The effects of buying a new car: an extension of the IDP Knowledge Base System

Pieter Van Hertum and Joost Vennekens and Bart Bogaerts
Jo Devriendt and Marc Denecker
Department of Computer Science, KU Leuven

submitted 10 April 2013; accepted 23 May 2013

Abstract

A long term goal in knowledge representation is the development of a knowledge base system (KBS). A KBS stores knowledge and is equipped with various domain independent inference methods to perform different reasoning tasks for a broad field of applications. In this paper we took a state-of-the-art KBS, IDP, which supports a rich extension of first-order logic and tested its applicability on a prototypical example from the business rule domain: a car rental system. This paper adresses two questions. On the language level, we investigate whether the relevant domain knowledge of this problem domain can be expressed in IDP's language; on the operational level, we study the forms of inferences needed to implement various use cases that arise in this domain. The results are mixed. One obstacle we encountered is that the IDP language does not have a good formalism to model the knowledge about situations where a new object is created. To remedy this, we propose a language extension to FO(·).

1 The Car Rental application

The goal of the IDP KBS ((Denecker and Vennekens 2008), (De Pooter et al. 2011)) is to implement complex behaviours in practical applications by means of declarative specifications. In industry *business rule systems* (Halle 2001) are often used for this purpose. In such systems, applications are encoded in the form of sets of "If-Then"-rules. The rules derive their meaning from the way they are interpreted in the match-resolve-act cycle, in which the Then-part of rules may be derived or executed given the satisfaction of the If-part at some stage in the process.

The language used in IDP is a rich extension of FO, denoted $\text{FO}(\cdot)^{\text{IDP}}$. An $\text{FO}(\cdot)^{\text{IDP}}$ theory has only a declarative meaning, i.e., it cannot be executed, but can be reasoned upon by various inference mechanisms. An advantage of this approach is that we can re-use this knowledge for implementing different functionalities.

To compare the current state of the IDP KBS with that of business rule systems, we will focus on a standard benchmark for business rule systems: the EU-Rent application (Group 2000).

EU-Rent is a fictitious car rental company. Cars can be rented at and returned to several branches. Handling reservations, purchases, moves of cars,... are tasks that a rental applications should be able to handle. This benchmark is proposed in the literature to evaluate business rule systems. This domain provides a simple but varied class of use cases to evaluate the expressivity and functionality of such systems. In this paper, we examine two specific use cases that we will try to specify in FO(·), so that the IDP system can run the appropriate inferences.

2 Implementing EU-Rent in IDP

2.1 General Description: Valid States

A car rental system maintains a state recording available cars, reservations, planning of car transports, information about clients, etc. We represent such a state using a vocabulary based on the domain specification (in (Van Hertum et al. 2013)), which will be used to represent a single state of the application. This vocabulary consists of types (*Car*, *Client*, *Reservation* ...), functions (like $Driver : Reservation \rightarrow Client$, which maps reservations to the client who made them) and predicates (like for example $Overlapping(Reservation, Reservation)$ and $Accepted(Reservation)$). A structure for this vocabulary records the actual state of the world (which cars are where, who has requested which reservations, etc.), as well as the current plans of the system (which reservations will be fulfilled by which cars, when should cars be moved between branches, etc.).

In (Van Hertum et al. 2013) an FO$(\cdot)^{\text{IDP}}$ theory describing valid states is formulated, containing constraints such as:

$$\forall r1\ r2 : Overlapping(r1, r2) \wedge r1 \neq r2 \wedge Accepted(r1) \wedge Accepted(r2) \\ \Rightarrow Driver(r1) \neq Driver(r2).$$

2.2 Building a dynamic system

The EU-rent problem domain is a dynamic domain in which new reservation requests are made, existing ones are cancelled, new cars are bought, cars are sold or stolen, reservation dates are changed, etc. We will refer to such dynamic changes as *transactions*. Modelling transactions in FO$(\cdot)$ can be done by a linear time theory representing how the state of the system evolves. To express a linear time theory, we need to extend our vocabulary, which so far only took static states into account. For this, we introduce a new type *State*, and add a *State* argument to all predicates and functions that may change state. For example, $Accepted(r, s)$ means that reservation r is accepted at state s. Each structure for this extended vocabulary now corresponds to an entire sequence of states.

At any stage, users may apply a transaction on a state. E.g. a reservation request may be issued. What form of inference is required to derive the effects of such transactions? Essentially it is a form of *progression inference* (Lin and Reiter 1997): computing a possible next state from a previous state given a temporal theory.

2.3 Two approaches for scheduling reservations

Having the necessary vocabulary, theory and inferences ready, we can approach our first use case, where the goal is to handle new reservations for renting a car. Depending on the available cars, such a request can be denied or accepted, in which case the planning must be modified. We solve this use case in two ways.

A first approach is by mimicking a business rule solution, where If-Then rules are used to implement an algorithm that will decide whether cars are available, and update the current reservation planning accordingly. Business rule systems are deterministic in nature: given a set of facts, only one set of new information –i.e. one model– is derived. Similar behaviour can be modelled in FO$(\cdot)^{\text{IDP}}$ using inductive definitions (Denecker 2000), which are also deterministic. Since definitions in FO$(\cdot)$ are well-founded (Van Gelder et al. 1991) and well-founded definitions only have one model, we can emulate this behaviour and model a subset of the car rental domain using one formal definition. An example of such a definition:

$$\left\{ \begin{array}{l} \forall c\ t : Allocated(ReqRes(t), c, t+1) \leftarrow Available(c, Period(t), t+1) \wedge \\ \qquad\qquad (\forall c' : c' < c \Rightarrow \neg Available(c', ReqRes(t), t+1)) \end{array} \right\}$$

This definition states that a car c is allocated to the requested reservation if the car is the smallest car available for the reservation.

This business rule-like definition can be refined to implement improved reservation algorithms that, for example, would also re-schedule assignments to previous car reservations to be able to satisfy more requests or to be more profitable for the company by reducing expenses or increasing income. However, such a methodology soon becomes very tedious and error-prone. Luckily, the power of logic can be used to simplify this task. We simply propose a new theory, based on the theory that statically states whether a given state is valid, that states that *every* state must be valid.

Advantages of this approach over the mimicking business rules approach are simplicity and guaranteed correctness. An operational difference with the business rules approach is that the result is no longer deterministic: at each state, the system can freely choose a new schedule, which might differ drastically from the schedule in the previous state. This may pose a practical problem for the company in the sense that the planning of manpower and resources becomes difficult (e.g. will tomorrow's planning still require to move a certain car from one branch to another?). A solution for this problem would be to apply model revision or to compute solutions minimising an objective function that penalises modifications to the previous schedule. We propose an extension of this approach in (Van Hertum et al. 2013), which is a model revision inference with a mapping from predicates and functions in the vocabulary to tuples of integers, representing a cost measure to changing a predicate or function's interpretation. For example, accepting more reservations, i.e. expanding *Accepted*'s interpretation, is profitable, while trimming *Accepted*'s interpretation by cancelling previously accepted reservations, can be very costly.

2.4 Adding New Objects

Our second use case is an extension of the first one: we want to extend the car rental application with the possibility of adding new cars. In many software systems, adding a new object is a trivial task: new classes or new entries in a database are being created without a second thought. The car rental problem domain also allows for cars to be purchased, and hence it must be possible to add a new car to the set of cars. In logic speak: it must be possible to add a new object to the *Car* domain.

The obvious way to do this would simply be to extend the structure (or database) with an extra car. However, this means that the knowledge about the introduction of a new car is not present in the logical theory representing the problem domain. As a result, the information about this transaction is not reusable for other forms of reasoning, e.g. making simulations of the system, verification, etc. To remedy this, we propose a new logical operator representing the knowledge that a new domain element is created.

The exact syntax and semantics of this operator are explained in (Van Hertum et al. 2013). Here we only take a look at how a novel logic operator could allow the creation of a car in our car rental problem domain:

$$\left\{ \begin{array}{l} \forall s\ sn : \textbf{new}\ c : SerialNr(c) = sn \\ \qquad \wedge PurchaseDate(c) = s \wedge Mileage(c) = 0 \leftarrow BuyCar(sn, s). \end{array} \right\}$$

The above definition states that if a car with serial number *sn* is bought in a certain state *s*, a new car has to be added to the system with that specific serial number, purchased at that point in time, with zero mileage, and some other initial attributes. In order to support this kind of expressions, we introduce a new logic, FO(ID^+), which extends FO(ID) by allowing more complex formulas in the head of a definition.

3 Conclusion

In this paper, we put the IDP KBS to the test in a car rental application from the business rules domain. We argued that an implementation in a KBS is more useful than the traditional

approach taken by business rule systems since modelling a problem domain in a KBS allows for reuse of knowledge and specifications in various inferences. With this in mind, we mimicked the rule-based approach in the IDP system and uncovered another drawback of this approach: the determinism that is implicit in a rule-based approach does not always allow the system to optimise profit. Rather, it encodes a heuristic to deterministically pick a reasonably good solution, and excludes other, possibly better solutions. For this reason we modelled a declarative description of the EU-Rent application, and explained how to solve different use cases in this problem domain. Doing this, we noticed that the mundane task of creating new objects tasks did not fit well in the KBS approach. This deficiency is not a fundamental fault of the KBS paradigm, but rather a lack of expressivity of the currently used version of FO(·). Therefore, we described an extension of FO(·) which allowed the modelling of creating a new object.

References

De Pooter, S., Wittocx, J., and Denecker, M. 2011. A prototype of a knowledge-based programming environment. In *International Conference on Applications of Declarative Programming and Knowledge Management*.

Denecker, M. 2000. Extending classical logic with inductive definitions. In *CL*, J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, Eds. LNCS, vol. 1861. Springer, 703–717.

Denecker, M. and Vennekens, J. 2008. Building a knowledge base system for an integration of logic programming and classical logic. In *ICLP*, M. García de la Banda and E. Pontelli, Eds. LNCS, vol. 5366. Springer, 71–76.

Group, B. R. 2000. Defining Business Rules ˜ What Are They Really? Tech. rep.

Halle, B. V. 2001. *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. John Wiley & Sons, Inc., New York, NY, USA.

Lin, F. and Reiter, R. 1997. How to progress a database. *Artif. Intell. 92,* 1-2, 131–167.

Van Gelder, A., Ross, K. A., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

Van Hertum, P., Vennekens, J., Bogaerts, B., Devriendt, J., and Denecker, M. 2013. The effects of buying a new car: an extension of the idp knowledge based system. *CW Reports CW640*.