A Multi-Engine Theorem Prover for a Description Logic of Typicality

Laura Giordano [1] Valentina Gliozzi [2] Nicola Olivetti [3] Gian Luca Pozzato [2] Luca Violanti [4]

[1] DISIT - Universitá Piemonte Orientale - Alessandria, Italy
[2] Dipartimento di Informatica, Universitá degli Studi di Torino, Italy
[3] Aix Marseille Univ. - CNRS, ENSAM, Univ. de Toulon - LSIS UMR 7296, Marseille - France
[4] NCR Edinburgh - United Kingdom

AI*IA 2015

Description Logics

Description Logics

- Important formalisms of knowledge representation
- Two key advantages:
  - well-defined semantics based on first-order logic
  - good trade-off between expressivity and complexity
- at the base of languages for the semantic (e.g. OWL)

Knowledge bases

- Two components:
  - TBox= inclusion relations among concepts
  - ABox= instances of concepts and roles = properties and relations among individuals

Description Logics

Description Logics

- Important formalisms of knowledge representation
- Two key advantages:
  - well-defined semantics based on first-order logic
  - good trade-off between expressivity and complexity
- at the base of languages for the semantic (e.g. OWL)

Knowledge bases

- Two components:
  - TBox=inclusion relations among concepts
    - $Platypus \sqsubseteq Mammal$
  - ABox= instances of concepts and roles = properties and relations among individuals
    - $Platypus(perry)$

Description Logics

Description Logics

- Important formalisms of knowledge representation
- Two key advantages:
  - well-defined semantics based on first-order logic
  - good trade-off between expressivity and complexity
- at the base of languages for the semantic (e.g. OWL)

Knowledge bases

- Two components:
  - TBox=inclusion relations among concepts
    - $Platypus \sqsubseteq Mammal$
  - ABox= instances of concepts and roles = properties and relations among individuals
    - $Platypus(perry)$

Description Logics

Description Logics

- Important formalisms of knowledge representation
- Two key advantages:
  - well-defined semantics based on first-order logic
  - good trade-off between expressivity and complexity
- at the base of languages for the semantic (e.g. OWL)

Knowledge bases

- Two components:
  - TBox=inclusion relations among concepts
    - $Platypus \sqsubseteq Mammal$
  - ABox= instances of concepts and roles = properties and relations among individuals
    - $Platypus(perry)$

Description Logics

Reasoning

- TBox = taxonomy of concepts
- need of representing prototypical properties and of reasoning about defeasible inheritance
- integration with nonmonotonic reasoning mechanism to handle defeasible inheritance [BH95, BLW06, DLN+98, DNR02, ELST04, Str93]
- all these methods present some difficulties

Our solution

- DLs + typicality operator $\mathbf{T}$ for defeasible reasoning in DLs [GGOP13]
- meaning of $\mathbf{T}$: (for any concept C) $\mathbf{T}(C)$ singles out the "typical" instances of C
- semantics of $\mathbf{T}$ defined by a set of postulates that are a restatement of Kraus-Lehmann-Magidor axioms of preferential logic $\mathbf{P}$

Description Logics

Reasoning

- TBox = taxonomy of concepts
- need of representing prototypical properties and of reasoning about defeasible inheritance
- integration with nonmonotonic reasoning mechanism to handle defeasible inheritance [BH95, BLW06, DLN$^+$98, DNR02, ELST04, Str93]
- all these methods present some difficulties

Our solution

- DLs + typicality operator **T** for defeasible reasoning in DLs [GGOP13]
- meaning of **T**: (for any concept C) $\mathbf{T}(C)$ singles out the "typical" instances of C
- semantics of **T** defined by a set of postulates that are a restatement of Kraus-Lehmann-Magidor axioms of preferential logic **P**

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Basic notions

- A KB comprises assertions $\mathbf{T}(C) \sqsubseteq D$
- $\mathbf{T}(Student) \sqsubseteq FacebookUsers$ means "normally, students use Facebook"
- $\mathbf{T}$ is nonmonotonic
  - $C \sqsubseteq D$ does not imply $\mathbf{T}(C) \sqsubseteq \mathbf{T}(D)$

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Basic notions

- A KB comprises assertions $\mathbf{T}(C) \sqsubseteq D$
- $\mathbf{T}(Student) \sqsubseteq FacebookUsers$ means "normally, students use Facebook"
- $\mathbf{T}$ is nonmonotonic
  - $C \sqsubseteq D$ does not imply $\mathbf{T}(C) \sqsubseteq \mathbf{T}(D)$

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Example

$$\mathbf{T}(BasketballPlayer) \sqsubseteq \neg Rich$$
$$\mathbf{T}(BasketballPlayer \sqcap NBAMember) \sqsubseteq Rich$$

Reasoning

- ABox:

$$BasketballPlayer(marco)$$

- Expected conclusions:

$$\neg Rich(marco)$$

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Example

$\mathbf{T}(BasketballPlayer) \sqsubseteq \neg Rich$

$\mathbf{T}(BasketballPlayer \sqcap NBAMember) \sqsubseteq Rich$

Reasoning

- ABox:
  - $BasketballPlayer(marco)$
- Expected conclusions:
  - $\neg Rich(marco)$

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Example

$$\mathbf{T}(BasketballPlayer) \sqsubseteq \neg Rich$$
$$\mathbf{T}(BasketballPlayer \sqcap NBAMember) \sqsubseteq Rich$$

Reasoning

- ABox:
  - $BasketballPlayer(marco)$
- Expected conclusions:
  - $\neg Rich(marco)$

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Example

$\mathbf{T}(BasketballPlayer) \sqsubseteq \neg Rich$

$\mathbf{T}(BasketballPlayer \sqcap NBAMember) \sqsubseteq Rich$

Reasoning

- ABox:
  - $BasketballPlayer(marco)$
  - $NBAMember(marco)$
- Expected conclusions:
  - $Rich(marco)$

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Example

$\mathbf{T}(BasketballPlayer) \sqsubseteq \neg Rich$

$\mathbf{T}(BasketballPlayer \sqcap NBAMember) \sqsubseteq Rich$

Reasoning

- ABox:
  - $BasketballPlayer(marco)$
  - $NBAMember(marco)$
- Expected conclusions:
  - $Rich(marco)$

The logic $\mathcal{ALC} + \mathbf{T}_{min}$

Example

$$\mathbf{T}(BasketballPlayer) \sqsubseteq \neg Rich$$
$$\mathbf{T}(BasketballPlayer \sqcap NBAMember) \sqsubseteq Rich$$

Reasoning

- ABox:
  - $BasketballPlayer(marco)$
  - $NBAMember(marco)$
- Expected conclusions:
  - $Rich(marco)$

Weakness of monotonic semantics

Logic $\mathcal{ALC} + \mathbf{T}$

- The operator $\mathbf{T}$ is nonmonotonic, but...
- The logic is monotonic
  - If KB $\models F$, then KB' $\models F$ for all KB' $\supseteq$ KB

Example

- in the KB of the previous slides:
  - if *BasketballPlayer*(*marco*) $\in$ ABox, we are not able to:

Weakness of monotonic semantics

Logic $\mathcal{ALC} + \mathbf{T}$

- The operator $\mathbf{T}$ is nonmonotonic, but...
- The logic is monotonic
  - If KB $\models F$, then KB' $\models F$ for all KB' $\supseteq$ KB

Example

- in the KB of the previous slides:
  - if $BasketballPlayer(marco) \in$ ABox, we are not able to:
    - assume that $\mathbf{T}(BasketballPlayer)(marco)$
    - infer that $\neg Rich(marco)$

Weakness of monotonic semantics

Logic $\mathcal{ALC} + \mathbf{T}$

- The operator $\mathbf{T}$ is nonmonotonic, but...
- The logic is monotonic
  - If KB $\models F$, then KB' $\models F$ for all KB' $\supseteq$ KB

Example

- in the KB of the previous slides:
  - if $\mathit{BasketballPlayer}(\mathit{marco}) \in$ ABox, we are not able to:
    - assume that $\mathbf{T}(\mathit{BasketballPlayer})(\mathit{marco})$
    - infer that $\neg \mathit{Rich}(\mathit{marco})$

Weakness of monotonic semantics

Logic $\mathcal{ALC} + \mathbf{T}$

- The operator $\mathbf{T}$ is nonmonotonic, but...
- The logic is monotonic
  - If KB $\models F$, then KB' $\models F$ for all KB' $\supseteq$ KB

Example

- in the KB of the previous slides:
  - if $BasketballPlayer(marco) \in$ ABox, we are not able to:
    - assume that $\mathbf{T}(BasketballPlayer)(marco)$
    - infer that $\neg Rich(marco)$

The nonmonotonic logic $\mathcal{ALC} + \mathbf{T}_{min}$ [GGOP13]

Minimal entailment

- Preference relation among models of a KB
  - $\mathcal{M}_1 < \mathcal{M}_2$ if $\mathcal{M}_1$ contains less exceptional (not minimal) elements
  - $\mathcal{M}$ minimal model of KB if there is no $\mathcal{M}'$ model of KB such that $\mathcal{M}' < \mathcal{M}$
- Minimal entailment
  - KB $\models_{min} F$ if F holds in all *minimal* models of KB
- Nonmonotonic logic
  - KB $\models_{min} F$ does not imply KB' $\models_{min} F$ with KB' $\supset$ KB

The nonmonotonic logic $\mathcal{ALC} + \mathbf{T}_{min}$ [GGOP13]

Minimal entailment

- Preference relation among models of a KB
  - $\mathcal{M}_1 < \mathcal{M}_2$ if $\mathcal{M}_1$ contains less exceptional (not minimal) elements
  - $\mathcal{M}$ minimal model of KB if there is no $\mathcal{M}'$ model of KB such that $\mathcal{M}' < \mathcal{M}$
- Minimal entailment
  - KB $\models_{min}$ F if F holds in all *minimal* models of KB
- Nonmonotonic logic
  - KB $\models_{min}$ F does not imply KB' $\models_{min}$ F with KB' $\supset$ KB

The nonmonotonic logic $\mathcal{ALC} + \mathbf{T}_{min}$ [GGOP13]

Minimal entailment

- Preference relation among models of a KB
  - $\mathcal{M}_1 < \mathcal{M}_2$ if $\mathcal{M}_1$ contains less exceptional (not minimal) elements
  - $\mathcal{M}$ minimal model of KB if there is no $\mathcal{M}'$ model of KB such that $\mathcal{M}' < \mathcal{M}$
- Minimal entailment
  - KB $\models_{min} F$ if F holds in all *minimal* models of KB
- Nonmonotonic logic
  - KB $\models_{min} F$ does not imply KB' $\models_{min} F$ with KB' $\supset$ KB

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$

Basic ideas

- for deciding whether a query F is minimally entailed from a KB
- two-phase computation:
  - Phase 1: verifies whether KB $\cup \{\neg F\}$ is satisfiable building candidate models
  - Phase 2: checks whether candidate models found in Phase 1 are *minimal*
- More precisely: if, for each branch **B** built by Phase 1, either:
  - **B** is closed or
  - the tableau built by Phase 2 is open,
- then the procedure says YES else the procedure says NO

Design of `DysToPic`

Basic concepts

- multi-engine theorem prover for reasoning in $\mathcal{ALC} + \mathbf{T}_{min}$
- SICStus Prolog implementation of the two-phases tableaux calculus wrapped by a Java interface which relies on the Java RMI APIs for the distribution of the computation
- "worker/employer" paradigm
  - the computational burden for the "employer" can be spread among an arbitrarily high number of "workers" which operate in complete autonomy, so that they can be either deployed on a single machine or on a computer grid

Design of `DysToPic`

Basic concepts

- multi-engine theorem prover for reasoning in $\mathcal{ALC} + \mathbf{T}_{min}$
- SICStus Prolog implementation of the two-phases tableaux calculus wrapped by a Java interface which relies on the Java RMI APIs for the distribution of the computation
- "worker/employer" paradigm
  - the computational burden for the "employer" can be spread among an arbitrarily high number of "workers" which operate in complete autonomy, so that they can be either deployed on a single machine or on a computer grid

Design of `DysToPic`

Basic concepts

- multi-engine theorem prover for reasoning in $\mathcal{ALC} + \mathbf{T}_{min}$
- SICStus Prolog implementation of the two-phases tableaux calculus wrapped by a Java interface which relies on the Java RMI APIs for the distribution of the computation
- "worker/employer" paradigm
  - the computational burden for the "employer" can be spread among an arbitrarily high number of "workers" which operate in complete autonomy, so that they can be either deployed on a single machine or on a computer grid

Design of `DysToPic`

Ideas

- no need for Phase 1 to wait for the result of one elaboration of Phase 2 on an open branch, before generating another candidate branch
  - in order to prove whether F entails from a KB, Phase 1 can be executed on a machine
  - every time that a branch remains open after Phase 1, the execution of Phase 2 for this branch is performed in parallel on a different machine
  - meanwhile, the main machine can carry on with the computation of Phase 1
  - if a branch remains open in Phase 2, then F is not minimally entailed from KB and the computation process can be interrupted early.

Design of `DysToPic`

Ideas

- no need for Phase 1 to wait for the result of one elaboration of Phase 2 on an open branch, before generating another candidate branch
  - in order to prove whether F entails from a KB, Phase 1 can be executed on a machine
  - every time that a branch remains open after Phase 1, the execution of Phase 2 for this branch is performed in parallel on a different machine
  - meanwhile, the main machine can carry on with the computation of Phase 1
  - if a branch remains open in Phase 2, then F is not minimally entailed from KB and the computation process can be interrupted early.

Design of `DysToPic`

Ideas

- no need for Phase 1 to wait for the result of one elaboration of Phase 2 on an open branch, before generating another candidate branch
  - in order to prove whether F entails from a KB, Phase 1 can be executed on a machine
  - every time that a branch remains open after Phase 1, the execution of Phase 2 for this branch is performed in parallel on a different machine
  - meanwhile, the main machine can carry on with the computation of Phase 1
  - if a branch remains open in Phase 2, then F is not minimally entailed from KB and the computation process can be interrupted early.

Design of `DysToPic`

Ideas

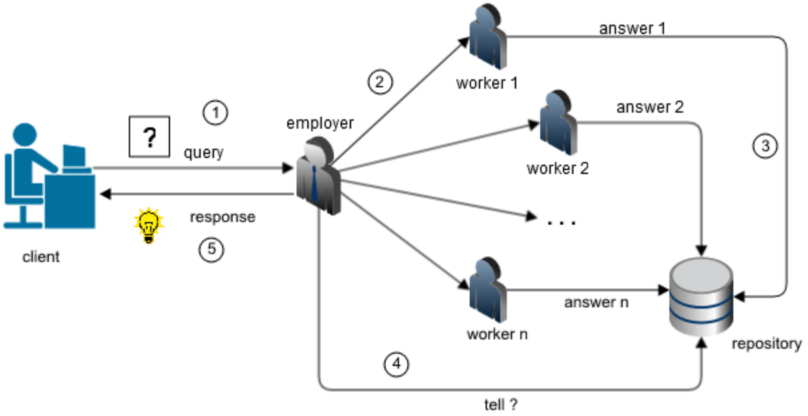
- no need for Phase 1 to wait for the result of one elaboration of Phase 2 on an open branch, before generating another candidate branch
  - in order to prove whether F entails from a KB, Phase 1 can be executed on a machine
  - every time that a branch remains open after Phase 1, the execution of Phase 2 for this branch is performed in parallel on a different machine
  - meanwhile, the main machine can carry on with the computation of Phase 1
  - if a branch remains open in Phase 2, then F is not minimally entailed from KB and the computation process can be interrupted early.

Design of `DysToPic`

Ideas

- no need for Phase 1 to wait for the result of one elaboration of Phase 2 on an open branch, before generating another candidate branch
  - in order to prove whether F entails from a KB, Phase 1 can be executed on a machine
  - every time that a branch remains open after Phase 1, the execution of Phase 2 for this branch is performed in parallel on a different machine
  - meanwhile, the main machine can carry on with the computation of Phase 1
  - if a branch remains open in Phase 2, then F is not minimally entailed from KB and the computation process can be interrupted early.

The architecture of `DysToPic`



Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - `DysToPic` is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - DysToPic is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - `DysToPic` is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - `DysToPic` is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - `DysToPic` is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - `DysToPic` is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - `DysToPic` is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

Technologies

- Tableaux rules implemented in SICStus Prolog
- Library `se.sics.jasper` to combine Java and SICStus Prolog and to decouple Phase 1 and Phase 2
- Concurrency via multithreading and RMI (Java)

Performances

- Comparison with a standard implementation PreDeLo
- Promising performances
  - `DysToPic` is better than the competitor in answering that F is not minimally entailed from KB
  - surprisingly enough, better performances also in case F is minimally entailed from KB
- advantages of distributing the computation justify the overhead of the machinery needed for that

References

- F. Baader and B. Hollunder (1995), Embedding defaults into terminological knowledge representation formalisms. *JAR*, 14(1):149–180.
- P. A. Bonatti, C. Lutz, and F. Wolter (2006). DLs with Circumscription. In *Proc. of KR*, pages 400–410.
- D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati (2007). Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *Journal of Automated Reasoning*, 39(3):385–429.
- F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf (1998). An epistemic operator for description logics. *Artificial Intelligence*, 100(1-2):225–274.
- F. M. Donini, D. Nardi, and R. Rosati (2002). Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logics (ToCL)*, 3(2):177–225.
- T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits (2004), Combining Answer Set Programming with Description Logics for the Semantic Web. In *Proc. of KR 2004*, pages 141–151.
- L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato (2013), A NonMonotonic Description Logic for Reasoning About Typicality. *Artificial Intelligence*, 195:165 – 202.
- L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato (2015), Semantic characterization of Rational Closure: from Propositional Logic to Description Logics. *Artificial Intelligence*, 226:1–33.
- U. Straccia (1993), Default inheritance reasoning in hybrid kl-one-style logics. In *Proc. of IJCAI'93*, pages 676–681.