Mathematicians in Paris :

...

8. Prove the Riemann Hypothesis. ...
9. ...
10. Determination of the solvability of a Diophantine equation Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients : to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.

Entscheidungsproblem

The Entscheidungsproblem is solved when one knows a procedure by which one can decide in a finite number of operations whether a given logical expression is generally valid or is satisfiable. The solution of the Entscheidungsproblem is of fundamental importance for the theory of all fields, the theorems of which are at all capable of logical development from finitely many axioms.

D. Hilbert, W. Ackermann
Grundzüge der theoretischen Logik, 1928

Computational Models

We have several ways to define the clone of computable functions :

- Register machines, Turing machines, Herbrand-Gödel, μ-recursive, -definable, while-computable, . . .
- Turing has an excellent plausibility argument that we really have captured intuitive computability.

The Turing machine was invented in 1936 by Alan Turing.

Turing Machine

A Turing Machine is a simplified model of an algorithm (computer program).

Example:

if read 'h', write 'f', move right

if read 'o', write 'e', move right

if read 'm' or 'e', move right

(Plus states; but these could also be encoded in the input)

Consensus is that anything that can be computed at all can be computed by a Turing Machine. (= Church Turing thesis)

Turing Machine by FOL

We can describe Turing-machines using the following predicates, functions, and constants, that have the provided standard interpretation I with its domain the set of natural numbers :

- 0 : the number 0
- s : the successor function
- $S(t, i)$: After t steps, the machine is in state i
- $C(t, i)$: After t steps, the machine is looking at cell i
- $M(t, i)$: After t steps, square i contains a 1 ('mark')

All transitions naturally translate in FOL expressions. Example :

- Transition : $\langle i, 0, 1, j \rangle$ ("if you are in state i, and see a 0, write a 1, and go to state j")
- Corresponding FOL statement :

$$\forall t\ \forall x\ [(S(t,i) \wedge C(t,x) \wedge \neg M(t,x)) \rightarrow$$
$$(S(s(t),j) \wedge C(s(t),x) \wedge M(s(t),x) \wedge$$
$$\forall y\ (y \neq x \rightarrow (M(s(t),y) \leftrightarrow M(t,y))))]$$

See `http://www.cogsci.rpi.edu/~heuveb/teaching/Logic/CompLogic/Web/Presentations/Undecidability%20of%`

Any undecidable problem ? Yes !

Problem : **halting problem**
Instance : an arbitrary program P and P's input D
Question : The decision problem is : does P halt on input D ?

That is, is there a program that takes two inputs (P and D) and stops with the answer yes if P halts on input D, or stops with the answer no if P does not halt on input D ?
If we had such a program, we could use it to check for infinite loops in programs.
Unfortunately no such program exists ; the halting problem is undecidable !

The Halting Problem is Undecidable : Proof

Proof by contradiction : Assume we have a procedure HALTS that takes as input a program P and input data D and answers yes if P halts on input D and no otherwise.

- Of course we can't just have HALTS simulate P on input D, since if P doesn't halt, we'll never know exactly when to quit the simulation and answer no.

Since there are no assumptions about the type of inputs we expect, the input D to a program P could itself be a program.

- Compilers and editors both take programs as inputs.

Given the program HALTS, we can construct a new (more limited) program that tests whether a program P halts when the input data is a copy of P.

```
procedure NEWHALTS(P);
    if HALTS(P,P) then writeln('Yes');
    else writeln('No');
```

Proof Continued

Given NEWHALTS, we can construct another program that does just the opposite of NEWHALTS :

```
procedure OPP(P);
    if NEWHALTS(P) outputs 'Yes' then
        loop forever
    else halt;
```

What happens when we call OPP(OPP)?

- Inside OPP, we call NEWHALTS(OPP), which calls HALTS(OPP,OPP). If OPP halts when fed OPP as input then the call OPP(OPP)loops forever.
- If OPP doesn't halt when fed OPP as input, then the call OPP(OPP) halts.
- OPP(OPP) can neither halt nor loop forever.

This is a contradiction! Since our only assumption was the existence of HALTS, procedure HALTS cannot exist.

The Equivalence Problem

The halting problem can be used to show that other problems are undecidable.

Equivalence Problem : Given two programs P and Q, do they compute the same function ? (ie, is $\mathrm{P}(x) = \mathrm{Q}(x)$ for all x ?) This problem is also undecidable.

$\rightsquigarrow$ Entscheidungsproblem is undecidable.

The Next Step

Decidability makes direct algorithmic sense, we are trying to test for certain properties in a computational manner.

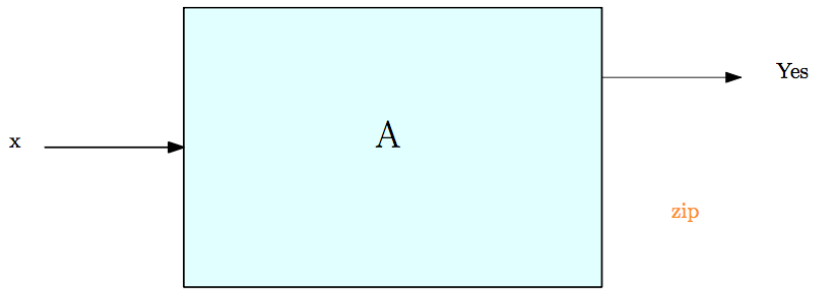
Alas, it turns out that there is a weaker property, called semidecidability, that is very closely related to decidability, and that is arguably even more fundamental and more important.

Semi-decidability

Here is a generalization of decidability :
A problem P is semidecidable if there is a turing machine that, on input x, halts if x is a yes-instance of P, and fails to halt otherwise.

We will call this a semidecision procedure.

You can think of this as a broken decision algorithm : if the answer is Yes, the algorithm works properly and stops after finitely many steps. But, if the answer is No, it just keeps running forever, it never produces a result.



Quoi ?

Note that Halting is a natural example of a semidecidable problem : we can determine convergence in finitely many steps, but divergence takes forever. This is exactly the idea behind an unbounded search : we find a witness in finitely many steps if there is one, but otherwise we keep searching forever.
Here is the critical connection between decidability and semidecidability :
Lemma A set is decidable iff the set and its complement are both semidecidable.

Proof

Clearly, A decidable implies that both A and A are semidecidable.

But the opposite direction is far from trivial : we have two semidecision procedures A0 and A1, but we do not know which one is going to halt.
The only way around this problem is to run both procedures in parallel on the given input : we have to combine two computations into one.

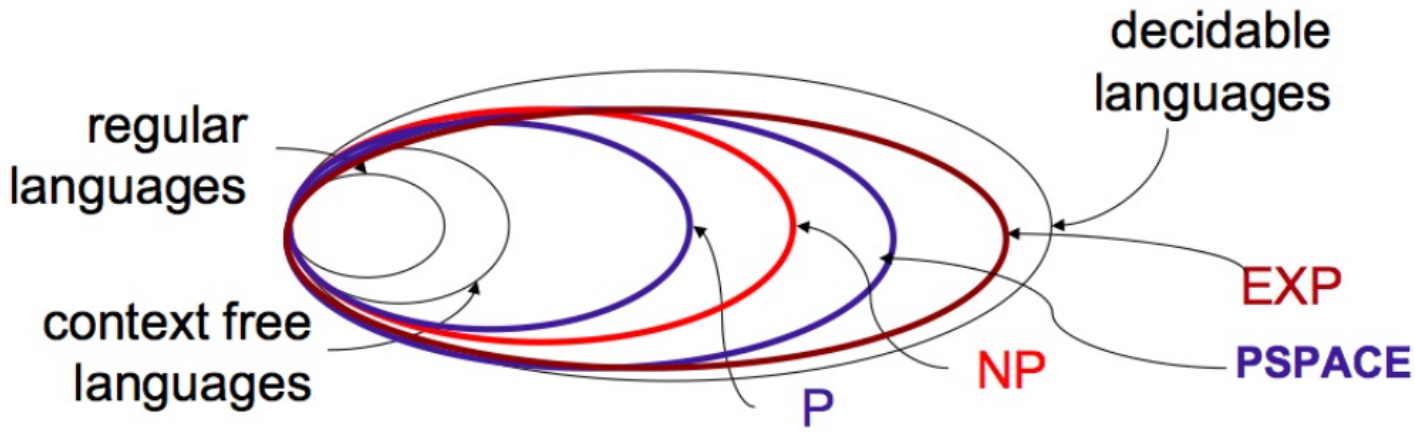
That this is possible, is another fundamental property of computation.

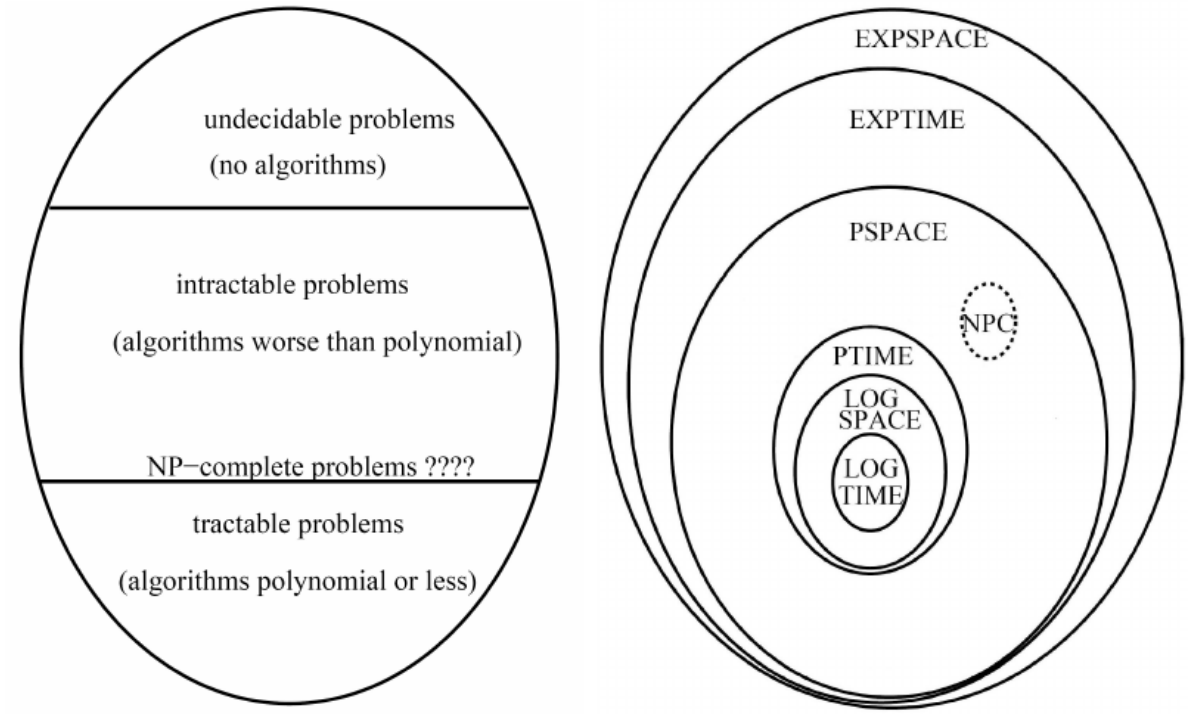
Decidable Problems ? Yes !

The Entscheidungsproblem is decidable if we consider only certain subsets of FOL.
Examples :

- Propositional logic (i.e., no quantifiers)
- Horn rules
- Bernays-Schönfinkel formulae without function symbols
- Description logics (not all !)

Complexity : Categories of Decidable Problems





Deterministic vs Non-deterministic Turing Machines

Deterministic Turing Machine (DTM) : the set of rules prescribes at most one action to be performed for any given situation.
Non-Deterministic Turing Machine(NTM) : the set of rules may prescribe more than one action to be performed for any given situation. For example, an X on the tape in state 3 might allow the NTM to :

Write a Y, move right, and switch to state 5

or

Write an X, move left, and stay in state 3.

Looking at multiple rules : imagine that the machine "branches" into many copies, each of which follows one of the possible transitions. Whereas a DTM has a single "computation path" that it follows, an NTM has a "computation tree". If at least one branch of the tree halts with an "accept" condition, we say that the NTM accepts the input.

P v.s. NP Problems

P (deterministic polynomial time) : the set of decision problems solvable in polynomial time by a theoretical non-deterministic Turing machine.

NP (nondeterministic polynomial time) : the set of decision problems solvable in polynomial time by a theoretical non-deterministic Turing machine. (E.g. $\rightsquigarrow$ SAT problem)

$$\mathrm{P} \neq \mathrm{NP} \, ? \, ? \, ?$$

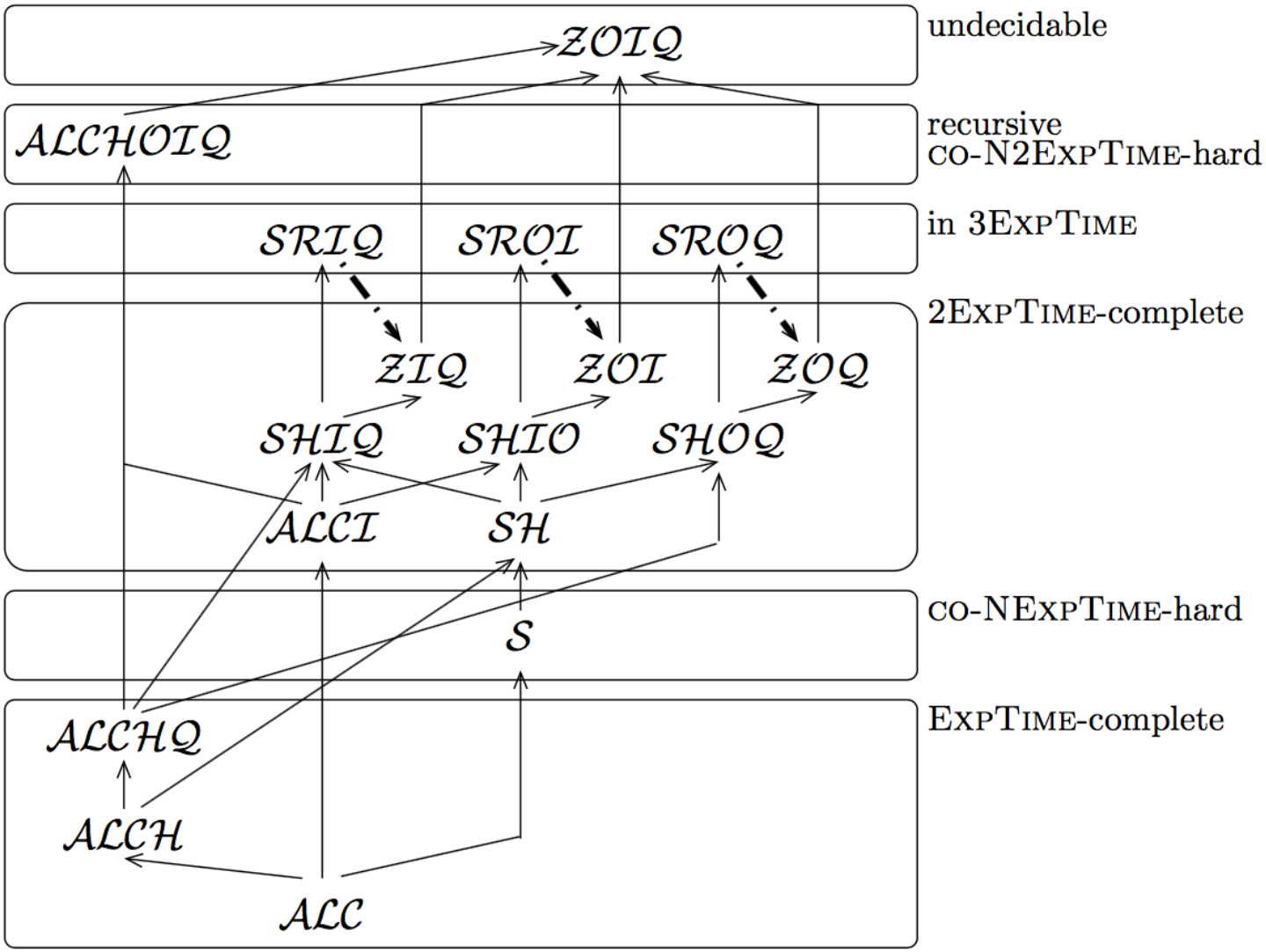
Complexities of Semantic Web Reasoning Problems

Complexity of Tractable Fragments of OWL2 (version 2006) :

Language	Reasoning Problems	Taxonomic Complexity	Data Complexity	Query Complexity	Combined Complexity
OWL DL	Ontology Consistency, Concept Satisfiability, Concept Subsumption, Instance Checking	NEXPTIME-complete	Open (NP-Hard)	Not Applicable	NEXPTIME-complete
	Conjunctive Query Answering	Open*	Open*	Open*	Open*
OWL Lite	Ontology Consistency, Concept Satisfiability	EXPTIME-complete	NP-complete	Not Applicable	EXPTIME-complete
	Concept Subsumption	EXPTIME-complete	co-NP-complete	Not Applicable	EXPTIME-complete
	Conjunctive Query Answering	EXPTIME-complete	co-NP-complete	in 2EXPTIME	in 2EXPTIME
	Instance Checking	EXPTIME-complete	co-NP-complete	Not Applicable	EXPTIME-complete
EL++	Ontology Consistency, Concept Satisfiability, Concept Subsumption, Instance Checking	PTIME-complete	PTIME-complete	Not Applicable	PTIME-complete
	Conjunctive Query Answering	Open	PTIME-hard	Open	Open
DL-Lite	Ontology Consistency, Concept Satisfiability, Concept Subsumption, Instance Checking,	In PTIME	In LOGSPACE	Not Applicable	In PTIME
	Conjunctive Query Answering	In PTIME	In LOGSPACE	NP-complete	NP-complete
DLP	Ontology Consistency, Concept Satisfiability, Concept Subsumption, Instance Checking	In EXPTIME	PTIME-complete	Not Applicable	In EXPTIME
	Conjunctive Query Answering	In EXPTIME	PTIME-complete	In EXPTIME	In EXPTIME
Horn-*SHIQ*	Ontology Consistency, Concept Satisfiability, Concept Subsumption, Instance Checking	EXPTIME-complete	PTIME-complete	Not Applicable	EXPTIME-complete
	Conjunctive Query Answering	Open	Open	Open	Open
RDF Schema	Ontology Consistency, Concept Satisfiability	Trivial	Trivial	Not Applicable	Trivial
	Concept Subsumption, Instance Checking	In PTIME	In LOGSPACE	Not Applicable	In PTIME
	Conjunctive Query Answering	In PTIME	In LOGSPACE	Open	Open

But these open questions are fully resolved for EL++, Horn-SHIQ, DL-Lite, and RDFS now.

Complexities of Semantic Web Reasoning Problems



See `https://pdfs.semanticscholar.org/1585/e5212ccd15685277d058d0a0e8603bba343f.pdf`

Complexities of Semantic Web Reasoning Problems

	Combined complexity		Data complexity	
	instance queries / standard reasoning	conjunctive queries	instance queries / standard reasoning	conjunctive queries
Plain databases		NP-complete [2]		in AC_0 [54]
DL-Lite	in P [13]	NP-complete [13]	in AC_0 [13]	in AC_0 [13]
$\mathcal{EL}$, $\mathcal{ELH}$	P-complete [6]	NP-complete [87,59]	P-complete [6]	P-complete [87,57,59]
$\mathcal{EL}^+$, $\mathcal{EL}^{++}$	P-complete [6]	undecidable [87,57,59] (PSPACE-complete for *regular*-$\mathcal{EL}^{++}$ [59])	P-complete [6]	undecidable [87,57,59] (P-complete for *regular*-$\mathcal{EL}^{++}$ [59])
Horn-$\mathcal{SHIQ}$, Horn-$\mathcal{SHOIQ}$	EXPTIME-complete (Horn-$\mathcal{SHIQ}$ [60], Horn-$\mathcal{SHOIQ}$ [77])	EXPTIME-complete (Horn-$\mathcal{SHIQ}$ [29], Horn-$\mathcal{SHOIQ}$ [78])	P-complete (Horn-$\mathcal{SHIQ}$ [53], Horn-$\mathcal{SHOIQ}$ [78])	P-complete (Horn-$\mathcal{SHIQ}$ [60], Horn-$\mathcal{SHOIQ}$ [77])
Horn-$\mathcal{SRIQ}$	in 2EXPTIME [77]	in 2EXPTIME [78]	P-complete [78]	P-complete [78]
Horn-$\mathcal{SROIQ}$	2EXPTIME-complete [77]	2EXPTIME-complete [78]	P-complete [78]	P-complete [78]
$\mathcal{ALC}$, $\mathcal{ALCHQ}$	EXPTIME-complete [92,70]	EXPTIME-complete [66,79]	CONP-complete l.b.[91], u.b.[75]	CONP-complete l.b.[91], u.b.[75]
$\mathcal{ALCI}$, $\mathcal{SH}$, $\mathcal{ALCHIQ}$, $\mathcal{SHIQ}$,$\mathcal{SHOQ}$,$\mathcal{SHIO}$ $\mathcal{ZIQ}$,$\mathcal{ZOQ}$,$\mathcal{ZOI}$	EXPTIME-complete (cf. [95,23] and references therein, see also [7])	2EXPTIME-complete (l.b. $\mathcal{ALCI}$ [66], $\mathcal{SH}$ [31]; u.b. $\mathcal{ZIQ}$, $\mathcal{ZOQ}$, $\mathcal{ZOI}$ [23], $\mathcal{SHIQ}$ [36,22], $\mathcal{SHOQ}$ [37], $\mathcal{ALCHIQ}$ [52], cf. Sect. 8)	CONP-hard [91], CONP-complete for many DLs ($\mathcal{SHIQ}$ [53], $\mathcal{SHOQ}$, $\mathcal{SHOI}$ [75])	CONP-hard [91], CONP-complete for many DLs ($\mathcal{ALCHIQ}$ [53], $\mathcal{SHIQ}$ [36] $\mathcal{ALCHOQ}$, $\mathcal{ALCHOI}$ [75])
$\mathcal{SRIQ}$	2EXPTIME-complete l.b.[55],u.b.[23]	in 3EXPTIME [23]	CONP-complete (follows from [85] and [55])	CONP-complete [85]
$\mathcal{ALCHOIQ}$	NEXPTIME-complete [94]	decidable [89], CO-N2EXPTIME-hard [39]	CONP-complete [85]	decidable [89]
$\mathcal{SHOIQ}$	NEXPTIME-complete [94]	open	CONP-complete [85]	open
$\mathcal{SROIQ}$	N2EXPTIME-complete [55]	open	CONP-complete (follows from [85] and [55])	open
$\mathcal{ZOIQ}$	open	undecidable [76]	open	undecidable [76]

See `https://pdfs.semanticscholar.org/1585/e5212ccd15685277d058d0a0e8603bba343f.pdf`

Outline

Solving Problems

Decidability, Semi-decidability, Complexties

Expressive and Lightweight DLs

Expressive DLs

Some expressive DLs between ALC and SHOIQ :

DL	TAs	RIAs	inverses	nominals	NRs
$\mathcal{ALC}$					
$\mathcal{ALCI}$			✓		
$\mathcal{ALCHQ}$		✓			✓
$\mathcal{SH}$	✓	✓			
$\mathcal{SHIQ}$	✓	✓	✓		✓
$\mathcal{SHOQ}$	✓	✓		✓	✓
$\mathcal{SHOI}$	✓	✓	✓	✓	
$\mathcal{ALCHOIQ}$		✓	✓	✓	✓
$\mathcal{SHOIQ}$	✓	✓	✓	✓	✓

- TA (transitivity axioms) : trans(r) for a role r
- RIA (role hierachys) : $r \sqsubseteq s$ for roles r, s
- inverse : r^-
- nominals : $\{o_1, o_2, ...o_n\}$
- NRs ((qualified) number restrictions) : $\geq nR.C, \leq nR.C$

Lightweight DLs

Motivation for lightweight DLs:

- good computational properties (reasoning in PTIME)
- limited (but useful) expressivity

Lightweight DLs

Motivation for lightweight DLs:

- good computational properties (reasoning in PTIME)
- limited (but useful) expressivity

We will consider the two main families of lightweight DLs:

- the $\mathcal{EL}$ family
  - designed to handle large TBoxes
  - key tasks: classification, instance checking
- the DL-Lite family
  - designed to handle large ABoxes
  - key task: conjunctive query answering

Lightweight DLs

Motivation for lightweight DLs:

- good computational properties (reasoning in PTIME)
- limited (but useful) expressivity

We will consider the two main families of lightweight DLs:

- the $\mathcal{EL}$ family — basis for OWL 2 EL profile
  - designed to handle large TBoxes
  - key tasks: classification, instance checking
- the DL-Lite family — basis for OWL 2 QL profile
  - designed to handle large ABoxes
  - key task: conjunctive query answering

1. $\mathcal{EL}$

The $\mathcal{EL}$ family

The logic $\mathcal{EL}$, and its extensions, are designed for applications requiring very large ontologies.

This family of DLs is well-suited for biomedical applications.

Examples of large biomedical ontologies:

- GO (Gene Ontology), around 20,000 concepts
- NCI (cancer ontology), around 30,000 concepts
- SNOMED (medical ontology), around 400,000 concepts (!)

$$\mathsf{Pericarditis} \sqsubseteq \mathsf{Inflammation} \sqcap \exists \mathsf{loc}.\mathsf{Pericardium}$$
$$\mathsf{Pericardium} \sqsubseteq \mathsf{Tissue} \sqcap \exists \mathsf{partOf}.\mathsf{Heart} \qquad \mathsf{Inflammation} \sqsubseteq \mathsf{Disease}$$
$$\mathsf{Disease} \sqcap \exists \mathsf{loc}.\exists \mathsf{partOf}.\mathsf{Heart} \sqsubseteq \mathsf{HeartDisease}$$

More about SNOMED: http://www.ihtsdo.org/snomed-ct/

Syntax of $\mathcal{EL}$

The basic logic $\mathcal{EL}$ allows complex concepts of the following form:

$$C := \top \mid A \mid C_1 \sqcap C_2 \mid \exists R.C$$

Inclusions $C_1 \sqsubseteq C_2$ and assertions $A(c)$, $R(c, d)$

Possible extensions:

- $\bot$ (to express disjoint classes)
- domain restrictions $\mathrm{dom}(R) \sqsubseteq C$
- range restrictions $\mathrm{range}(R) \sqsubseteq C$
- complex role inclusions $R_1 \circ \ldots \circ R_n \sqsubseteq R_{n+1}$ (transitivity: $R \circ R \sqsubseteq R$)

OWL 2 EL includes all these extensions.

Today we will focus on plain $\mathcal{EL}$ (without these extensions).

DL-Lite

Using ontologies to access data

Objective: enrich standard relational databases (DBs) with ontologies

- ontology provides a convenient vocabulary for users to specify queries
- ontology can be used to link multiple datasets with different schemas
- knowledge in ontology can yield additional answers to queries

Using ontologies to access data

Objective: enrich standard relational databases (DBs) with ontologies

- ontology provides a convenient vocabulary for users to specify queries
- ontology can be used to link multiple datasets with different schemas
- knowledge in ontology can yield additional answers to queries

Desiderata:

- efficiency is crucial – must scale up to huge datasets

Using ontologies to access data

Objective: enrich standard relational databases (DBs) with ontologies

- ontology provides a convenient vocabulary for users to specify queries
- ontology can be used to link multiple datasets with different schemas
- knowledge in ontology can yield additional answers to queries

Desiderata:

- efficiency is crucial – must scale up to huge datasets
- instance queries too simple – want expressive queries like in DBs
  - conjunctive queries $\sim$ select-project-join queries in SQL

Using ontologies to access data

Objective: enrich standard relational databases (DBs) with ontologies

- ontology provides a convenient vocabulary for users to specify queries
- ontology can be used to link multiple datasets with different schemas
- knowledge in ontology can yield additional answers to queries

Desiderata:

- efficiency is crucial – must scale up to huge datasets
- instance queries too simple – want expressive queries like in DBs
  - conjunctive queries $\sim$ select-project-join queries in SQL

DL-Lite family: designed for efficient conjunctive query answering

Syntax of *DL-Lite*

We present the dialect *DL-Lite*$_{\mathcal{R}}$ (which underlies OWL2 QL).

ABox assertions: $A(c)$, $R(c, d)$

TBox inclusions: $B_1 \sqsubseteq B_2$, $B_1 \sqsubseteq \neg B_2$, $S_1 \sqsubseteq S_2$, $S_1 \sqsubseteq \neg S_2$ where

$$B := A \mid \exists S \qquad\qquad S := R \mid R^{-}$$

with A an atomic concept and R an atomic role

Syntax of *DL-Lite*

We present the dialect *DL-Lite*$_{\mathcal{R}}$ (which underlies OWL2 QL).

ABox assertions: $A(c)$, $R(c, d)$

TBox inclusions: $B_1 \sqsubseteq B_2$, $B_1 \sqsubseteq \neg B_2$, $S_1 \sqsubseteq S_2$, $S_1 \sqsubseteq \neg S_2$ where

$$B := A \mid \exists S \qquad\qquad S := R \mid R^{-}$$

with A an atomic concept and R an atomic role

Other *DL-Lite* dialects allow:

- functional roles (funct S)
- cardinality restrictions ($\geq q\, S$, $\leq q\, S$)
- Horn inclusions ($B_1 \sqcap \ldots \sqcap B_n \sqsubseteq (\neg) B_{n+1}$)
- roles which are symmetric, asymmetric, reflexive, or anti-reflexive

Example: Describing families in $DL\text{-}Lite_{\mathcal{R}}$

- subclasses

$$Mother \sqsubseteq Parent \qquad Father \sqsubseteq Parent \qquad Mother \sqsubseteq Female$$

- subrelations

$$MotherOf \sqsubseteq ParentOf \qquad FatherOf \sqsubseteq ParentOf$$

- domain and range constraints

$$\exists ParentOf \sqsubseteq Parent \qquad \exists ChildOf^{-} \sqsubseteq Parent$$

- participation constraints

$$Parent \sqsubseteq \exists ChildOf^{-} \qquad Mother \sqsubseteq \exists MotherOf$$

- inverses of binary relations

$$ParentOf \sqsubseteq ChildOf^{-} \qquad ChildOf \sqsubseteq ParentOf^{-}$$

- disjointness

$$Male \sqsubseteq \neg Female \qquad Childless \sqsubseteq \neg \exists ParentOf$$

Conjunctive queries

An atom takes the form $P(t_1, \ldots, t_n)$ where:

- P is a relation symbol
- each t_i is a term (= variable or individual name)

A conjunctive query (CQ) has the form

$$\exists x_1, \ldots, x_m \;\; \alpha_1 \wedge \ldots \wedge \alpha_r$$

where each α_i is an atom, and each x_i appears in some atom.

- $x_1, \ldots, x_m$ are called quantified (or existential) variables
- all other variables are called answer variables.

Since we consider DLs, relation symbols are atomic concept and roles.

Example: conjunctive queries

Find all pairs of teachers such that the first is the parent of the second:

$$Teacher(x) \wedge ParentOf(x, y) \wedge Teacher(y)$$

Find all teachers that have a child who is a teacher:

$$\exists y \; Teacher(x) \wedge ParentOf(x, y) \wedge Teacher(y)$$

Find all teachers that have a parent who is a teacher:

$$\exists x \; Teacher(x) \wedge ParentOf(x, y) \wedge Teacher(y)$$

Does there exist a teacher with a child who is a teacher?

$$\exists x, y \; Teacher(x) \wedge ParentOf(x, y) \wedge Teacher(y)$$

Does Marie have a child who is a teacher?

$$\exists y \; ParentOf(marie, y) \wedge Teacher(y)$$

Semantics of conjunctive queries (1)

Boolean CQ = CQ that has no answer variables

Satisfaction of a Boolean CQ in an interpretation:

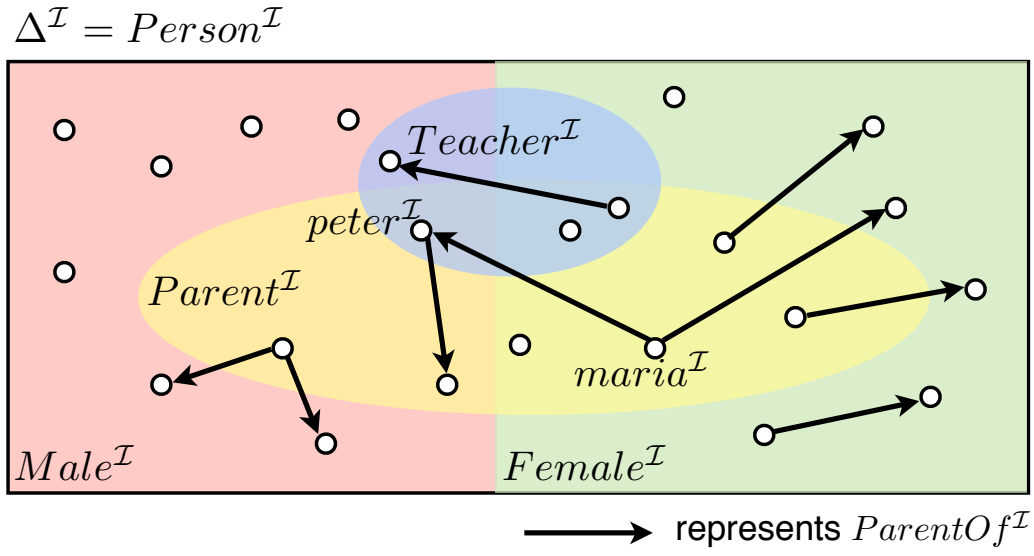
Interpretation $\mathcal{I}$ satisfies a Boolean CQ q if there exists a function π mapping each term of q to an element of $\Delta^{\mathcal{I}}$ such that:

- for every individual a in q, $\pi(a) = a^{\mathcal{I}}$
- for every atom $\alpha = P(t_1, \ldots, t_k)$ of q,

$$(\pi(t_1), \ldots, \pi(t_k)) \in P^{\mathcal{I}}$$

Example: Satisfaction in an interpretation

Reconsider the interpretation $\mathcal{I}$:



Which of the following Boolean CQs are satisfied in $\mathcal{I}$?

- $\exists x\, ParentOf(x, peter) \wedge Female(x)$
- $\exists x\, ParentOf(peter, x) \wedge Female(x)$
- $\exists x\, ParentOf(maria, x) \wedge Female(x)$
- $ParentOf(peter, maria)$
- $\exists x, y\, Male(x) \wedge ParentOf(x, y) \wedge Female(y)$
- $\exists x\, ParentOf(x, x)$
- $\exists x, y\, Teacher(x) \wedge ParentOf(x, y) \wedge Teacher(y)$
- $\exists x\, Teacher(x) \wedge Female(x)$

Semantics of conjunctive queries (2)

Entailment of a Boolean CQ:

Boolean CQ q is entailed from $\mathcal{K}$ (written $\mathcal{K} \models q$) if and only if every model of $\mathcal{K}$ satisfies q.

Certain answers to a CQ:

Suppose q has answer variables $x_1, \ldots, x_k$. A tuple $(a_1, \ldots, a_k)$ of individuals is a (certain) answer to q w.r.t. $\mathcal{K}$ if and only if

$$\mathcal{K} \models q[a_1, \ldots, a_k]$$

where $q[a_1, \ldots, a_k]$ is q with every x_i replaced by a_i.

We denote by $\mathsf{cert}(q, \mathcal{K})$ the certain answers to q w.r.t. $\mathcal{K}$

Example: Certain answers

Consider the TBox:

$\mathit{Parent} \sqsubseteq \exists \mathit{ParentOf} \quad \exists \mathit{ParentOf} \sqsubseteq \mathit{Parent} \quad \mathit{Mother} \sqsubseteq \mathit{Parent} \quad \mathit{Father} \sqsubseteq \mathit{Parent}$

$\mathit{Mother} \sqsubseteq \mathit{Female} \quad \mathit{Father} \sqsubseteq \mathit{Male} \quad \mathit{ParentOf} \sqsubseteq \mathit{ChildOf}^- \quad \mathit{ChildOf}^- \sqsubseteq \mathit{ParentOf}$

and the ABox:

$\mathit{Mother}(\mathit{mary}) \quad \mathit{Father}(\mathit{paul}) \quad \mathit{ParentOf}(\mathit{julie}, \mathit{marc}) \quad \mathit{ParentOf}(\mathit{marc}, \mathit{paul})$

Determine the certain answers to the following queries.

- $\mathit{Female}(x)$
- $\mathit{ChildOf}(x, y)$
- $\exists y\, \mathit{ParentOf}(x, y)$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(y, z)$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(z, y)$

Example: Certain answers

Consider the TBox:

$\mathit{Parent} \sqsubseteq \exists \mathit{ParentOf}$ $\exists \mathit{ParentOf} \sqsubseteq \mathit{Parent}$ $\mathit{Mother} \sqsubseteq \mathit{Parent}$ $\mathit{Father} \sqsubseteq \mathit{Parent}$

$\mathit{Mother} \sqsubseteq \mathit{Female}$ $\mathit{Father} \sqsubseteq \mathit{Male}$ $\mathit{ParentOf} \sqsubseteq \mathit{ChildOf}^-$ $\mathit{ChildOf}^- \sqsubseteq \mathit{ParentOf}$

and the ABox:

$\mathit{Mother}(\mathit{mary})$ $\mathit{Father}(\mathit{paul})$ $\mathit{ParentOf}(\mathit{julie}, \mathit{marc})$ $\mathit{ParentOf}(\mathit{marc}, \mathit{paul})$

Determine the certain answers to the following queries.

- $\mathit{Female}(x)$ mary
- $\mathit{ChildOf}(x, y)$
- $\exists y\, \mathit{ParentOf}(x, y)$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(y, z)$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(z, y)$

Example: Certain answers

Consider the TBox:

$\mathit{Parent} \sqsubseteq \exists \mathit{ParentOf}$ $\exists \mathit{ParentOf} \sqsubseteq \mathit{Parent}$ $\mathit{Mother} \sqsubseteq \mathit{Parent}$ $\mathit{Father} \sqsubseteq \mathit{Parent}$

$\mathit{Mother} \sqsubseteq \mathit{Female}$ $\mathit{Father} \sqsubseteq \mathit{Male}$ $\mathit{ParentOf} \sqsubseteq \mathit{ChildOf}^-$ $\mathit{ChildOf}^- \sqsubseteq \mathit{ParentOf}$

and the ABox:

$\mathit{Mother}(\mathit{mary})$ $\mathit{Father}(\mathit{paul})$ $\mathit{ParentOf}(\mathit{julie}, \mathit{marc})$ $\mathit{ParentOf}(\mathit{marc}, \mathit{paul})$

Determine the certain answers to the following queries.

- $\mathit{Female}(x)$ mary
- $\mathit{ChildOf}(x, y)$ $(\mathit{marc}, \mathit{julie}), (\mathit{paul}, \mathit{marc})$
- $\exists y\, \mathit{ParentOf}(x, y)$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(y, z)$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(z, y)$

Example: Certain answers

Consider the TBox:

$\mathit{Parent} \sqsubseteq \exists \mathit{ParentOf} \quad \exists \mathit{ParentOf} \sqsubseteq \mathit{Parent} \quad \mathit{Mother} \sqsubseteq \mathit{Parent} \quad \mathit{Father} \sqsubseteq \mathit{Parent}$

$\mathit{Mother} \sqsubseteq \mathit{Female} \quad \mathit{Father} \sqsubseteq \mathit{Male} \quad \mathit{ParentOf} \sqsubseteq \mathit{ChildOf}^- \quad \mathit{ChildOf}^- \sqsubseteq \mathit{ParentOf}$

and the ABox:

$\mathit{Mother}(\mathit{mary}) \quad \mathit{Father}(\mathit{paul}) \quad \mathit{ParentOf}(\mathit{julie}, \mathit{marc}) \quad \mathit{ParentOf}(\mathit{marc}, \mathit{paul})$

Determine the certain answers to the following queries.

- $\mathit{Female}(x)$ $\quad \mathit{mary}$
- $\mathit{ChildOf}(x, y)$ $\quad (\mathit{marc}, \mathit{julie}), (\mathit{paul}, \mathit{marc})$
- $\exists y\, \mathit{ParentOf}(x, y)$ $\quad \mathit{julie}, \mathit{marc}, \mathit{mary}, \mathit{paul}$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(y, z)$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(z, y)$

Example: Certain answers

Consider the TBox:

$\mathit{Parent} \sqsubseteq \exists \mathit{ParentOf}$ $\exists \mathit{ParentOf} \sqsubseteq \mathit{Parent}$ $\mathit{Mother} \sqsubseteq \mathit{Parent}$ $\mathit{Father} \sqsubseteq \mathit{Parent}$

$\mathit{Mother} \sqsubseteq \mathit{Female}$ $\mathit{Father} \sqsubseteq \mathit{Male}$ $\mathit{ParentOf} \sqsubseteq \mathit{ChildOf}^-$ $\mathit{ChildOf}^- \sqsubseteq \mathit{ParentOf}$

and the ABox:

$\mathit{Mother}(\mathit{mary})$ $\mathit{Father}(\mathit{paul})$ $\mathit{ParentOf}(\mathit{julie}, \mathit{marc})$ $\mathit{ParentOf}(\mathit{marc}, \mathit{paul})$

Determine the certain answers to the following queries.

- $\mathit{Female}(x)$ mary
- $\mathit{ChildOf}(x, y)$ $(\mathit{marc}, \mathit{julie}), (\mathit{paul}, \mathit{marc})$
- $\exists y\, \mathit{ParentOf}(x, y)$ $\mathit{julie}, \mathit{marc}, \mathit{mary}, \mathit{paul}$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(y, z)$ $(\mathit{julie}, \mathit{marc}, \mathit{julie}), (\mathit{marc}, \mathit{paul}, \mathit{marc})$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(z, y)$

Example: Certain answers

Consider the TBox:

$\mathit{Parent} \sqsubseteq \exists \mathit{ParentOf}$ $\exists \mathit{ParentOf} \sqsubseteq \mathit{Parent}$ $\mathit{Mother} \sqsubseteq \mathit{Parent}$ $\mathit{Father} \sqsubseteq \mathit{Parent}$

$\mathit{Mother} \sqsubseteq \mathit{Female}$ $\mathit{Father} \sqsubseteq \mathit{Male}$ $\mathit{ParentOf} \sqsubseteq \mathit{ChildOf}^-$ $\mathit{ChildOf}^- \sqsubseteq \mathit{ParentOf}$

and the ABox:

$\mathit{Mother}(\mathit{mary})$ $\mathit{Father}(\mathit{paul})$ $\mathit{ParentOf}(\mathit{julie}, \mathit{marc})$ $\mathit{ParentOf}(\mathit{marc}, \mathit{paul})$

Determine the certain answers to the following queries.

- $\mathit{Female}(x)$ mary
- $\mathit{ChildOf}(x, y)$ $(\mathit{marc}, \mathit{julie}), (\mathit{paul}, \mathit{marc})$
- $\exists y\, \mathit{ParentOf}(x, y)$ $\mathit{julie}, \mathit{marc}, \mathit{mary}, \mathit{paul}$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(y, z)$ $(\mathit{julie}, \mathit{marc}, \mathit{julie}), (\mathit{marc}, \mathit{paul}, \mathit{marc})$
- $\mathit{ParentOf}(x, y) \wedge \mathit{ChildOf}(z, y)$ $(\mathit{julie}, \mathit{marc}, \mathit{paul})$