Semantic DMN: Formalizing Decision Models with Domain Knowledge

Diego Calvanese[1(✉)], Marlon Dumas[2], Fabrizio M. Maggi[2], and Marco Montali[1]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
{calvanese,montali}@inf.unibz.it
[2] University of Tartu, Tartu, Estonia
{marlon.dumas,f.m.maggi}@uu.ee

Abstract. The Decision Model and Notation (DMN) is a recent OMG standard for the elicitation and representation of decision models. DMN builds on the notion of decision table, which consists of columns representing the inputs and outputs of a decision, and rows denoting rules. DMN models work under the assumption of complete information, and do not support integration with background domain knowledge. In this paper, we overcome these issues, by proposing *decision knowledge bases* (DKBs), where decisions are modeled in DMN, and domain knowledge is captured by means of first-order logic with datatypes. We provide a logic-based semantics for such an integration, and formalize how the different DMN reasoning tasks introduced in the literature can be lifted to DKBs. We then consider the case where background knowledge is expressed as an $\mathcal{ALC}$ description logic ontology equipped with datatypes, and show that in this setting, all reasoning tasks can be actually decided in ExpTime. We discuss the effectiveness of our framework on a case study in maritime security.

1 Introduction

The Decision Model and Notation (DMN) [11] is a recent OMG standard for the elicitation and representation of decision models, and for managing their interconnection with business processes, separating decision and control-flow logic [4]. The standard is already receiving widespread adoption in the industry, and an increasing number of tools and techniques are being developed to assist users in modeling, checking, and applying DMN models. DMN builds on the notion of a *decision table* (cf. [13]), which consists of columns representing the inputs and outputs of a decision, and rows denoting rules. Each rule is a conjunction of basic expressions, which in our case are captured in a language known as S-FEEL, which is also part of the DMN standard itself.

According to the standard, DMN models work under the assumption of complete information, and do not support integration with background domain knowledge. In this paper, we overcome this limitation, by proposing a combined framework, which we call *Semantic DMN*, that is based on *decision*

S. Costantini et al. (Eds.): RuleML+RR 2017, LNCS 10364, pp. 70–86, 2017.
DOI: 10.1007/978-3-319-61252-2_6

Table 1. Ontology of cargo ships and their features.

Ship type	Short name	Length (m)	Draft (m)	Capacity (TEU)
Converted Cargo Vessel	*CCV*	135	0–9	500
Converted Tanker	*CT*	200	0–9	800
Cellular Containership	*CC*	215	10	1000–2500
Small Panamax Class	*SPC*	250	11–12	3000
Large Panamax Class	*LPC*	290	11–12	4000
Post Panamax	*PP*	275–305	11–13	4000–5000
Post Panamax Plus	*PPP*	335	13–14	5000–8000
New Panamax	*NP*	397	15.5	11000–14500

knowledge bases (DKBs). In a DKB, decisions are modeled in DMN, and background domain knowledge[1] is captured by means of an ontology expressed in multi-sorted first-order logic. The different sorts are used to seamlessly integrate abstract domain objects with the data values belonging to the concrete domains used in the DMN rules (such as strings, integers, and reals).

For the enriched setting of Semantic DMN, we provide a logic-based semantics, and we formalize how the different DMN reasoning tasks that have been introduced in the literature can be lifted to DKBs. We then approach the problem of actually reasoning on DKBs, and of devising effective algorithms for the different reasoning tasks captured by our formalization. For this purpose, we need to put restrictions on how to express background knowledge, and we consider the significant case where such knowledge is formulated in terms of an ontology expressed in a description logic (DL) [3] equipped with *datatypes* [2,9,10,14]. In such a DL, besides the domain of abstract objects, one can refer to concrete domains of data values (such as strings, integers, and reals) accessed through functional relations, and one can express conditions on such values by making use of *unary* predicates[2] over the concrete domains. Specifically, we prove that for the case where the DL ontology is epressed in $\mathcal{ALC}(\mathfrak{D})$, i.e., $\mathcal{ALC}$ [3] extended with multiple datatypes, all reasoning tasks can be actually decided in ExpTime.

We show the effectiveness of our framework by considering a case study in maritime security, arguing that our approach facilitates modularity and separation of concerns.

[1] We remark that our notion of *domain knowledge* is different from that of *business knowledge model* in DMN. The latter is a reusable decision logic, with a purely operational meaning.

[2] The restriction to unary predicates only, is what distinguishes DLs with datatypes from the richer setting of DLs with concrete domains, where in general arbitrary predicates over the datatype/concrete domain can be specified.

2 Case Study

Our case study is inspired by the international Ship and Port Facility Security Code[3], used by port authories to determine whether a ship can enter a Dutch port.

2.1 Domain Description

As shown in Table 1, there are several types of cargo ships that may enter a port, with different characteristics: *(i) Length* of the ship (in m); *(ii) Draft* size (in m); *(iii) Capacity* of the ship (in *TEU*, for Twenty-foot Equivalent Units). Such characteristics, together with other data about the ships, allow the port managers to decide whether to grant entrance permission to an incoming ship or not. More specifically, a ship can enter the port only if it complies with the requirements of the inspection, which is the case if it is equipped with a *valid certificate of registry*, and it meets the *safety requirements*.

The ship's certificate is valid if its expiration date is after the current date. The rules for establishing whether a ship meets the safety requirements depend on its characteristics, and the amount of its residual cargo. In particular, small ships (with length $\leq$260 m and draft $\leq$10 m) may enter only if their capacity is $\leq$1000 TEU. Ships with a small length ($\leq$260 m), medium draft $>$10 and $\leq$12 m, and capacity $\leq$4000 TEU, may enter only if cargo residuals have $\leq$0.75 mg dry weight per cm^2. Medium-sized ships (with length $>$260 m and $<$320 m, and draft $>$10 m and $\leq$13 m), and with a cargo capacity $<$6000 TEU, may enter only if their residuals have $\leq$0.5 mg dry weight per cm^2. Big ships with length between 320 m and 400 m, draft $>$13 m, and capacity $>$4000 TEU, may enter only if their carried residuals have $\leq$0.25 mg dry weight per cm^2.

2.2 Challenges

The first challenge posed by this case study concerns modeling, representation, management, and actual application of the decision rules that relate the numerical inputs capturing the characteristics of ships, to the boolean, clearance output. All these issues are tackled by the DMN standard. In particular, the standard defines clear guidelines to encode and graphically represent the input/output attributes and the rules of interest in the form of a DMN decision table. This table, in turn, may be used to document the decision logic for clearance determination, and to match the data of a ship with the modeled rules, computing the corresponding output(s), i.e., whether the ship can enter or not. this latter mechanism is backed up by a formal semantics in predicate logic [5].

In addition, DMN allows the modeler to decorate the decision with meta-information: *completeness* indicates that rules cover all possible input configurations, while the *hit policy* describes how input may match with the rules. Different hit policies are used to declare whether rules are non-overlapping, or may instead simultaneously match with the same input, then also specifying how to calculate the final output.

[3] https://dmcommunity.wordpress.com/challenge/challenge-march-2016/.

A crucial aspect is that such meta-information is declared in DMN without actually checking whether it suitably captures how the decision logic behaves. Dedicated algorithms have been thus developed to accomplish this task (see, e.g., [5,12,15–17]). There are, however, a number of additional crucial challenges that cannot be tackled by capturing the decision logic alone. Let us imagine how a decision table for ship clearance could actually be employed in an actual Dutch port, when a ship is approaching the port. How would the port authority know about all ship characteristics needed to take the decision? An immediate, but quite inconvenient, solution would be to measure all required characteristics on a per ship basis, then applying the decision table directly so as to compute the clearance outcome. A more pragmatic and feasible approach is to exploit the *domain knowledge* captured in Table 1, by acquiring from the ship only the information regarding *ship type* and *cargo residuals*, while using Table 1 to infer from the ship type the information about length, draft, and capacity. It is important to stress that the possibility of interconnecting multiple DMN tables (so that the output of one table is used as input of another table), also supported by the standard, is not applicable here: Table 1 is not a decision table, since it is not always possible to univocally compute the ship characteristics from the type (see, e.g., the case of *Post Panamax* ship type). In fact, the domain knowledge captured by Table 1 is a set of *constraints*, implicitly discriminating between allowed combinations of ship types and characteristics, from those that are impossible. In this light, Table 1 captures a *domain ontology*.

The interplay between such a domain ontology and the ship clearance decision model is far from trivial. On the one hand, it requires to lift from an approach working under complete information to one that works under *incomplete information*, and where the background knowledge is used to complement the known inputs, before the corresponding outputs are inferred. On the other hand, it does not only impact how decision table outputs are computed, but it also changes the interpretation of the completeness and hit policy indicators: they cannot be checked anymore by analyzing the decision table in isolation (as in [5]), but *in the context of the domain knowledge*.

In particular, by elaborating on the rules above, one would understand that rules are non-overlapping regardless of the domain knowledge, since their input conditions are mutually exclusive. However, one would also conclude, by mistake, that they are not complete, since, e.g., they do not cover the case of a long ship ($\geq$320 m) with small draft ($\leq$10 m). However, under the assumption that all possible ship types are those listed in Table 1, one would know that such a combination of parameters is impossible and, more in general, that the set of rules is indeed complete w.r.t. the domain knowledge.

3 Sources of Decision Knowledge

We now generalize the discussion in Sect. 2 by introducing the two main sources of decision knowledge: background knowledge expressed using a logical theory enriched with datatypes, and decision logic captured in DMN.

3.1 Logics with Datatypes

To capture background knowledge, we resort to a variant of multi-sorted first-order logic (see, e.g., [6]), which we call $\mathsf{FOL}(\mathfrak{D})$, where one sort Δ denotes a domain of abstract objects, while the remaining sorts represent a finite collection $\mathfrak{D}$ of datatypes. We consider a countably infinite set Σ of predicates, where each $p \in \Sigma$ comes with an arity n, and a signature $\mathsf{Sig}_p : \{1, \ldots, n\} \to \mathfrak{D} \uplus \{\Delta\}$, mapping each position of p to one of the sorts. $\mathsf{FOL}(\mathfrak{D})$ contains unary and binary predicates only. A unary predicate N with $\mathsf{Sig}_N(1) = \Delta$ is called a *concept*, a binary predicates P with $\mathsf{Sig}_P(1) = \mathsf{Sig}_P(2) = \Delta$ a *role*, and a binary predicate F with $\mathsf{Sig}_F(1) = \Delta$ and $\mathsf{Sig}_F(2) \in \mathfrak{D}$ a *feature*.

Example 1. The cargo ship ontology in Table 1 should be interpreted as follows: each entry applies to a ship, and expresses how the specific ship type constrains the other features of the ship, namely length, draft, and capacity. Thus the first table entry is encoded in $\mathsf{FOL}(\mathfrak{D})$ as

$$\forall s.\mathit{CCV}(s) \rightarrow \mathit{Ship}(s) \wedge \forall l.(\mathit{length}(s,l) \rightarrow l = 135) \wedge \\ \forall d.(\mathit{draft}(s,d) \rightarrow d \geq 0 \wedge d \leq 9) \wedge \forall c.(\mathit{capacity}(s,c) \rightarrow c = 500),$$

where *CCV* and *Ship* are concepts, while *length*, *draft*, and *capacity* are features whose second component is of sort *real*. ■

We consider also well-behaved fragments of $\mathsf{FOL}(\mathfrak{D})$ that are captured by description logics (DLs) extended with datatypes. For details on DLs, we refer to [3], and for a survey of DLs equipped with datatypes (also called, in fact, *concrete domain*), to [9]. Here we adopt the DL $\mathcal{ALC}(\mathfrak{D})$, a slight extension of the DL $\mathcal{ALC}(\mathcal{D})$ [9] with multiple datatypes. As for datatypes, we follow [1], which is based on the OWL 2 datatype map [10, Sect. 4], but we adopt some simplifications that suffice for our purposes.

A *(primitive) datatype* $\mathcal{D}$ is a pair $\langle \Delta_\mathcal{D}, \Gamma_\mathcal{D} \rangle$, where $\Delta_\mathcal{D}$ is the *domain* of values[4] of $\mathcal{D}$, and $\Gamma_\mathcal{D}$ is a (possibly infinite) set of *facets*, denoting unary predicate symbols. Each facet $S \in \Gamma_\mathcal{D}$ comes with a set $S^\mathcal{D} \subseteq \Delta_\mathcal{D}$ that rigidly defines the semantics of S as a subset of $\Delta_\mathcal{D}$. Given a primitive datatype $\mathcal{D}$, *datatypes* $\mathcal{E}$ *derived from* $\mathcal{D}$ are defined according to the following syntax

$$\mathcal{E} \longrightarrow \mathcal{D} \mid \mathcal{E}_1 \cup \mathcal{E}_2 \mid \mathcal{E}_1 \cap \mathcal{E}_2 \mid \mathcal{E}_1 \setminus \mathcal{E}_2 \mid \{d_1, \ldots, d_m\} \mid \mathcal{D}[S]$$

where S is a facet for $\mathcal{D}$, and $d_1, \ldots, d_m$ are datatype values in $\Delta_\mathcal{D}$. The domain of a derived datatype is obtained for $\cup$, $\cap$, and $\setminus$, by applying the corresponding set operator to the domains of the component datatypes, for $\{d_1, \ldots, d_m\}$ as the set $\{d_1, \ldots, d_m\}$, and for $\mathcal{D}[S]$ as $S^\mathcal{D}$. In the remainder of the paper, we consider the (primitive) datatypes present in the S-FEEL language of the DMN standard: strings equipped with equality, and numerical datatypes, i.e., naturals, integers,

[4] We blur the distinction between *value space* and *lexical space* of OWL 2 datatypes, and consider the datatype domain elements as elements of the lexical space interpreted as themselves.

rationals, and reals equipped with their usual comparison operators (which, for simplicity, we all illustrate using the same set of standard symbols $=$, $<$, $\leq$, $>$, $\geq$). We denote this core set of datatypes as $\mathfrak{D}$. Other S-FEEL datatypes, such as that of datetime, are syntactic sugar on top of $\mathfrak{D}$.

A facet for one of these datatypes $\mathcal{D}$ is specified using a binary comparison predicate $\approx$, together with a *constraining value* v, and is denoted as $\approx_v$. E.g., using the facet $\leq_9$ of the primitive datatype *real*, we can define the derived datatype $real[\leq_9]$, whose value domain are the real numbers that are ≤ 9. In the following, we abbreviate $\mathcal{D}[S_1] \cap \mathcal{D}[S_2]$ as $\mathcal{D}[S_1 \wedge S_2]$, $\mathcal{D}[S_1] \cup \mathcal{D}[S_2]$ as $\mathcal{D}[S_1 \vee S_2]$, and $\mathcal{D}[S_1] \setminus \mathcal{D}[S_2]$ as $\mathcal{D}[S_1 \wedge \neg S_2]$, where S_1 and S_2 are either facets or their combinations with boolean/set operators.

Let Δ be a countably infinite universe of objects. A (DL) *knowledge base with datatypes* (KB hereafter) is a tuple $\langle \Sigma, T, A \rangle$, where Σ is the *KB signature*, T is the *TBox* (capturing the intensional knowledge of the domain of interest), and A is the *ABox* (capturing extensional knowledge). When the focus is on the intensional knowledge only, we omit the ABox, and call the pair $\langle \Sigma, T \rangle$ *intensional KB* (IKB). The form of T and A depends on the specific DL of interest. We review each component next.

Signature. $\Sigma = \Sigma_C \uplus \Sigma_R \uplus \Sigma_F$ consists of: *(i)* a finite set Σ_C of *concept names*, i.e., unary predicates interpreted over Δ, *(ii)* a finite set Σ_R of *role names*, binary predicates connecting pairs of objects in Δ; and *(iii)* a finite set Σ_F of *features*, i.e., binary predicates connecting objects to corresponding typed values. In particular, each feature F comes with its datatype $\mathcal{D}_F \in \mathfrak{D}$.

TBox. T is a finite set of universal FO axioms based on predicates in Σ, and on predicates and values of datatypes in $\mathfrak{D}$. To capture such axioms, we employ the usual DL syntax, using the boolean connectives $\sqcap$, $\sqcup$ and $\neg$ for intersection, union and complement, and $\exists R.\, C$ for qualified existential restriction. In the case of $\mathcal{ALC}(\mathfrak{D})$, such axioms are built from $\mathcal{ALC}(\mathfrak{D})$ *concepts*, inductively defined as follows:

- An atomic concept $N \in \Sigma_C$ is a concept;
- $\top$ and $\bot$ are concepts, respectively denoting the top and empty concepts;
- given a concept C, its complement $\neg C$ is a concept;
- given two concepts C and D, their conjunction $C \sqcap D$ is a concept;
- given a role $R \in \Sigma_R$ and a concept C, the qualified existential restriction $\exists R.\, C$ is a concept;
- given a feature $F \in \Sigma_F$, and a datatype r that is either $\mathcal{D}_F$ or a datatype derived from $\mathcal{D}_F$, the feature restriction $\exists F.\, r$ is a concept.

Intuitively, $\exists F.\, r$ allows the modeler to single out those objects having an F-feature that satisfies condition r, interpreted in accordance with the underlying datatype. We adopt the usual abbreviations $C \sqcup D$ for $\neg(\neg C \sqcap \neg D)$, and $\forall R.\, C$ for $\neg \exists R.\, \neg C$.

An $\mathcal{ALC}(\mathfrak{D})$ TBox is a finite set of *inclusion assertions* of the form $C \sqsubseteq D$, where C and D are $\mathcal{ALC}(\mathfrak{D})$ concepts. Intuitively, such assertions model that whenever an individual is an instance of C, then it is also an instance of D.

Example 2. The $\mathcal{ALC}(\mathfrak{D})$ encoding of the first entry in Table 1 is:

$$\mathit{CCV} \sqsubseteq \mathit{Ship} \sqcap \forall \mathit{length}.\, \mathit{real}[=_{135}] \sqcap \forall \mathit{draft}.\, \mathit{real}[\geq_0 \wedge \leq_9] \sqcap \forall \mathit{capacity}.\, \mathit{real}[=_{500}]$$

All other table entries can be formalized in a similar way. The entire table is then captured by the union of all so-obtained inclusion assertions, plus an assertion expressing that the types mentioned in Table 1 exhaustively *cover* all possible ship types:

$$\mathit{Ship} \sqsubseteq \mathit{CCV} \sqcup \mathit{CT} \sqcup \mathit{CC} \sqcup \mathit{SPC} \sqcup \mathit{LPC} \sqcup \mathit{PP} \sqcup \mathit{PPP} \sqcup \mathit{NP} \qquad \blacksquare$$

ABox. The ABox A is a finite set of *assertions*, or *facts*, of the form $N(d)$, $P(d, d')$, or $F(d, \mathtt{v})$, with N a concept name, P a role name, F a feature, $d, d' \in \Delta$, and $\mathtt{v} \in \Delta_{\mathcal{D}_F}$.

Semantics. The semantics of an $\mathcal{ALC}(\mathfrak{D})$ KB $\mathcal{K} = \langle \Sigma, T, A \rangle$ relies, as usual, on first-order interpretations $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ over the fixed domain Δ, where $\cdot^{\mathcal{I}}$ is an interpretation function mapping each atomic concept N in T to a set $N^{\mathcal{I}} \subseteq \Delta$, $\top$ to Δ, $\bot$ to $\emptyset$, each role R to a relation $R^{\mathcal{I}} \subseteq \Delta \times \Delta$, and each feature F to a relation $F^{\mathcal{I}} \subseteq \Delta \times \Delta_{\mathcal{D}_F}$. Complex concepts are interpreted as follows:

- $(\neg C)^{\mathcal{I}} = \Delta \setminus C^{\mathcal{I}}$;
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$;
- $(\exists R.\, C)^{\mathcal{I}} = \{x \in \Delta \mid \exists y \in \Delta \text{ s.t. } \langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$;
- $(\exists F.\, r)^{\mathcal{I}} = \{x \in \Delta \mid \exists \mathtt{v} \in \Delta_{\mathcal{D}_F} \text{ s.t. } \langle x, \mathtt{v} \rangle \in F^{\mathcal{I}} \text{ and } r(\mathtt{v}) \text{ holds}\}$.

When an interpretation $\mathcal{I}$ *satisfies* an assertion is defined as follows:

$$C \sqsubseteq D \quad \text{if} \quad C^{\mathcal{I}} \subseteq D^{I}; \qquad P(d_1, d_2) \quad \text{if} \quad \langle d_1, d_2 \rangle \in P^{\mathcal{I}};$$
$$N(d) \quad \text{if} \quad d \in N^{\mathcal{I}}; \qquad F(d, \mathtt{v}) \quad \text{if} \quad \langle d, \mathtt{v} \rangle \in F^{\mathcal{I}}.$$

Finally, we say that $\mathcal{I}$ is a *model* of T if it satisfies all inclusion assertions of T, and a *model* of $\mathcal{K}$ if it satisfies all assertions of T and A.

Reasoning in $\mathcal{ALC}(\mathfrak{D})$. Reasoning in $\mathcal{ALC}$ with a single concrete domain is decidable in ExpTime (and hence ExpTime-complete) under the assumption that *(i)* the logic allows for unary concrete domain predicates only, *(ii)* the concrete domain is *admissible* [7,8], and *(iii)* checking the satisfiability of conjunctions of predicates of the datatype is decidable in ExpTime. This follows from a slightly more general result shown in [9, Sect. 2.4.1]. Admissibility requires that the set of predicate names is closed under negation and that it contains a predicate name denoting the entire domain. Hence, reasoning in $\mathcal{ALC}$ extended with one of the concrete domains used in DMN (e.g., integers or reals, with facets based on comparison predicates together with a constraining value), is ExpTime-complete. The variant of DL with concrete domains that we consider here, $\mathcal{ALC}(\mathfrak{D})$, makes only use of unary concrete domain (i.e., datatype) predicates, but allows for multiple datatypes. Hence, the above decidability results do not directly apply. However, exploiting the absence of non-unary datatype predicates, and considering that each feature is *typed* with a specified datatype, it is easy to see that the various datatypes essentially do not interact with each other, and that therefore the complexity of reasoning is not affected.

Theorem 1. *Checking satisfiability of an $\mathcal{ALC}(\mathfrak{D})$ KB is decidable in* ExpTime *(and so are the problems of deciding instance checking and subsumption w.r.t. a KB).*

Rich KBs. We also consider rich KBs where axioms are specified in full $\mathsf{FOL}(\mathfrak{D})$ (and the signature is that of a $\mathsf{FOL}(\mathfrak{D})$ theory). We call such KBs $\mathsf{FOL}(\mathfrak{D})$ *KBs.*

3.2 DMN Decision Tables

To capture the business logic of a complex decision, we rely on the DMN standard and its S-FEEL language [11]. Specifically, we resort to [5] for a formal definition of the notion of decision as specified in the standard. We do not consider priorities here. An *S-FEEL DMN decision table* $\mathcal{M}$ (called simply *decision table* in the following) is a tuple $\langle \mathit{Name}, I, O, \mathsf{AType}, \mathsf{AFacet}, R, C, H \rangle$, where:

- *Name* is the *table name.*
- I and O are disjoint, finite sets of *input* and *output attributes.*
- $\mathsf{AType} : I \uplus O \to \mathfrak{D}$ is a *typing function* that associates each input/output attribute to its corresponding data type.
- AFacet is a *facet function* that associates each input/output attribute $\mathbf{a} \in I \uplus O$ to an S-FEEL condition over $\mathsf{AType}(\mathbf{a})$ (see below).
- R is a finite set $\{r_1, \ldots, r_p\}$ of *rules.* Each rule r_k is a pair $\langle \mathsf{If}_k, \mathsf{Then}_k \rangle$, where If_k is an *input entry function* that associates each input attribute $\mathbf{a^{in}} \in I$ to an S-FEEL condition over $\mathsf{AType}(\mathbf{a^{in}})$, and Then_k is an *output entry function* that associates each output attribute $\mathbf{a^{out}} \in O$ to an object in $\mathsf{AType}(\mathbf{a^{out}})$.
- $C \in \{\mathtt{c}, \mathtt{i}\}$ is the *completeness indicator* - $\mathtt{c}$ (resp., $\mathtt{i}$) stands for *(in)complete* table.
- H is the *(single) hit indicator* defining the policy for the rule application. Since we do not focus on priorities, the interesting policies are: *(i)* $\mathtt{u}$ for *unique hit policy*, *(ii)* $\mathtt{a}$ for *any hit policy.*

In the following, we use a dot notation to single out an element of a decision table. For example, $\mathcal{M}.I$ denotes the set of input attributes for decision $\mathcal{M}$.

An *(S-FEEL) condition* $\mathcal{Q}$ over type $\mathcal{D}$ is inductively defined as follows:

- "$-$" is an S-FEEL condition representing *any value* (i.e., it evaluates to true for every object in $\Delta_{\mathcal{D}}$);
- given a constant $\mathtt{v}$, expressions "$\mathtt{v}$" and "$\mathtt{not}(\mathtt{v})$" are S-FEEL conditions respectively denoting that the value shall (not) match with $\mathtt{v}$.
- if $\mathcal{D}$ is a numerical datatype, given two numbers $\mathtt{v_1}, \mathtt{v_2} \in \Delta_{\mathcal{D}}$, the interval expressions "$[\mathtt{v_1}, \mathtt{v_2}]$", "$[\mathtt{v_1}, \mathtt{v_2})$", "$(\mathtt{v_1}, \mathtt{v_2}]$", and "$(\mathtt{v_1}, \mathtt{v_2})$" are S-FEEL conditions (interpeted in the usual, mathematical way);
- given two S-FEEL conditions $\mathcal{Q}_1$ and $\mathcal{Q}_2$, "$\mathcal{Q}_1, \mathcal{Q}_2$" is an S-FEEL condition representing their disjunction (i.e., it evaluates to true for a value $\mathtt{v} \in \Delta_{\mathcal{D}}$ if either $\mathcal{Q}_1$ or $\mathcal{Q}_2$ evaluates to true for $\mathtt{v}$).

Table 2. Decision table for determining vessel clearance in Dutch ports; symbol `today` is a shortcut for the milliseconds representing time 00:00:00 of the current date.

Vessel Clearance						
C U	***Cer. Exp.*** (date)	***Length*** (m)	***Draft*** (m)	***Capacity*** (TEU)	***Cargo*** (mg/cm^2)	***Enter***
	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	**Y**,**N**
1	$\leq$ `today`	–	–	–	–	**N**
2	$>$ `today`	<260	<10	<1000	–	**Y**
3	$>$ `today`	<260	<10	≥1000	–	**N**
4	$>$ `today`	<260	[10,12]	<4000	≤0.75	**Y**
5	$>$ `today`	<260	[10,12]	<4000	>0.75	**N**
6	$>$ `today`	[260,320)	(10,13]	<6000	≤0.5	**Y**
7	$>$ `today`	[260,320)	(10,13]	<6000	>0.5	**N**
8	$>$ `today`	[320,400)	≥13	>4000	≤0.25	**Y**
9	$>$ `today`	[320,400)	≥13	>4000	>0.25	**N**

Example 3. We use our case study to illustrate how a complex decision can be captured in DMN. Table 2 depicts the decision table for ship clearance, formalizing Sect. 2.1. The first two rows (below the table title) indicate the table meta-information. In particular, the leftmost cell indicates that the table is meant to be complete, and that rules are declared to not overlap.[5] Blue-colored cells (i.e., all other cells but the rightmost one), together with the cells below, respectively model the input attributes used to determine ship clearance, and the facets over their corresponding datatypes. In particular, the input attributes are: *(i)* the certificate expiration date, *(ii)* the length, *(iii)* the size, *(iv)* the capacity, and *(v)* the amount of cargo residuals of a ship. Such attributes are nonnegative real numbers; this is captured by typing them as reals, adding restriction "≥ 0" as facet. The rightmost, red cell represents the output attribute, i.e., whether the ship under scrutiny may enter the port. This is modeled by typing the output attribute as string, allowing only values **Y** and **N**. Every other row models a rule. The intuitive interpretation of such rules relies on the usual "if ... then ..." pattern. For example, the first rule states that if the certificate of the ship is expired, then the ship cannot enter the port (regardless of the other input attributes). The second rule, instead, states that if the ship has a valid certificate, a length shorter than 260 m, a draft smaller than 10 m, a capacity smaller than 1000 TEU, then the ship is allowed to enter the port (regardless of the cargo residuals it carries). Other rules are interpreted similarly. ■

4 Semantic Decision Models

We now combine the two knowledge sources discussed in Sect. 3.1, namely FOL($\mathfrak{D}$) knowledge bases and DMN decision tables, into an integrated *decision knowledge base* (DKB) that *empowers DMN with semantics*.

[5] Recall that such indicators are provided by the user, and may not reflect the actual table content.

4.1 Decision Knowledge Bases

The intuition behind our proposal for integration is to consider decision tables as a sort of enhancement of a KB describing a domain of interest. In this respect, a decision table $\mathcal{M}$ is linked to a specific concept. The idea is that given an object o of the specified type, $\mathcal{M}$ inspects all features of o that correspond to its input attributes $\mathcal{M}.I$, matching their values against the decision rules. Depending on which rule(s) match, $\mathcal{M}$ then dictates which are the values to which o must be connected via those features that correspond to the output attributes $\mathcal{M}.O$. Hence, the KB and the decision table "interact" on (some of) the input attributes of the decision, while the output attributes exclusively belong to the decision table, which is in fact used to infer new knowledge about the domain.

Formally, a *decision knowledge base* over datatypes $\mathfrak{D}$ ($\mathfrak{D}$-DKB, or DKB for short) is a tuple $\langle \Sigma, T, \mathcal{M}, C, A \rangle$, where:

- T is a $\mathsf{FOL}(\mathfrak{D})$ IKB with signature Σ.
- $\mathcal{M}$ is a decision table that satisfies the following two typing conditions:
  (output uniqueness) $\mathcal{M}.O \cap \Sigma = \emptyset$;
  (input type compatibility) for every binary predicate $P \in \Sigma$ whose name appears in $\mathcal{M}.I$, their types are compatible, i.e., $\mathcal{M}.\mathsf{AType}(P) = \mathsf{Sig}_P(2)$.
- $C \in \Sigma_C$ is a *bridge concept*, that is, a concept from Σ that links T with $\mathcal{M}$.
- A is an ABox over the extended signature $\Sigma \cup \mathcal{M}.I$.

When the focus is on the intensional decision knowledge only, we omit the ABox, and call the tuple $\langle \Sigma, T, \mathcal{M}, C \rangle$ intensional DKB (IDKB).

Example 4. The combination of Tables 1 and 2 using "ship" as bridge concept gives rise to a DKB for the ship clearance domain. On the one hand, Table 1 introduces different types of ships, which can be modeled as subtype concepts of the generic concept of "ship", together with a set of axioms constraining the length, draft, and capacity features depending on the specific subtype (cf. Example 1). On the other hand, Table 2 extends the signature of Table 1 with three additional features for ships, namely certificate expiration and cargo, as well as the indication of whether a ship can enter a port or not. This latter feature is the output of the decision, and is in fact inferred by applying the ship clearance decision table in the context of a specific port. ■

4.2 Formalizing DKBs

From the formal point of view, the integration between a KB and a decision table is obtained by encoding the latter into $\mathsf{FOL}(\mathfrak{D})$, consequently enriching the KB with additional axioms that capture its intended semantics. The purpose of this section is to provide such an encoding. To this end, we use the predicate logic-based formalization of DMN introduced in [5] as a starting point. However, we cannot simply rely on it, since it does not interpret input and output attributes as features of a certain type of object, but directly encodes decisions as formulae relating tuples of input values to corresponding tuples of output values.

This "objectification" is essential in our setting: it is the basis for the integration between the two sources of knowledge.

Technically, we introduce a translation function τ that transforms a DKB $\mathcal{X} = \langle \Sigma, T, \mathcal{M}, C, A \rangle$ into a corresponding $\mathsf{FOL}(\mathfrak{D})$ KB $\tau(\mathcal{X}) = \langle \Sigma', T', A \rangle$ (or an IDKB $\overline{\mathcal{X}} = \langle \Sigma, T, \mathcal{M}, C \rangle$ into a corresponding $\mathsf{FOL}(\mathfrak{D})$ IKB $\tau(\overline{\mathcal{X}})$) as follows. Signature $\Sigma' = \Sigma \cup \Sigma^i_{\mathcal{M}} \cup \Sigma^o_{\mathcal{M}}$ is the signature obtained from the original DKB by incorporating a set $\Sigma^i_{\mathcal{M}} = \{P^i/2 \mid P^i \in \mathcal{M}.I\}$ of binary predicates that account for the input attributes of the table, and a set $\Sigma^o_{\mathcal{M}} = \{P^{o,k}/2 \mid P^o \in \mathcal{M}.O, k \in \{1, \ldots, |\mathcal{M}.R|\}\}$ of predicates that account for the output attributes. Specifically, each input attribute becomes a binary predicate with the same name, while each output attribute gives rise to a series of corresponding binary predicates, one per rule in the table. In this way, an output value retains information about its provenance, i.e., which rule was applied to produce it. All so-generated predicates have first component typed with Δ, and second component typed according to the type assigned by $\mathcal{M}$ to their corresponding attribute.

TBox $T' = T \cup \tau_C(\mathcal{M})$ extends the original axioms in T with a set of additional axioms that encode $\mathcal{M}$ into $\mathsf{FOL}(\mathfrak{D})$, relativizing the encoding to the bridge concept C.

The encoding $\tau_C(\mathcal{M})$ of decision table $\mathcal{M}$ consists of the union of formulae obtained by encoding: *(i)* input/output attributes of $\mathcal{M}$; *(ii)* the facet conditions attached to such attributes; *(iii)* rules in $\mathcal{M}$.

In the following, given a predicate $P \in \Sigma^i_{\mathcal{M}} \cup \Sigma^o_{\mathcal{M}}$, we denote by $attr(P)$ the attribute in $\mathcal{M}.I \cup \mathcal{M}.O$ from which P has been obtained. In addition, given $m \in \{1, \ldots, |\mathcal{M}.R|\}$, we denote by $\Sigma^o_{\mathcal{M}}|_m = \{P^{o,k} \mid P^{o,k} \in \Sigma^o_{\mathcal{M}}, k = m\}$ the subset of $\Sigma^o_{\mathcal{M}}$ containing only the predicates associated to index m.

Encoding of attributes. For each predicate $P \in \Sigma^i_{\mathcal{M}} \cup \Sigma^o_{\mathcal{M}}$, function τ_C produces two formulae: *(i)* a typing formula $\forall x, y.P(x, y) \rightarrow C(x)$, declaring that the domain of the attribute is the bridge concept; *(ii)* a functionality formula $\forall x, y, z.P(x, y) \wedge P(x, z) \rightarrow x = z$, declaring that every object of the bridge concept cannot be connected to more than one value through P. If $attr(P)$ is an input attribute, functionality guarantees that the application of the decision table is unambiguous. If $attr(P)$ is an output attribute, functionality simply captures that there is a single value present in an output cell of the decision table. In general, multiple outputs for the same column may in fact be obtained when applying a decision, but if so, they would be generated by different rules.

Encoding of facet conditions. For each attribute predicate $P \in \Sigma^i_{\mathcal{M}} \cup \Sigma^o_{\mathcal{M}}$, function τ_C produces the facet formula imposing that the range of the predicate must satisfy the restrictions imposed by the S-FEEL condition attached to attribute $attr(P)$:

$$\forall x, y.P(x, y) \rightarrow \tau^y(\mathcal{M}.\mathsf{AFacet}(attr(P))),$$

where, given an S-FEEL condition Q, function $\tau^x(Q)$ builds a unary $\mathsf{FOL}(\mathfrak{D})$ formula that encodes the application of Q to x. This is defined as follows:

$$\tau^x(Q) \triangleq \begin{cases} \mathit{true} & \text{if } \mathcal{Q} = \text{“}-\text{”} \\ x \neq \mathtt{v} & \text{if } \mathcal{Q} = \text{“}\mathtt{not(v)}\text{”} \\ x = \mathtt{v} & \text{if } \mathcal{Q} = \text{“}\mathtt{v}\text{”} \\ x \approx \mathtt{v} & \text{if } \mathcal{Q} = \text{“}\approx \mathtt{v}\text{” and } \approx \in \{<, >, \leq, \geq\} \\ x > \mathtt{v}_1 \wedge x < \mathtt{v}_2 & \text{if } \mathcal{Q} = \text{“}(\mathtt{v}_1..\mathtt{v}_2)\text{”} \\ \ldots & \text{(similarly for the other types of intervals)} \\ \tau^x(\mathcal{Q}_1) \vee \tau^x(\mathcal{Q}_2) & \text{if } \mathcal{Q} = \text{“}\mathcal{Q}_1, \mathcal{Q}_2\text{”} \end{cases}$$

Example 5. Consider the length attribute in Table 2. Assuming that *Ship* acts as bridge concept, its typing and facet $\mathsf{FOL}(\mathfrak{D})$ formulae are:

$$\forall x, y.\mathit{length}(x, y) \rightarrow \mathit{Ship}(x). \qquad \forall x, y.\mathit{length}(x, y) \rightarrow y \geq 0. \qquad \blacksquare$$

Encoding of rules. Each rule is translated into a formula expressing that whenever an object belonging to the bridge concept is related, via predicates accounting for the input attributes, to values that satisfy the S-FEEL conditions associated by the rule to such attributes, then the same object must be related, via predicates accounting for the output attributes, to the values associated by the rule to such attributes. Formally, fix an ordering over the rules $\mathcal{M}.R$. For every $m \in \{1, \ldots, |\mathcal{M}.R|\}$, given the m-rule $r_m = \langle \mathsf{If}, \mathsf{Then} \rangle \in R$, function τ_C produces:

$$\forall x, \boldsymbol{y}. \bigwedge_{P_j^i \in \Sigma^i_{\mathcal{M}}} \left(P_j^i(x, y_j) \wedge \tau^{y_j}\left(\mathit{attr}(\mathsf{If}(P_j^i))\right)\right) \rightarrow \bigwedge_{P_k^{o,m} \in \Sigma^o_{\mathcal{M}|m}} \left(\exists z_k . P_k^{o,m} \wedge \tau^{z_k}\left(\mathsf{Then}(\mathit{attr}(P_k^{o,m}))\right)\right)$$

Example 6. Rule 2 in Table 2 is encoded in $\mathsf{FOL}(\mathfrak{D})$ as:

$$\forall x, e, l, d, c.\ \mathit{exp}(x, e) \wedge e > \mathtt{today} \wedge \mathit{length}(x, l) \wedge l < \mathtt{260} \wedge \mathit{draft}(x, d) \wedge d < \mathtt{10} \wedge \mathit{cap}(x, c) \wedge c < \mathtt{1000} \rightarrow \exists o.\mathit{enter}_2(x, o) \wedge o = \mathtt{Y}.$$

where enter_2 is obtained from output attribute **enter** in the context of Rule 2. ■

We close this section by arguing that our encoding can be seen as a sort of “objectification” of the encoding in [5], where a tuple of values is now reified into an explicit object, together with corresponding predicates pointing to the different tuple components.

4.3 Reasoning Tasks

We formally revisit the main reasoning tasks introduced in [5] for DMN, in the presence of background knowledge. Such reasoning tasks aim at understanding whether the metadata attached to a DMN decision to indicate completeness and hit policies, indeed reflect the semantics of the DKB of interest. In the following, we fix a DKB $\mathcal{X} = \langle \Sigma, T, \mathcal{M}, C, A \rangle$, and denote by $\overline{\mathcal{X}} = \langle \Sigma, T, \mathcal{M}, C \rangle$ its corresponding IDKB.

I/O relationship. The first and most fundamental reasoning task is to check whether the DKB induces a certain input/output relationship over a given object. The *I/O relationship problem* for DKBs is defined as follows:

Input: (i) DKB $\mathcal{X}$, *(ii)* object $\mathtt{c} \in \Delta$ of type C, *(iii)* output attribute $P^o \in \mathcal{M}.O$, *(iv)* value $\mathtt{v} \in \mathcal{M}.\mathsf{AType}(P^o)$.
Question: Is it the case that $\mathcal{X}$ assigns output $\mathtt{v}$ for attribute P^o to object $\mathtt{c}$? Formally, this amounts to check whether fact $P^{o,k}(\mathtt{c},\mathtt{v})$ is implied for some rule k, i.e., whether: $\tau(\mathcal{X}) \models \bigvee_{P^{o,k} \in \Sigma^o_{\mathcal{M}}} P^{o,k}(\mathtt{c},\mathtt{v})$.

Table completeness. Completeness is declared by setting $\mathcal{M}.C = \mathtt{c}$, and indicates that the rules in $\mathcal{M}$ cover all possible configurations for the input values. The *table completeness problem* is then defined as follows:

Input: IDKB $\overline{\mathcal{X}}$.
Question: Is it the case that at least one rule of $\mathcal{M}$ is guaranteed to trigger? Formally:

$$\tau(\overline{\mathcal{X}}) \models \forall x, \boldsymbol{y}. \bigvee_{\langle \mathsf{If},\mathsf{Then}\rangle \in \mathcal{M}.R} \bigwedge_{P^i_j \in \Sigma^i_{\mathcal{M}}} \Big(P^i_j(x, y_j) \rightarrow \tau^{y_j}(\mathsf{If}(attr(P^i_j)))\Big)?$$

Correctness of unique hit. Unique hit is declared by setting $\mathcal{M}.H = \mathtt{u}$, and indicates that at most one rule of $\mathcal{M}$ may trigger on a given input object. The *correctness of unique hit problem* is hence defined as follows:

Input: IDKB $\overline{\mathcal{X}}$.
Question: Is it the case that rules in $\mathcal{M}$ do not overlap? Formally: is it the case that, for every pair $\langle \mathsf{If}_1, \mathsf{Then}_1\rangle$ and $\langle \mathsf{If}_2, \mathsf{Then}_2\rangle$ of rules in $\mathcal{M}.R$,

$$\tau(\overline{\mathcal{X}}) \models \forall x, \boldsymbol{y}. \bigvee_{P^i_j \in \mathcal{M}.I} \Big(P^i_j(x, y_j) \rightarrow \neg\big(\bigwedge_{k \in \{1,2\}} \tau^{y_j}(\mathsf{If}_k(attr(P^i_j)))\big)\Big)?$$

Correctness of any hit. Any hit is declared by setting $\mathcal{M}.H = \mathtt{a}$, and states that whenever multiple rules may simultaneously trigger, they need to agree on the produced output. In this light, checking whether this policy is correct can be directly reduced to the case of unique hit, but considering only those pairs of rules that differ in output.

DMN reasoning tasks. We conclude by pointing out that, in the case where background knowledge is absent, i.e., $T = \emptyset$, the different reasoning tasks reduce to the case of pure DMN (as defined in [5]). The reduction is direct for table completeness, and also for checking correctness of unique/any hit. Checking I/O relationship is obtained instead by fixing A to contain exactly the following facts: *(i)* a fact $C(\mathtt{c})$ where $\mathtt{c}$ is an arbitrary object from Δ; *(ii)* a set of facts of the form $\{P^i_j(\mathtt{c},\mathtt{v}_j) \mid P^i_j \in \mathcal{M}.I\}$, denoting the assignment of input attributes for $\mathtt{c}$ to the corresponding values of interest.

5 Reasoning on Decision Knowledge Bases

While the translation from DKBs to $\mathsf{FOL}(\mathfrak{D})$ presented in Sect. 4.2 provides the logic-based semantics of DKBs, it does not give any insight on how to actually approach the different reasoning tasks of Sect. 4.3. Obviously, decidability and complexity of such reasoning tasks depend on the background knowledge and on the decision component. Since the decision component comes with the fixed S-FEEL language, we approach this problem as follows. First, we show that DMN decision tables based on S-FEEL can be encoded in $\mathcal{ALC}(\mathfrak{D})$. Then, we show that all reasoning tasks defined in Sect. 4.3 can be reduced to (un)satisfiability of an $\mathcal{ALC}(\mathfrak{D})$ concept w.r.t. a KB consisting of the union of the background knowledge with the $\mathcal{ALC}(\mathfrak{D})$ formalization of the decision table. We consequently obtain that such satisfiability checks can be carried out in ExpTime, whenever the background knowledge is expressed in $\mathcal{ALC}(\mathfrak{D})$.

5.1 Encoding Decision Tables in $\mathcal{ALC}(\mathfrak{D})$

We revisit the translation from DKBs to $\mathsf{FOL}(\mathfrak{D})$ introduced in Sect. 4.2, showing that the translation of decision tables can be reformulated so as to obtain an $\mathcal{ALC}(\mathfrak{D})$ IKB. Given a bridge concept C and a decision table $\mathcal{M}$, we introduce a translation function ρ_C that encodes $\mathcal{M}$ into the corresponding $\mathcal{ALC}(\mathfrak{D})$ IKB $\rho_C(\mathcal{M}) = \langle \Sigma_\mathcal{M}, T_\mathcal{M} \rangle$, using C to provide a context for the encoding. Specifically, the signature of the target IKB is simply obtained from the bridge concept and the input/output attributes of $\mathcal{M}$, adopting exactly the same strategy followed for the encoding into $\mathsf{FOL}(\mathfrak{D})$: input attributes become binary predicates, and output attributes become binary predicates relativized w.r.t. the different rules present in $\mathcal{M}$. In formulae, $\Sigma_\mathcal{M} = C \cup \Sigma^i_\mathcal{M} \cup \Sigma^o_\mathcal{M}$. The encoding of $T_\mathcal{M}$ reconstructs that of Sect. 4.2, and in fact deals with: *(i)* input/output attributes of $\mathcal{M}$; *(ii)* the facet conditions attached to such attributes; *(iii)* rules in $\mathcal{M}$.

Encoding of attributes. For each attribute $P \in \Sigma^i_\mathcal{M} \cup \Sigma^o_\mathcal{M}$, function ρ_C produces the typing axiom $\exists P \sqsubseteq C$. Note that functionality is not explicitly asserted, since $\mathcal{ALC}(\mathfrak{D})$ features are functional by default.

Encoding of facet conditions. For each attribute $P \in \Sigma^i_\mathcal{M} \cup \Sigma^o_\mathcal{M}$, function ρ_C produces a derived datatype declaration of the form: $C \sqsubseteq \rho^P(\mathcal{M}.\mathsf{AFacet}(attr(P)))$, where, given an S-FEEL condition Q, and assuming that $\mathcal{M}.\mathsf{AType} = type$, $\rho^P(Q)$ builds an $\mathcal{ALC}(\mathfrak{D})$ concept application of Q to x. This is defined as follows:

$$\rho^P(Q) \triangleq \begin{cases} \top & \text{if } \mathcal{Q} = \text{“}-\text{”} \\ \neg\exists P.type[=_\mathtt{v}] & \text{if } \mathcal{Q} = \text{“}\mathtt{not(v)}\text{”} \\ \forall P.type[=_\mathtt{v}] & \text{if } \mathcal{Q} = \text{“}\mathtt{v}\text{”} \\ \forall P.type[COP_\mathtt{v}] & \text{if } \mathcal{Q} = \text{“}COP\ \mathtt{v}\text{” and } COP \in \{<, >, \leq, \geq\} \\ \forall P.type[>_{\mathtt{v}_1} \wedge <_{\mathtt{v}_2}] & \text{if } \mathcal{Q} = \text{“}(\mathtt{v}_1..\mathtt{v}_2)\text{”} \\ \ldots & \text{(similarly for the other types of intervals)} \\ \rho^P(Q_1) \sqcup \rho^P(Q_2) & \text{if } \mathcal{Q} = \text{“}\mathcal{Q}_1, \mathcal{Q}_2\text{”} \end{cases}$$

Example 7. Consider the length attribute in Table 2. Assuming that *Ship* acts as bridge concept, its typing and facet $\mathsf{FOL}(\mathfrak{D})$ formulae are $\exists \mathit{length} \sqsubseteq \mathit{Ship}$ and $\mathit{Ship} \sqsubseteq \forall \mathit{length}.\mathit{real}[>_0]$. ■

Encoding of rules. Fix an ordering over the rules $\mathcal{M}.R$. For every $m \in \{1, \ldots, |\mathcal{M}.R|\}$, given the m-rule $r_m = \langle \mathsf{If}, \mathsf{Then} \rangle \in R$, function ρ_C produces an inclusion assertion of the form:

$$\prod_{P_j^i \in \Sigma_{\mathcal{M}}^i} \left(\rho^{P_j^i} \left(\mathit{attr}(\mathsf{If}(P_j^i)) \right) \right) \sqsubseteq \prod_{P_k^{o,m} \in \Sigma_{\mathcal{M}}^o|_m} \left(\exists P_k^{o,m} \sqcap \rho^{P_k^{o,m}} \left(\mathsf{Then}(\mathit{attr}(P_k^{o,m})) \right) \right)$$

Example 8. Rule 2 in Table 2 is encoded in $\mathcal{ALC}(\mathfrak{D})$ as:

$$\forall \mathit{exp}.\mathit{real}[>_{\mathtt{today}}] \sqcap \forall \mathit{length}.\mathit{real}[<_{260}] \sqcap \forall \mathit{draft}.\mathit{real}[<_{10}] \sqcap \forall \mathit{cap}.\mathit{real}[<_{1000}] \sqsubseteq \exists \mathit{enter}_2 \sqcap \forall \mathit{enter}_2.\mathit{string}[=_{\mathtt{Y}}]$$ ■

Thanks to the fact that $\mathcal{ALC}(\mathfrak{D})$ can be seen as a fragment of $\mathsf{FOL}(\mathfrak{D})$, we can directly establish that the $\mathcal{ALC}(\mathfrak{D})$ encoding of decision tables is indeed correct.

Theorem 2. *For every decision table $\mathcal{M}$, and (bridge) concept C, we have that the $\mathsf{FOL}(\mathfrak{D})$ IKB $\tau_C(\mathcal{M})$ is* logically equivalent *to the $\mathcal{ALC}(\mathfrak{D})$ IKB $\rho_C(\mathcal{M})$.*

Proof. Direct by construction of the translation functions τ_C and ρ_C, noting that, once the standard $\mathsf{FOL}(\mathfrak{D})$ encoding of $\mathcal{ALC}(\mathfrak{D})$ is applied to the $\mathcal{ALC}(\mathfrak{D})$ IKB $\rho_C(\mathcal{M})$, it becomes syntactically identical to the $\mathsf{FOL}(\mathfrak{D})$ IKB $\tau_C(\mathcal{M})$. □

5.2 Reasoning over $\mathcal{ALC}(\mathfrak{D})$ Decision Knowledge Bases

In this section, we leverage the possibility of encoding decision tables into $\mathcal{ALC}(\mathfrak{D})$ so as to obtain a characterization of the decidability and complexity of reasoning in the case of $\mathcal{ALC}(\mathfrak{D})$ *DKBs*, i.e., DKBs whose background knowledge is specified in $\mathcal{ALC}(\mathfrak{D})$. Formally, a DKB $\mathcal{X} = \langle \Sigma, T, \mathcal{M}, C, A \rangle$ is an $\mathcal{ALC}(\mathfrak{D})$ *DKB* if $\langle \Sigma, T, A \rangle$ is an $\mathcal{ALC}(\mathfrak{D})$ KB. We extend the translation function of Sect. 5 to handle the entire $\mathcal{ALC}(\mathfrak{D})$ KB as follows: $\rho(\mathcal{X}) = \langle \Sigma \cup \Sigma_{\mathcal{M}}, T \cup T_{\mathcal{M}}, A \rangle$, where $\langle \Sigma_{\mathcal{M}}, T_{\mathcal{M}} \rangle = \rho_C(\mathcal{M})$ (similarly for an $\mathcal{ALC}(\mathfrak{D})$ IKB). With these notions at hand, we show the following.

Theorem 3. *The I/O relationship, table completeness, and correctness of unique hit problems can all be decided in* ExpTime *for $\mathcal{ALC}(\mathfrak{D})$ DKBs.*

Proof. The proof is based on a reduction from the three decision problems to a polynomial number of instance or subsumption checks w.r.t. an $\mathcal{ALC}(\mathfrak{D})$ KB, which can be decided in ExpTime (cf. Theorem 1). Let $\mathcal{X} = \langle \Sigma, T, \mathcal{M}, C, A \rangle$ be an $\mathcal{ALC}(\mathfrak{D})$ DKB, and let $\overline{\mathcal{X}} = \langle \Sigma, T, \mathcal{M}, C \rangle$ be its corresponding IDKB.

(*I/O relationship*) Fix an ordering over the rules $\mathcal{M}.R$. Given *(i)* $\mathcal{X}$, *(ii)* object $\mathsf{c} \in \Delta$ of type C, *(iii)* output attribute $P^o \in \mathcal{M}.O$, and *(iv)* value

$\mathtt{v} \in \mathcal{M}.\mathsf{AType}(P^o)$, we have that $\mathcal{X}$ assigns output $\mathtt{v}$ for attribute P^o to object $\mathtt{c}$ iff there exists $k \in \{1, \ldots, |\mathcal{M}.R|\}$ such that instance checking for fact $P^{o,k}(\mathtt{c}, \mathtt{v})$ w.r.t. KB $\rho(\mathcal{X})$ succeeds.

(*Table completeness*) Decision rules in $\overline{\mathcal{X}}$ are complete iff the following $\mathcal{ALC}(\mathfrak{D})$ subsumption holds with respect to KB $\rho(\overline{\mathcal{X}})$:

$$\top \sqsubseteq \bigsqcup_{\langle \mathsf{If}, \mathsf{Then} \rangle \in \mathcal{M}.R} \prod_{P^i_j \in \Sigma^i_{\mathcal{M}}} \left(\rho^{P^i_j}(\mathsf{If}(attr(P^i_j))) \right)$$

(*Unique hit*) Decision rules in $\overline{\mathcal{X}}$ do not overlap iff for every pair $\langle \mathsf{If}_1, \mathsf{Then}_1 \rangle$ and $\langle \mathsf{If}_2, \mathsf{Then}_2 \rangle$ of rules in $\mathcal{M}.R$, the following subsumption holds w.r.t KB $\rho(\overline{\mathcal{X}})$:

$$\top \sqsubseteq \bigsqcup_{P^i_j \in \mathcal{M}.I} \neg \prod_{k \in \{1,2\}} \rho^{P^i_j}(\mathsf{If}_k(attr(P^i_j)))$$

□

Example 9. As discussed in Example 2, the ship ontology in Table 1 can be formalized in $\mathcal{ALC}(\mathfrak{D})$. Hence, the maritime security DKB of Example 4 is actually an $\mathcal{ALC}(\mathfrak{D})$ DKB. Thanks to Theorem 3, standard $\mathcal{ALC}(\mathfrak{D})$ reasoning tasks can then be used to check that such a DKB guarantees table completeness and the correctness of the unique hit indicators, as specified in Table 2. Recall that completeness holds because the table is interpreted w.r.t. the ship ontology. ■

6 Conclusions

In this work, we have provided a threefold contribution to the area of decision management, recently revived by the introduction of the DMN OMG standard. First, we have introduced decision knowledge bases (DKBs) as a framework to integrate DMN decision tables with background knowledge, captured by means of a DL KB. Second, we have formalized the framework, as well as different fundamental reasoning tasks, in $\mathsf{FOL}(\mathfrak{D})$. Third, we have shown that, in the case where background knowledge is expressed in $\mathcal{ALC}(\mathfrak{D})$, all such reasoning tasks are decidable in ExpTime. Before delving into the implementation of such reasoning tasks, we are interested in refining the analysis of their complexity, by varying the DL used to capture the background knowledge. On the one hand, we argue that DMN decision tables can actually be integrated with more expressive DLs, such as OWL 2, by retaining the complexity of reasoning that comes with the DL. On the other hand, we note that the DL encoding of DMN decision tables falls within the lightweight fragment of $\mathcal{ALC}(\mathfrak{D})$ constituted by the $\mathit{DL\text{-}Lite}^{(\mathcal{HN})}_{bool}(\mathfrak{D})$ logic extended with qualified existentials on the left-hand side of inclusions. This logic has been very recently introduced in [1, Sect. 4.3], and although upper bounds for the standard DL reasoning services are not yet established for such logic, we conjecture that it is strictly less complex than $\mathcal{ALC}(\mathfrak{D})$. This paves the way towards the study of lightweight DKBs.

Acknowledgements. This work is supported by the Euregio IPN12 project *Knowledge-Aware Operational Support* (KAOS), funded by the "European Region Tyrol-South Tyrol-Trentino" (EGTC) under the first call for basic research projects, and by the unibz project *Knowledge-driven ENterprise Distributed cOmputing* (KENDO).

References

1. Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V., Savković, O.: Datatypes in DL-Lite. Technical report KRDB17-1, KRDB Research Centre for Knowledge and Data, Free Univ. of Bozen-Bolzano. https://www.inf.unibz.it/krdb/pub/TR/KRDB17-1.pdf
2. Artale, A., Kontchakov, R., Ryzhikov, V.: DL-Lite with attributes and datatypes. In: Proceedings of ECAI, pp. 61–66 (2012)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2nd edn. (2007)
4. Batoulis, K., Meyer, A., Bazhenova, E., Decker, G., Weske, M.: Extracting decision logic from process models. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 349–366. Springer, Cham (2015). doi:10.1007/978-3-319-19069-3_22
5. Calvanese, D., Dumas, M., Laurson, Ü., Maggi, F.M., Montali, M., Teinemaa, I.: Semantics and analysis of DMN decision tables. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 217–233. Springer, Cham (2016). doi:10.1007/978-3-319-45348-4_13
6. Enderton, H.B.: A Mathematical Introduction to Logic, 2nd edn. Academic Press (2001)
7. Haarslev, V., Möller, R., Wessel, M.: The description logic $\mathcal{ALCNH}_{R+}$ extended with concrete domains: a practically motivated approach. In: Proceedings of IJCAR, pp. 29–44 (2001)
8. Horrocks, I., Sattler, U.: Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In: Proceedings of IJCAI, pp. 199–204 (2001)
9. Lutz, C.: Description logics with concrete domains-A survey. Advances in Modal Logic, pp. 265–206 (2002)
10. Motik, B., Parsia, B., Patel-Schneider, P.F.: OWL 2 Web Ontology Language structural specification and functional-style syntax, 2nd edn. W3C Recommendation, W3C, December 2012. http://www.w3.org/TR/owl2-syntax/
11. Object Management Group: Decision Model and Notation (DMN) 1.0 (2015). http://www.omg.org/spec/DMN/1.0/
12. Pawlak, Z.: Decision tables - a rough set approach. Bull. EATCS **33**, 85–95 (1987)
13. Pooch, U.W.: Translation of decision tables. ACM Comput. Surv. **6**(2), 125–151 (1974)
14. Savkovic, O., Calvanese, D.: Introducing datatypes in DL-Lite. In: Proceedings of ECAI, pp. 720–725 (2012)
15. Vanthienen, J., Dries, E.: Illustration of a decision table tool for specifying and implementing knowledge based systems. Int. J. Artif. Intell. Tools **3**(2), 267–288 (1994)
16. Vanthienen, J., Mues, C., Aerts, A.: An illustration of verification and validation in the modelling phase of KBS development. DKE **27**(3), 337–352 (1998)
17. Zaidi, A.K., Levis, A.H.: Validation and verification of decision making rules. Automatica **33**(2), 155–169 (1997)