Building the Knowledge Base System IDP^3

Marc Denecker

Knowledge Representation and Reasoning,
Department of Computer Science, KU-Leuven
Marc.Denecker@cs.kuleuven.be

Abstract IDP^3 is a Knowledge Base System for a rich logic that combines first order logic with logic programming. This article presents the motivation for building this system, discusses its logic and gives an overview of the architecture of the IDP-system, the forms of inference that it supports, and the techniques used to implement them. It also mentions applications and experiments performed with the system.

1 Motivation

Arguably, knowledge, information is an essential resource to humans. Human experts possess declarative information and use it to solve problems, accomplish tasks. How does this work?

This question boils down to what arguably can be viewed as the global research question for the field of Knowledge Representation and Reasoning (KRR): what is information, what kinds are there, and how can information be used to solve problems. Below, this question is referred to as the KRR-research question. To study information and how it is used to solve problems is obviously a scientific topic of fundamental importance. The KRR research question is about the relation between declarative background knowledge on a problem domain and computational tasks that arise in it. This places KRR at the foundations of Computer Science and Artificial Intelligence.

All fields of Computational Logic, Declarative Programming, Formal Methods and also many areas in Artificial Intelligence are concerned with aspects or instances of the KRR-research question. At present many declarative paradigms exist: first order deductive logic (FO), Deductive Databases (SQL, Datalog), Logic Programming, Abductive Logic Programming, Answer set Programming, Constraint Programming languages such as OPL, Zinc, Comet, languages for verification such as Z, B, TLA+, Alloy, temporal logics for model checking, planning languages (PDDL), Description Logics, ... In all declarative programming paradigms, declarative information is expressed; also phrasing an SQL query is an act of knowledge representation. Therefore, all of them share a concern to develop a natural knowledge representation language and solve problems with it. There are important differences between these languages, but if we peek through superficial differences in style and terminology, many overlaps in these languages are apparent.

Nevertheless, the correspondences between the languages are not always clear. Syntactical styles of languages developed in the different fields, and the concepts and terminologies used to define the languages differ widely. The issue that fragments computational logic and declarative programming more than anything else is the *reasoning task*. Declarative programming disciplines focus on one specific reasoning task, on a specific form of inference: classical logic serves to solve deductive problems, database languages serve to query databases (SQL, Datalog), logic programs are executable programs solving queries, answer set programs serve to generate answer sets, abductive logic programs solve abductive problems, inductive logic programming serves to solve inductive problems, constraint programming languages express constraint problems, etc. Expressions, theories or programs are seen as statements of a problem. Often the representation of declarative information is mixed with information about the task to be solved, and the concepts and terminologies used to specify semantics are pervaded with aspects of the task to be solved. But is declarative information not supposed to be independent of the task at hand, and hence of the specific form of inference? Here the principle of separation of concerns seems to be violated.

From a KRR point of view, the good news is that in the different declarative programming paradigms, a range of powerful solving techniques and solvers have been developed, for a range of different inference tasks. Also, a great amount of know-how on knowledge representation principles has been gathered and tested in the context of real problems. The downside is that this know-how is scattered over these different fields.

For example, consider the following proposition: "*adjacent vertices in graph G have a different color*". It can be expressed elegantly in many languages, including in FO:

$$\forall x\ y : Vertex(x) \wedge Vertex(y) \wedge G(x, y) \Rightarrow Col(x) \neq Col(y)$$

In the context of a graph coloring problem, this proposition would be the main constraint. However, this proposition has no intrinsic "constraint solving" quality; it is merely the representation of a piece of information. In other contexts, the proposition could equally well be a query or an integrity constraint for a colored graph database, or a desired correctness property to be verified for a graph coloring algorithm. Is there any principled reason why the encoding of this proposition should depend on the task? Is its FO representation less natural or modular than its encoding in CP languages, in ASP, in query or verification languages? (These are rhetoric questions.) The task-orientedness of declarative programming paradigms obscures insights in the knowledge representation principles underlying these languages.

Perhaps worse is that the task-orientedness of declarative programming paradigms is blocking an open-minded investigation of the KRR-research question itself: what problems can be solved with the specified declarative information? In declarative programming paradigms the inference task is an a priori of the paradigm. Expressions, programs or theories are written with this task in mind and are seen as specifications of such a task. The focus on the task may act as

a pair of blinders for what other tasks could be solved using the same information. Take classical logic. For most of its history, FO was seen as the language of deductive reasoning; it is still widely defined that way. A graph coloring problem is clearly not a deductive problem, and this at least partially explains why until recently, FO was not considered as an approach for this type of problem. Yet, its solution in FO is very natural. The input of a graph coloring problem is the set of data describing the graph and the colors. Given that "a database = a structure", a suitable logical representation of this input is as a structure $\mathcal{I}_o$ interpreting symbols $G/2$, $Vertex/1$ and $Colour/1$. The solution of the problem is the interpretation of function symbol Col in any structure $\mathcal{I}$ that (1) expands $\mathcal{I}_o$ and (2) satisfies the above axiom (and some other ones). Thus, this problem is a *model generation* inference problem, or more specifically, a *model expansion* problem [26]. Mutatis mutandis, similar phenomena play in all other fields of declarative problem solving. E.g., few would think of SQL as a language to express constraints in a constraint problem, or the CP-language Zinc as a query language or a language to express desirable properties to be verified in the context of the formal specification of a system.

If only we could bundle what is known about the KRR-research question in the different fields of computational logic into one "information centered" scientific framework!?

The above considerations form the motivation and the background of the FO($\cdot$)-KBS research project [13,29] that is currently conducted at the KRR-research group at the KU-Leuven. At the language level, the goal is to design a rich KR language FO($\cdot$)$^{\text{IDP}}$ that integrates language constructs from different areas of computational logic. As we explain in the next section, FO is used as a base language in this project. The notation FO($\cdot$) stands for the family of extensions of FO, and FO($\cdot$)$^{\text{IDP}}$ is the FO($\cdot$) language supported by IDP^3. At the system level, the aim is to develop a Knowledge Base System that supports various forms of inference. The system also provides a programming environment with an interface to IDP^3 to call a range of inference methods and other functionalities to solve tasks. On the implementation level, new techniques are combined with existing ones from CP, SAT, Answer Set Programming, Logic Programming, ... The system IDP^3 is the current platform of the project; it supports the language FO($\cdot$)$^{\text{IDP}}$. In the rest of the article we will explain the language and give an overview of the system.

2 The IDP^3 system

IDP^3 is developed as a Knowledge Base System (knowledge-base system (KBS)) that supports a rich declarative logic and provides various forms of inference and other logic-related functionalities for this logic.

The KB language The knowledge representation language of IDP^3 is the logic FO($\cdot$)$^{\text{IDP}}$, an extension of full FO [11]. FO was chosen as the base language

because we believe that its connectors and quantifiers are key connectors for expressing human knowledge. Other advantages of FO are the clarity of its informal semantics and the precision and simplicity of its Tarskian *possible world* model semantics. These are valuable properties of FO; however, as a KR language, FO is far too primitive. FO$(\cdot)^{\text{IDP}}$ extends First-Order Logic (FO) with types, inductive definitions [12], aggregates [28] (cardinality, sum, min and maximum), numerical quantifiers, e.g., $\exists_{\leq n}$, etc.

We illustrate the formalism with the following scenario. In a company, the hierarchy of its employees is represented by $WorksFor(Person, Person)$. The total salary cost of an employee ($SalaryCost(Person, int)$) is the sum of the salaries of the employee and everyone below him in the hierarchy. Every employee with a total salary cost of more than a million dollar is a manager.

$$\left\{\begin{array}{l} \forall p\ sc : SalaryCost(p, sc) \leftarrow \\ \qquad sc = Salary(p))+ \\ \qquad\qquad sum\,(\{x\ sal : WorksFor(x, p) \wedge SalaryCost(x, sal) : sal\}) \end{array}\right\}$$
$$\forall x\ s : SalaryCost(x, s) \wedge s \geq 1000000 \Rightarrow Manager(x)$$

This condensed theory shows the most important features of FO$(\cdot)^{\text{IDP}}$: FO, types, non-Herbrand functions ($Salary(Person) : int$), inductive definitions, aggregates, arithmetic.

The definition defines the typed relation symbol $SalaryCost$ in terms of the $Salary$ function and $WorksFor$. The definition is recursive and defines $SalaryCost$ by induction on the employee hierarchy. Its single rule specifies both the base case (for employees at the bottom of hierarchy) and the inductive case. The sum expression sums values sal over the set of all pairs (x, sal) that satisfy the enclosed formula.[1]

FO$(\cdot)^{\text{IDP}}$ is an integration of KR principles found in various declarative languages (and a few new ones too). The FO$(\cdot)^{\text{IDP}}$ definition construct is logic programming's contribution to it. Definitions in FO$(\cdot)^{\text{IDP}}$ extend logic programs with general rule bodies and open symbols (e.g., $Salary/2, WorksFor/2$). Its rule-based syntax and the well-founded semantics, extended to handle complex bodies and open symbols, make it suitable to express the most common sorts of (inductive) definitions: monotone inductive definitions and inductive definitions over a well-founded order [8,9,12]. E.g., the definition of $SalaryCost$ is a (non-monotone)definition by induction over the well-founded employee hierarchy. We believe this to be LP's most useful contribution to declarative KR and to classical logic. The way LP is integrated with FO preserves FO's assets: a clear and precise informal semantics, and a precise Tarskian possible world semantics [10].

The logic FO$(\cdot)^{\text{IDP}}$ is related to and shows overlap with the languages of SAT Modulo Theories (SMT) solvers [27], the Alloy language [20], the language of the ProB system [23] and also with Constraint Programming (CP) languages like

[1] This theory should be completed with an axiom that $WorksFor$ is a hierarchy; that is, its transitive closure relation is irreflexive. This is expressed by defining a newly introduced predicate constant $T(Person, Person)$ as the transitive closure of $WorksFor$, and expressing irreflexivity : $\neg\exists x : T(x, x)$.

OPL, Essence, Zinc [25]. Some of these languages contain higher order language constructs while FO$(\cdot)^{\text{IDP}}$ is the only one with inductive definitions. FO$(\cdot)^{\text{IDP}}$ is tightly linked to ASP as well [10] and formally extends Abductive Logic Frameworks [22].

The knowledge base system The system IDP^3 is a knowledge base system. The system manages a class of logical objects: vocabularies, structures, terms, formulas and theories. It supports a range of functionalities, from language related tasks such as normalization of formulas, to inference tasks that can be applied to an input of logical objects. The main forms of inference implemented in IDP^3 are the following.

Finite model expansion: Input: a partial structure $\mathcal{I}_o$ and FO$(\cdot)^{\text{IDP}}$ theory T; output: a model $\mathcal{I}$ of T that expands $\mathcal{I}_o$. This is the kernel of the IDP^3 system. It supports Herbrand model generation and finite model expansion which were proposed as logic-based methods for constraint solving, respectively in [16] and model expansion [26]. The architecture of this system is similar to that of recent CP-systems and ASP-systems. Like ASP-systems it is based on grounding and solving, using the solver MiniSAT(ID) [24,3]. Intelligent grounding techniques were developed to support the FO bodies and to reduce grounding size [35].

Like other solvers in CP, SMT and ASP, the solver MiniSAT(ID) has an SMT-architecture with a clause learning SAT-solver as kernel and various theory propagators for non-CNF language constructs. For example, the propagator of definitions includes a system to detect unfounded sets. Recently, the system was extended with CP technology to handle integer functions efficiently [3]. Another recent extension is lazy grounding [5].

Model checking is a special case of *model expansion* with $\mathcal{I}$ a two-valued structure interpreting the vocabulary of T; it is implemented through the model expander. The system also provides *optimization* inference. This has an additional input argument in the form of a cost term t to be minimized.

Querying structures: Input: a structure $\mathcal{I}$ and an FO sentence φ (or set comprehension $\{\overline{x} : \varphi[\overline{x}]\}$); output $\varphi^{\mathcal{I}}$ (or $\{\overline{x} : \varphi[\overline{x}]\}^{\mathcal{I}}$). The implementation in IDP^3 makes use of first order Binary Decision Diagrams as described in [35].

Propagation: Input: a theory T and partial structure $\mathcal{I}$; output: a more precise partial structure $\mathcal{I}'$ that approximates all models of T that are expansions of $\mathcal{I}$ [34]. Equivalently, propagation deduces ground literals that hold in all models of T that expand $\mathcal{I}$. The system provides a complete propagation system (co-NP complete) and a sound but incomplete polynomial algorithm based on a lifted form of unit propagation. This form of inference proved very useful for building interactive knowledge-based configuration systems [31]. It is also used internally in IDP^3 to reduce the size of the grounding. The propagation algorithm and its applications are described in [34].

Theorem proving: Input: an FO$(\cdot)^{\text{IDP}}$ theory T and FO sentence φ; output: true if $T \models \varphi$. It is implemented by translating T into a weaker FO theory

T' and calling the theorem prover Spass [32] to check whether $T' \models \varphi$. The theorem prover is used internally in the IDP system to detect functional dependencies, which are then used for optimizations as described in [4].

Model Revision: In many real-life search domains, a solution must be updated to satisfy new constraints (e.g., train rescheduling, network reconfiguration,...). The problem of updating a model of a theory to satisfy some new constraints is an incremental variant of the model generation problem [33].

Δ-model expansion: Input: a definition Δ and a structure $\mathcal{I}_o$ interpreting all parameter symbols of Δ; output the unique model $\mathcal{I}$ of Δ that expands $\mathcal{I}_o$. This problem extends the view materialization problem in Datalog. It is an instance of model expansion but is solved in IDP^3 using different technology. The close relationship between definitions and logic programs under the well-founded semantics was exploited to translate Δ and $\mathcal{I}$ into a tabled Prolog program and use XSB to compute $\mathcal{I}$ or to solve queries with respect to Δ and $\mathcal{I}$ in a goal-oriented way [21].

Several other tools and components have been build in and around IDP^3. It comprises a system for symmetry detection and breaking [14]. This system is standalone and can be used with any SAT system. Its effectiveness is demonstrated in the 2013 SAT-competition where a SAT solver combined with it won the track "Hard combinatorial" [30].

An IDE was developed for IDP^3. It is available at `dtai.cs.kuleuven.be/krr/software/idp-ide`.

We developed a useful tool ID^P_{Draw} for visualizing structures [18]. This system is an application of Δ-model expansion. To visualize a structure $\mathcal{I}$, the user writes a definition Δ defining an ontology of graphical predicates in terms of the symbols interpreted in $\mathcal{I}$. The system performs Δ-model expansion to compute the graphical predicates, which are then passed on to a graphical system that visualizes the graphical data.

The system supports translators from different predicate and propositional languages. One is a translator from $\text{FO}(\cdot)^{\text{IDP}}$ to FO(arithm) (FO extended with arithmetic). It is available externally and is used internally as a preprocessing step to call the theorem prover. Other translators are for propositional languages: translators of IDP^3's ground format ECNF from and to standard CNF, lparse and flatzinc, and translators from QBF and OPB to ECNF.

IDP^3 provides a programming environment with an interface from which the different functionalities of IDP^3 can be called [29,7] and in which the different sorts of logic objects can be created and manipulated. This interface was build in the imperative programming language Lua [19].

The KBS-paradigm offers the opportunity to apply different sorts of inference to the same theory. The idea is that quite often, a software product solves several problems in the same problem domain. The question then is whether the same declarative representation of the background knowledge can be used to solve these different tasks, by applying the suitable form of inference. We only have begun to explore this. An experiment where this worked very well is presented in [31].

IDP was used there to implement an interactive configuration system. Several forms of inference on the same theory, of which propagation inference was the most important, were helpful to guide a user through the configuration problem. Other applications of IDP^3 were conducted for solving machine-learning problems [2], for university course scheduling [6] and for modelling and verification of software architectures [17].

IDP^3 is succesfully used as a didactic tool in introductory and advanced courses on predicate logic and knowledge representation in several universities.

At present the system is among the best ASP systems, and its solver performs very well in the class of minizinc systems [1]. Both the KBS IDP^3 and the search algorithm MiniSAT(ID) are open-source. They are available from `dtai.cs.kuleuven.be/krr/software/`. The webpage contains a tutorial and a list of examples.

References

1. Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. Features for building csp portfolio solvers. arXiv:1308.0227 [cs.AI], 2013.
2. Hendrik Blockeel, Bart Bogaerts, Maurice Bruynooghe, Broes De Cat, Stef De Pooter, Marc Denecker, Anthony Labarre, Jan Ramon, and Sicco Verwer. Modeling machine learning and data mining problems with FO($\cdot$). In Dovier and Santos Costa [15], pages 14–25.
3. Broes De Cat, Bart Bogaerts, Jo Devriendt, and Marc Denecker. Model expansion in the presence of function symbols using constraint programming. Accepted for the IEEE International Conference on Tools with Artificial Intelligence (ICTAI) - 2013.
4. Broes De Cat and Maurice Bruynooghe. Detection and exploitation of functional dependencies for model generation. Accepted for the 29th International Conference On Logic Programming.
5. Broes De Cat, Marc Denecker, and Peter Stuckey. Lazy model expansion by incremental grounding. In Dovier and Costa [15], pages 201–211.
6. Broes De Cat, Christophe Machiels, Gerda Janssens, and Marc Denecker. Regularity requirements in university course timetabling. In *Proceedings of the 11th Workshop on Preferences and Soft Constraints (SofT 2011),*, pages 31–45, September 2011.
7. Stef De Pooter, Johan Wittocx, and Marc Denecker. A prototype of a knowledge-based programming environment. In *International Conference on Applications of Declarative Programming and Knowledge Management*, 2011.
8. Marc Denecker. The well-founded semantics is the principle of inductive definition. In Jürgen Dix, Luis Fariñas del Cerro, and Ulrich Furbach, editors, *JELIA*, volume 1489 of *LNCS*, pages 1–16. Springer, 1998.
9. Marc Denecker, Maurice Bruynooghe, and Victor W. Marek. Logic programming revisited: Logic programs as inductive definitions. *ACM Transactions on Computational Logic (TOCL)*, 2(4):623–654, 2001.
10. Marc Denecker, Yulia Lierler, Miroslaw Truszczynski, and Joost Vennekens. A tarskian informal semantics for asp. In *International Conference on Logic Programming (Technical Communications)*, 2012.

11. Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. In *ACM Transactions on Computational Logic (TOCL)* [12], pages 14:1–14:52.
12. Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL)*, 9(2):14:1–14:52, April 2008.
13. Marc Denecker and Joost Vennekens. Building a knowledge base system for an integration of logic programming and classical logic. In María García de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *LNCS*, pages 71–76. Springer, 2008.
14. Jo Devriendt, Bart Bogaerts, Christopher Mears, Broes De Cat, and Marc Denecker. Symmetry propagation: Improved dynamic symmetry breaking in SAT. In *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence, ICTAI'12*, 2012.
15. Agostino Dovier and Vítor Santos Costa, editors. *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
16. Deborah East and Mirosław Truszczyński. Predicate-calculus-based logics for modeling and solving search problems. *ACM Transactions on Computational Logic (TOCL)*, 7(1):38–83, 2006.
17. Thomas Heyman. *A Formal Analysis Technique for Secure Software Architectures (Een formele analysetechniek voor veilige softwarearchitecturen)*. PhD thesis, Arenberg Doctoral School, KU Leuven, March 2013.
18. IDPDraw: finite structure visualization. `http://dtai.cs.kuleuven.be/krr/software/visualisation`.
19. Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. Lua – an extensible extension language. *Software: Practice and Experience*, 26(6):635–652, 1996.
20. Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM'02)*, 11(2):256–290, 2002.
21. Joachim Jansen, Albert Jorissen, and Gerda Janssens. Compiling input$*$ FO($\cdot$) inductive definitions into tabled prolog rules for IDP^3. Accepted for publication in Theory and Practice of Logic Programming, Special Issue ICLP-2013.
22. Antonis C. Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *J. Log. Comput.*, 2(6):719–770, 1992.
23. Michael Leuschel and Michael J. Butler. ProB: An automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
24. Maarten Mariën, Johan Wittocx, Marc Denecker, and Maurice Bruynooghe. SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In Hans Kleine Büning and Xishun Zhao, editors, *SAT*, volume 4996 of *LNCS*, pages 211–224. Springer, 2008.
25. Kim Marriott, Nicholas Nethercote, Reza Rafeh, Peter J. Stuckey, Maria Garcia de la Banda, and Mark Wallace. The design of the Zinc modelling language. *Constraints*, 13(3):229–267, 2008.
26. David G. Mitchell and Eugenia Ternovska. A framework for representing and solving NP search problems. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 430–435. AAAI Press / The MIT Press, 2005.
27. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

28. Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming (TPLP)*, 7(3):301–353, 2007.
29. Stef De Pooter, Johan Wittocx, and Marc Denecker. A prototype of a knowledge-based programming environment. *CoRR*, abs/1108.5667, 2011.
30. The international SAT competition 2013. `http://satcompetition.org/2013/results.shtml`, 2013.
31. Hanne Vlaeminck, Joost Vennekens, and Marc Denecker. A logical framework for configuration software. In António Porto and Francisco Javier López-Fraguas, editors, *PPDP*, pages 141–148. ACM, 2009.
32. Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. Spass version 3.5. In Renate A. Schmidt, editor, *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.
33. Johan Wittocx, Broes De Cat, and Marc Denecker. Towards computing revised models for FO theories. In Salvador Abreu and Dietmar Seipel, editors, *INAP*, pages 199–212, 2009.
34. Johan Wittocx, Marc Denecker, and Maurice Bruynooghe. Constraint propagation for first-order logic and inductive definitions. *ACM Trans. Comput. Logic*, 14(3):17:1–17:45, August 2013.
35. Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *Journal of Artificial Intelligence Research*, 38:223–269, 2010.