Semantic Web 0 (0) 1
IOS Press

Model Outlines: a Visual Language for DL Concept Descriptions [1]

Editor(s): Thomas Lukasiewicz, Oxford University, UK
Solicited review(s): Natalya Keberle, Zaporozhye National University, Ukraine; Gergely Lukácsy, CISCO Systems, Ireland; Domenico Lembo, Sapienza Università di Roma, Italy

Fernando Náufel do Amaral
LLaRC – Laboratório de Lógica e Representação do Conhecimento,
Depto. de Física e Matemática, Pólo Universitário de Rio das Ostras,
Universidade Federal Fluminense, Rio das Ostras, RJ, Brazil
Email: fnaufel@gmail.com

Abstract. The development and use of ontologies may require users with no training in formal logic to handle complex concept descriptions. To aid such users, we propose a new visualization framework called "model outlines", where more emphasis is placed on the *semantics* of concept descriptions than on their *syntax*. We present a rigorous definition of our visual language, as well as detailed algorithms for translating between model outlines and the Description Logic $\mathcal{ALCN}$. We have recently conducted a usability study comparing model outlines and Manchester OWL; here, we report on its results, which indicate the potential benefits of our visual language for understanding concept descriptions.

Keywords: Description Logics, visual languages, diagrammatic reasoning, usability

1. Introduction

When working with formal ontologies, one often needs to formally represent conditions for membership in the defined classes. In this paper, we will call such conditions *concept descriptions*, following the Description Logic (DL) tradition [1].

Concept descriptions are important in many scenarios related to ontology development and use. For example, DL reasoners perform logical inferences by manipulating concept descriptions according to a specific deductive calculus. In many cases, users may be interested not only in the answers provided by such reasoners, but also in the chains of reasoning that led to those answers. In order to understand such chains of reasoning, users must be able to understand the meaning of the concept descriptions involved. This area of study is referred to as *proof explanation* [17].

Another situation where concept descriptions play a role is in *ontology query languages* [2]; for example, in the DL Query Tab Protégé plugin [5], a concept description C is given to a reasoner, which then returns items of information about C, such as all its instances, named subclasses, superclasses and equivalent classes.

Because many users of formal ontologies have no specific training in logic, the problem of representing concept descriptions in a user-friendly fashion is an important one, and many researchers have proposed different ways of solving it: replacing logical symbols with keywords in DL languages [12], automatically generating natural language paraphrases of concept descriptions [10], or using diagrammatic representations [11,15].

As an example to make this discussion more concrete, consider the following concept description in DL syntax (to be formally introduced in Sect. 2 below),

[1] Work partially supported by research grant E-26/112.038/2008 from FAPERJ.

which appears in [4], a paper about proof explanation:

$$\begin{aligned} &\exists \mathit{hasChild}.\top \\ &\sqcap \forall \mathit{hasChild}. \\ &\quad \neg((\exists \mathit{hasChild}.\neg \mathit{Doctor}) \\ &\qquad \sqcup (\exists \mathit{hasChild}.\mathit{Lawyer})) \end{aligned} \tag{1}$$

(This description represents individuals having one or more children, such that these children, in turn, have no children who are not doctors, and no children who are lawyers.)

Diagrammatic representations of concept descriptions have given rise to implementations of "visual" ontology browsers. One such example is the visualization tool GrOWL [16], which produces the diagram in Figure 1 for the concept description in (1). As can be seen, the diagram is essentially an abstract syntax tree, which offers nonspecialist users little help in understanding the semantics of the description, especially if those users are not familiar with the DL operators "∃", "∀", "¬" and "⊔". In fact, we have found this to be a common phenomenon: many visualization frameworks for concept descriptions are too faithful to the *syntax* of the representation languages (e.g., DL), a feature which may prevent users from grasping the *semantics* of the concept descriptions.

This paper discusses *model outlines*, which depart from the syntax-based tradition in that they consist of diagrams characterizing the class of *models* of a given concept description — here, we use the term "model" in the logical sense; see, e.g., [1]. The model outline for (1), produced after applying a carefully defined set of simplification rules to the original concept description (see Section 4), is presented in Figure 2. By adhering to some simple graphical conventions, such as the convention that *figures drawn in dashed stroke represent elements whose existence is optional*, a user can understand that the concept description represents a set of individuals having at least one child and having as grandchildren (if any) only doctors and non-lawyers.

Our previous papers [6,8] introduced the first version of model outlines and compared them to natural language paraphrases of concept descriptions. Since then [7], we have reformulated the visual language so as to make it more intuitive (e.g., including optional labeled clusters, rendering cardinality restrictions as text and fine-tuning the placement of inner boxes). We have also altered the conversion algorithms to conform to the new visual language.

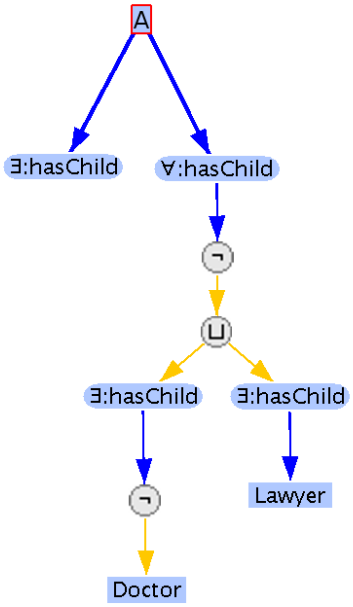

Fig. 1. Diagram produced by GrOWL [16] (manually laid out)

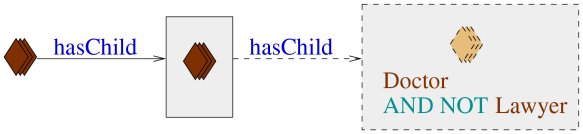

Fig. 2. Model outline for description (1)

Most importantly, we have conducted a first usability test of model outlines, with promising results. Users from different backgrounds were shown concept descriptions in two formalisms: our model outlines and Manchester OWL (a textual notation for DL which uses keywords for logical symbols, infix notation for restrictions, syntax highlighting and indentation in order to make descriptions more readable for nonspecialists — see [12]). We then tested ease of understanding for each formalism by asking the users questions about the concept descriptions shown.

This paper is structured as follows: Sect. 2 presents the syntax of model outlines for the Description Logic $\mathcal{ALCN}$, at the concrete (token) and at the abstract (type) levels, as is recommended for diagrammatic systems [14]; Sect. 3 defines the precise semantics of model outlines, in the form of algorithms that translate from model outlines to $\mathcal{ALCN}$ concept descriptions; Sect. 4 discusses the translation of $\mathcal{ALCN}$ concept de-

DL	Manchester	Meaning
$C, D \rightarrow A$	A	$\mathcal{I}(A)$
$\mid \top$	**THING**	Δ
$\mid \bot$	**NOTHING**	$\varnothing$
$\mid \neg C$	**NOT** C	$\Delta - \mathcal{I}(C)$
$\mid C \sqcap D$	C **AND** D	$\mathcal{I}(C) \cap \mathcal{I}(D)$
$\mid C \sqcup D$	C **OR** D	$\mathcal{I}(C) \cup \mathcal{I}(D)$
$\mid \forall R.C$	R **ONLY** C	$\{a \in \Delta \mid \forall b.[(a,b) \in \mathcal{I}(R) \Rightarrow b \in \mathcal{I}(C)]\}$
$\mid \exists R.C$	R **SOME** C	$\{a \in \Delta \mid \exists b.[(a,b) \in \mathcal{I}(R) \wedge b \in \mathcal{I}(C)]\}$
$\mid \leq n.R$	R **MAX** n	$\{a \in \Delta \mid \#\{b \mid (a,b) \in \mathcal{I}(R)\} \leq n\}$
$\mid \geq n.R$	R **MIN** n	$\{a \in \Delta \mid \#\{b \mid (a,b) \in \mathcal{I}(R)\} \geq n\}$
$\mid = n.R$	R **EXACTLY** n	$\{a \in \Delta \mid \#\{b \mid (a,b) \in \mathcal{I}(R)\} = n\}$

Fig. 3. $\mathcal{ALCN}$ concept descriptions and their meanings

scriptions to model outlines; Sect. 5 reports and analyzes the results of the usability test; Sect. 6 considers some specific aspects and possible applications of our visual language; Sect. 7 contains our concluding remarks; an appendix provides a proof of correctness of the translation algorithms.

2. Syntax of model outlines

We consider the Description Logic $\mathcal{ALCN}$, whose language of concept descriptions is specified in Figure 3, both in the DL syntax and in Manchester OWL. (Work is under way to define model outlines for more expressive languages, such as the concept language underlying OWL 2 DL [13].) In Figure 3, A stands for a class name (i.e., an atomic concept term), R stands for a property name (i.e., an atomic role term), and n represents a natural number. The (set-theoretical) meaning of these descriptions is given by a nonempty set Δ (the *universe* or *domain*) along with an interpretation $\mathcal{I}$ mapping each concept description C to a set $\mathcal{I}(C) \subseteq \Delta$, and each role term R to a binary relation $\mathcal{I}(R) \subseteq \Delta \times \Delta$. An interpretation $\mathcal{I}$ must map each description in the first two columns to the set in the third column. $\#S$ denotes the cardinality of a set S. A *literal* is $\top$, or $\bot$, or a description of the form A or of the form $\neg A$, where A is an atomic concept term.

The *concrete syntax* of model outlines defines their physical representation. What follows is an informal definition.

A model outline contains *clusters* (*solid* or *dashed*), *arrows* (*solid* or *dashed*) and *boxes*. We follow the symbology established by the Protégé[1] user interfaces in representing individuals as dark-colored diamonds. When cardinality restrictions are not present, the number of individuals satisfying certain conditions is not important; therefore, so as not to mislead users into thinking that only one individual is allowed in a certain situation, we show a little cluster of diamonds.

The *root* of the model outline is a solid cluster. A cluster may have an optional *class label* below it, consisting of a disjunction or of a conjunction of literals. So may a box. A box may also have an optional *cardinality label* below it, which may be of the form "(from m thru n)", "(m or more)", or "(exactly m)", with m, n natural numbers, $m < n$. The *source* of an arrow may be a cluster or a box. The *target* of an arrow is always a box. Each box is the target of exactly one arrow. An arrow must have a *role label* above it, consisting of a role name. A box *contains* one or more clusters, according to constraints that we do not include in this informal description, but which will be made explicit in the abstract syntax below. A box may also contain at most one *"among-which" inner box*, which in turn contains one or more clusters, all of them solid. Inner boxes are never the source of arrows. A box or a cluster may have a *case widget* above it.

Figure 4 shows an example model outline. The target box of the arrow labeled "*hasAttendance*" has both a class label ("*Enrolled*") and a cardinality label ("from 10 to 50"). The target box of the arrow labeled "*hasAttendance*" also has an "among-which" inner box. This model outline does not have case widgets.

At this point, the reader should test the appropriateness of the choice of visual presentation of the components of model outlines. We suggest that the reader (without any further knowledge of the meaning of

[1] http://protege.stanford.edu

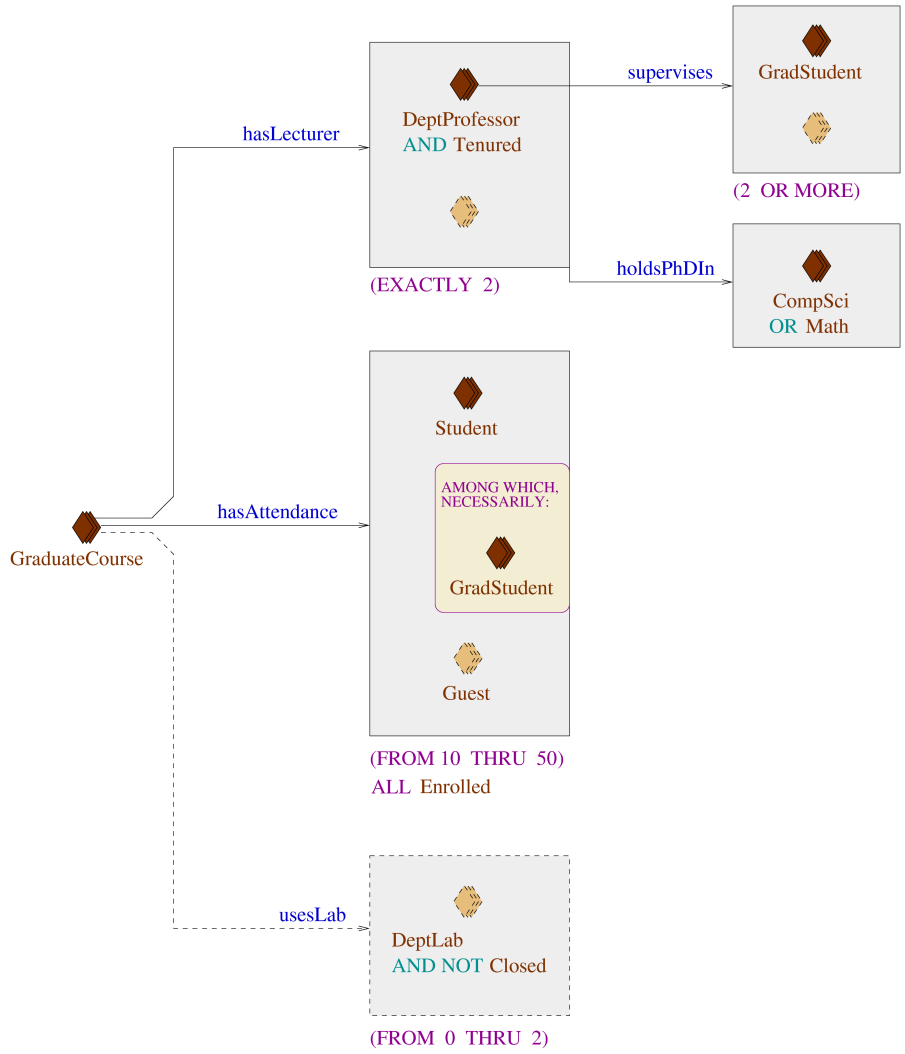

Fig. 4. Example model outline

these components) formulate a natural language description of the constraints imposed upon the individuals of class *GraduateCourse* at the root of the outline. If the reader is knowledgeable in DL syntax, the reader should also produce an $\mathcal{ALCN}$ concept description. In Sect. 3 below, we explain the precise meaning of this model outline, and in Sect. 4 we show the steps involved in its construction.

Case widgets indicate alternatives (i.e., disjunction). If a cluster or a box has a case widget above it, the user may browse the different cases interactively, one case at a time, by clicking on the triangles on either side of the case widget.

In Figure 5, for example, there are 4 cases altogether, specifying objects that satisfy one or more of the following conditions:

1. *Textbook*s having all extras (if any) translated to *Portuguese* (and possibly other languages), or
2. *Textbook*s having all extras (if any) in *Audio* format (and possibly other formats), or
3. *ClassNotes* having at least one *Free* copy in *PDF* format (and possibly other formats, and other copies), or
4. *ClassNotes* having at least one *Low*-priced copy (and possibly other copies).

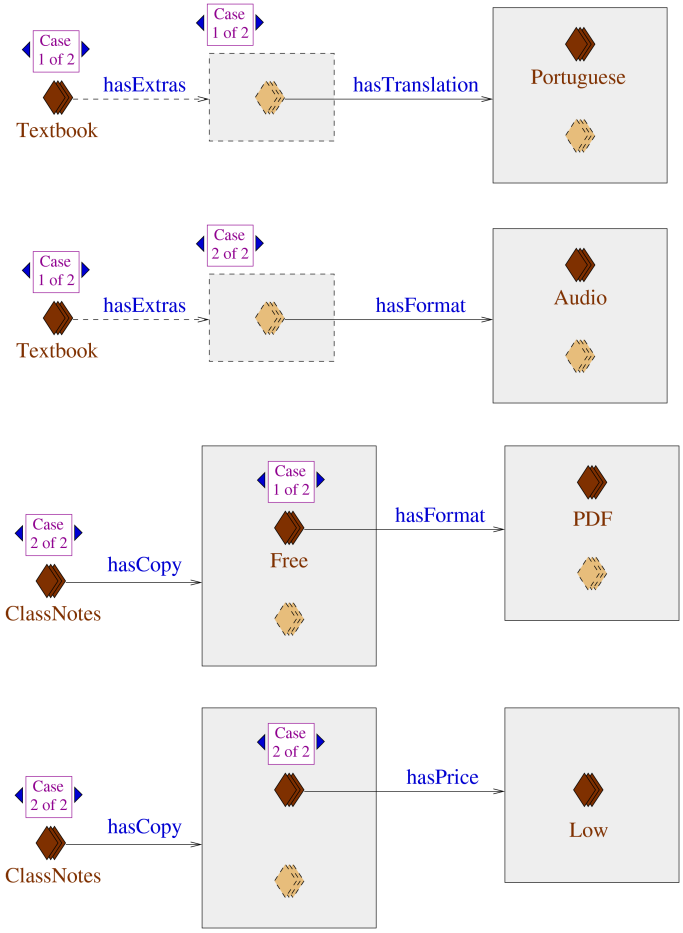

Fig. 5. Example model outline with case widgets

More formally, the model outline in Figure 5 corresponds to the description

$$
\begin{aligned}
&[\mathit{Textbook} \sqcap \forall \mathit{hasExtras}.\\
&\quad (\exists \mathit{hasTranslation}.\mathit{Portuguese} \sqcup\\
&\qquad \exists \mathit{hasFormat}.\mathit{Audio})] \sqcup\\
&\{\mathit{ClassNotes} \sqcap \exists \mathit{hasCopy}.\\
&\quad [(\mathit{Free} \sqcap \exists \mathit{hasFormat}.\mathit{PDF}) \sqcup\\
&\qquad (\exists \mathit{hasPrice}.\mathit{Low} \sqcap \forall \mathit{hasPrice}.\mathit{Low})]\}
\end{aligned}
$$

Note that there are textual elements in the concrete syntax of model outlines: labels associated to arrows, clusters and boxes. These textual elements, however, are kept simple so as not to compete with the diagrammatic elements for the user's attention.

An arrow label consists solely of a role name. Cluster and box labels are either conjunctions or disjunctions of literals (box labels are prefixed by the word "ALL" so as to emphasize that the label serves as a universal restriction). Cardinality restrictions are of one of three forms: "EXACTLY n", "n OR MORE", "FROM m THRU n".

We believe that these textual expressions are simple enough to be understood by a nonspecialist user; in particular, as cluster and box labels never mix disjunction and conjunction, the logical complexity of labels is kept in check. Furthermore, we see no substantial gain in trying to present the information in these labels in diagrammatic form (e.g., as abstract syntax trees). As we asserted in [6], a previous paper by us comparing textual and diagrammatic representations of concepts, "logocentric [i.e., textual] and diagrammatic approaches to the representation of concepts are not mu-

$$
\begin{aligned}
\langle \textit{outline} \rangle &\rightarrow \langle \textit{solidClstrCases} \rangle \\
\langle \textit{solidClstrCases} \rangle &\rightarrow \textbf{(}\ \textbf{cases}\ \langle \textit{solidCluster} \rangle^{+}\ \textbf{)} \\
\langle \textit{solidCluster} \rangle &\rightarrow \textbf{(}\ \textbf{cluster solid}\ \langle \textit{classLabel} \rangle\ \textbf{(}\ \langle \textit{arrow} \rangle^{\star}\ \textbf{)}\ \textbf{)} \\
\langle \textit{classLabel} \rangle &\rightarrow \textbf{(}\ \textbf{)} \mid \textbf{(}\ \langle \textit{literal} \rangle\ \textbf{)} \\
&\quad \mid \textbf{(}\ \textbf{and}\ \langle \textit{literal} \rangle\ \langle \textit{literal} \rangle^{+}\ \textbf{)} \mid \textbf{(}\ \textbf{or}\ \langle \textit{literal} \rangle\ \langle \textit{literal} \rangle^{+}\ \textbf{)} \\
\langle \textit{literal} \rangle &\rightarrow \langle \textit{conceptName} \rangle \mid \textbf{(}\ \textbf{not}\ \langle \textit{conceptName} \rangle\ \textbf{)} \\
\langle \textit{arrow} \rangle &\rightarrow \textbf{(}\ \textbf{arrow solid}\ \langle \textit{roleName} \rangle\ \textbf{(}\ \langle \textit{intrvl} \rangle^{\star}\ \textbf{)}\ \langle \textit{solidBoxCases} \rangle\ \textbf{)} \\
&\quad \mid \textbf{(}\ \textbf{arrow dashed}\ \langle \textit{roleName} \rangle\ \textbf{(}\ \langle \textit{intrvl} \rangle^{\star}\ \textbf{)}\ \langle \textit{dashedBoxCases} \rangle\ \textbf{)} \\
\langle \textit{intrvl} \rangle &\rightarrow \textbf{(}\ \langle \textit{number} \rangle\ \langle \textit{number} \rangle\ \textbf{)} \mid \textbf{(}\ \langle \textit{number} \rangle\ \textbf{infty}\ \textbf{)} \\
\langle \textit{solidBoxCases} \rangle &\rightarrow \textbf{(}\ \textbf{cases}\ \langle \textit{solidBox} \rangle^{+}\ \textbf{)} \\
\langle \textit{solidBox} \rangle &\rightarrow \textbf{(}\ \textbf{box}\ \langle \textit{classLabel} \rangle\ \textbf{(}\ \langle \textit{solidClstrCases} \rangle^{+}\ \textbf{)}\ \langle \textit{opt} \rangle\ \textbf{(}\ \langle \textit{arrow} \rangle^{\star}\ \textbf{)}\ \textbf{)} \\
\langle \textit{opt} \rangle &\rightarrow \textbf{(}\ \langle \textit{unlabeledCluster} \rangle\ \textbf{)} \mid \textbf{(}\ \langle \textit{innerBox} \rangle ?\ \textbf{(}\ \langle \textit{dashedClstrCases} \rangle^{\star}\ \textbf{)}\ \textbf{)} \\
\langle \textit{unlabeledCluster} \rangle &\rightarrow \textbf{(}\ \textbf{cluster dashed (}\ \textbf{)(}\ \textbf{)}\ \textbf{)} \\
\langle \textit{innerBox} \rangle &\rightarrow \textbf{(}\ \textbf{innerBox}\ \langle \textit{solidClstrCases} \rangle^{+}\ \textbf{)} \\
\langle \textit{dashedBoxCases} \rangle &\rightarrow \textbf{(}\ \textbf{cases}\ \langle \textit{dashedBox} \rangle^{+}\ \textbf{)} \\
\langle \textit{dashedBox} \rangle &\rightarrow \textbf{(}\ \textbf{box}\ \langle \textit{classLabel} \rangle\ \textbf{(}\ \langle \textit{snglDashedCluster} \rangle\ \textbf{)(}\ \textbf{)(}\ \langle \textit{arrow} \rangle^{\star}\ \textbf{)}\ \textbf{)} \\
\langle \textit{snglDashedCluster} \rangle &\rightarrow \textbf{(}\ \textbf{cases (}\ \textbf{cluster dashed}\ \langle \textit{classLabel} \rangle\ \textbf{(}\ \langle \textit{arrow} \rangle^{\star}\ \textbf{)}\ \textbf{)}\ \textbf{)} \\
\langle \textit{dashedClstrCases} \rangle &\rightarrow \textbf{(}\ \textbf{cases}\ \langle \textit{dashedCluster} \rangle^{+}\ \textbf{)} \\
\langle \textit{dashedCluster} \rangle &\rightarrow \textbf{(}\ \textbf{cluster dashed}\ \langle \textit{classLabel} \rangle\ \textbf{(}\ \langle \textit{arrow} \rangle^{\star}\ \textbf{)}\ \textbf{)}
\end{aligned}
$$

Fig. 6. Abstract, formal syntax for $\mathcal{ALCN}$ model outlines

tually exclusive; in fact, since human beings are fluent in both modes of communication (verbal and pictorial), we believe these modes can complement each other to great benefit."

If model outlines are used to help *construct* concept descriptions, a software tool can provide guidance to the user, ensuring that labels are filled out according to the syntactic restrictions explained above. For example, a graphical interface can present the user with a list of atomic class names and options to negate chosen names of the list and to join them in a conjunction or in a disjunction.

The *abstract syntax* of a model outline is formally defined as a LISP-style list generated by the grammar in Figure 6, which is in extended BNF notation. The abstract syntax of a model outline is intended to provide a structural representation of the components of the diagram, without regard for graphical details such as position, color, size or orientation. As an example, Figure 7 contains the list representation of the model outline in Figure 5. Note that the abstract syntax is not meant for human consumption, but rather for automatic processing by algorithms such as the ones presented in the next section.

3. Semantics of model outlines

The appearance of the components of a model outline follows some (hopefully intuitive) graphical conventions:

Individuals are represented by clusters of diamonds. The presence of a cluster (as opposed to a single diamond) emphasizes the idea that one *or more* individuals may appear in a given situation. E.g., in Figure 4, the graduate courses in question may have as lecturers more than one tenured department professor holding a CompSci or Math PhD degree and supervising at least one graduate student from a total of 2 or more individuals; the presence of the dashed cluster in the target box of the arrow labeled "supervises" indicates that the professors may also supervise individuals that are not graduate students (e.g., undergraduates).

Clusters of *solid* diamonds represent individuals that must exist. In Figure 4, it is mandatory that the graduate courses in question have as lecturer at least one tenured department professor holding a CompSci or Math PhD degree and supervising at least one graduate student from a total of 2 or more individuals. Likewise, the attendance must include students and graduate students.

Clusters of *dashed* diamonds represent optional individuals. If the cluster is labeled or has outgoing arrows, the individuals must belong to the corresponding

```
(cases (cluster solid (Textbook)
        ((arrow dashed (hasExtras) ()
          (cases (box () (cases (cluster dashed ()())) ()
                  (arrow solid (hasTranslation) ()
                   (cases (box () (cases (cluster solid (Portuguese) ()))
                           (cluster dashed ()()) ())))))
                 (box () (cases (cluster dashed ()())) ()
                  (arrow solid (hasFormat) ()
                   (cases (box () (cases (cluster solid (Audio) ()))
                           (cluster dashed ()()) ()))))))))
       (cluster solid (ClassNotes)
        ((arrow solid (hasCopy) ()
          (cases (box ()
                  (cases (cluster solid (Free)
                          (arrow solid (hasFormat) ()
                           (cases (box () (cases (cluster solid (PDF) ()))
                                   (cluster dashed ()()) ())))))
                         (cluster solid ()
                          (arrow solid (hasPrice) ()
                           (cases (box () (cases (cluster solid (Low) ()))
                                   ()()))))))
                 (cluster dashed ()()) ()))))))
```

Fig. 7. Abstract syntax for model outline in Fig. 5

class (e.g., "*Guest*" in Figure 4). If the cluster is unlabeled, the individuals may belong to any class, subject to the constraints stipulated by the label and the outgoing arrows of the outer box where the cluster is located (e.g., in Figure 4, the unlabeled cluster in the "*hasLecturer*" box represents lecturers that do not have to be tenured department professors, but that must hold a CompSci or Math PhD degree).

Arrows represent relationships. Arrows may originate from clusters or from boxes. Given a role name R, each cluster or box may have at most one outgoing arrow labeled by R.

As previously indicated, *box labels and arrows originating from boxes* represent constraints that must be satisfied by all individuals corresponding to clusters in the box. E.g., in Figure 4, all individuals attending the graduate courses in question must belong to class "*Enrolled*".

The *absence of a dashed cluster* in a box means that all the individuals represented in the box must belong to the classes specified by their respective labels and to the class specified by the box label and arrows (if present). This is evident in Figure 4, where it is required that the lecturers hold a PhD degree *only* in CompSci or Math (a rather exclusivist and unfair requirement, but this is only an example).

Dashed boxes, always the target of dashed arrows, always contain a dashed cluster, representing optional individuals. In Figure 4, the graduate courses in question may or may not involve the use of (up to 2) department labs.

"Among which" inner boxes contain clusters representing individuals that belong to subclasses of one or more classes specified in the outer box. In Figure 4, the attendance of the graduate courses in question consists of students, some of which are required to be graduate students. Optionally, guests may attend.

The above remarks are included here only for pedagogical purposes. In fact, we define the precise semantics of model outlines by means of the DESCR procedure, which, when given a model outline C (in abstract syntax), yields the $\mathcal{ALCN}$ concept description taken as the meaning of C. The DESCR procedure calls BOXDESCR to build the concept description denoted by a box. Algorithm 1 shows both procedures in pseudocode.

The reader should refer to the grammar in Figure 6 for the structure of the lists that the algorithms manipulate. These algorithms can be modified to produce more legible output; here, their only purpose is to serve as the precise semantics of model outlines. When given as input the model outline in Figure 2, e.g., algo-

```
DESCR(C)        ▷ C has the form ( cases C_1 ··· C_m )
1  Descr ← ⊥
2  for each C_i in C_1, ..., C_m        ▷ C_i has the form ( cluster S L ( A_1 ··· A_n ) )
3       do if L = ( )
4            then Case ← ⊤
5            else if L = ( NOTHING ) then Case ← ⊥ else Case ← L
6          for each A_j in A_1, ..., A_n
7               do Case ← Case ⊓ BOXDESCR(A_j)
8          Descr ← Descr ⊔ Case
9  return Descr

BOXDESCR(A)        ▷ A has the form ( arrow S RN ( I_1 ··· I_n ) ( cases B_1 ··· B_m ) )
1  BDescr ← ⊥
2  if n = 0        ▷ No cardinality restrictions
3     then Card ← ⊤
4     else Card ← ⊥
5          for each I_j in I_1, ..., I_n        ▷ I_j is "interval" of the form ( X Y )
6               do if Y = infty
7                    then Card ← (Card ⊔ ≥ X.RN)
8                    else Card ← (Card ⊔ (≥ X.RN ⊓ ≤ Y.RN))
9  for each B_i in B_1 ··· B_m        ▷ Each box case B_i has the form ( box BL ( C_1 ··· C_p ) Opt ( A'_1 ··· A'_q ) )
10       do Universal ← ⊥; Existentials ← ⊤
11          for each C_j in C_1, ..., C_p        ▷ Solid cluster cases
12               do Universal ← Universal ⊔ DESCR(C_j)
13                  if C_j has the form ( cluster solid ... )
14                    then Existentials ← Existentials ⊓ ∃RN .DESCR(C_j)
15          if Opt contains ( innerBox C'_1 ··· C'_r )
16            then for each C'_j in C'_1, ..., C'_r
17                      do Existentials ← Existentials ⊓ ∃RN .DESCR(C'_j)
18          if Opt contains ( cluster dashed ( ) ( ) )
19            then Universal ← ⊤
20          if Opt contains ( cases C''_1 ··· C''_s )        ▷ Optional clusters
21            then for each C''_j in C''_1, ..., C''_s
22                      do Universal ← Universal ⊔ DESCR(C''_j)
23          Universal ← ∀RN.(Universal)
24          if BL = ( )
25            then BCase ← Universal ⊓ Existentials
26            else BCase ← ∀RN.BL ⊓ Universal ⊓ Existentials
27          for each A'_j in A'_1, ..., A'_q        ▷ Box arrows
28               do BCase ← BCase ⊓ ∀RN.BOXDESCR(A'_j)
29          BDescr ← BDescr ⊔ BCase
30 return Card ⊓ BDescr
```

Algorithm 1. Conversion from model outlines to $\mathcal{ALCN}$

rithm DESCR returns the following description, which is equivalent to (1):

$$\begin{aligned}\bot \sqcup \{\top \sqcap \forall \textit{hasChild}.(\bot \sqcup \bot \sqcup \top) \sqcap \top \sqcap \\ \exists \textit{hasChild}.(\bot \sqcup \top) \sqcap \forall \textit{hasChild}.[\top \sqcap \\ (\bot \sqcup \forall \textit{hasChild}.(\bot \sqcup (\textit{Doctor} \sqcap \neg \textit{Lawyer})))]\}\end{aligned}$$

The remarks below will help the reader follow the code of DESCR and BOXDESCR in Algorithm 1:

– When DESCR(C) is called, C is a list containing a collection of cluster cases. Each cluster case C_i is processed by the loop in lines 2–8. In line 9, the disjunction of the descriptions corresponding to the cluster cases is returned.

– In DESCR, each cluster case C_i has its label processed (lines 3–5) and its collection of arrows sent to BOXDESCR (line 7). The conjunction of the descriptions corresponding to those label and arrows is stored in variable $Case$.
– Procedure BOXDESCR is responsible for processing the arrow represented by the argument A and all the box cases that the arrow points to.
– In BOXDESCR, lines 2–8 handle the cardinality restrictions (if any) associated to the arrow.
– In BOXDESCR, the loop in lines 9–29 handles each box case B_i that the arrow points to.
– In the loop, variable $Universal$ will store a description of the form $\forall R.D$. At first, D will be a disjunction of all the descriptions corresponding to the solid cluster cases in the box (line 12).
– However, if there is an unlabeled dashed cluster in the box, then there is no universal restriction corresponding to the box (based on the clusters in the box, that is), and $Universal$ gets $\top$ (line 19).
– If there are labeled dashed clusters in the box (which precludes the unlabeled dashed cluster mentioned in the previous remark — see grammar in Figure 6), the descriptions corresponding to such clusters will also be included in the disjunction stored in $Universal$ (line 22).
– In the loop, variable $Existentials$ will hold a conjunction of descriptions, each one of the form $\exists R.D$. These concept descriptions originate from the solid clusters in the box (line 14) and from the solid clusters in the "among which" inner box (line 17).
– Finally, BOXDESCR will process the box label (lines 24–26), and the box arrows (lines 27–28).

4. Constructing model outlines

In [8] we presented a first algorithm for translating $\mathcal{ALCN}$ concept descriptions into model outlines. In this paper, we incorporate some important changes to the algorithm (e.g., to account for labeled optional clusters) and give a more informal explanation of the main steps involved in such a translation, using as a working example the concept description that originated the model outline in Figure 4.

Given an $\mathcal{ALCN}$ concept description C, we start by converting C to modified disjunctive normal form (mDNF), applying simplification rules in the process. A concept description is in mDNF if it fits the pattern

$$D_1 \sqcup \ldots \sqcup D_n$$

where each disjunct D_i is a conjunction of the form

$$C_1 \sqcap \cdots \sqcap C_p$$

where each conjunct C_j is either a literal, or a collection of "intervals" of natural numbers (whose upper bound may be ∞) associated to a role R, or a description of the form $\forall R.C'$ or of the form $\exists R.C'$, where C' is itself in mDNF.

The modification is in the way number restrictions are represented: using appropriate rewrite rules, any conjunction of cardinality restrictions over a role R_i can be converted to a collection of "intervals" of natural numbers; for role R, the interval $[m, n]$ represents the constraint $(\geq m.R \ \sqcap \ \leq n.R)$. Likewise, $[m, m]$ represents $(= m.R)$, and $[0, m]$ represents $(\leq m.R)$, and $[m, \infty]$ represents $(\geq m.R)$.

To each D_i we then apply the simplification rule

$$\forall R.C_1 \sqcap \ldots \sqcap \forall R.C_n \ \rhd \ \forall R.(C_1 \sqcap \ldots \sqcap C_n)$$

As a result, we obtain C', which is a disjunction $D'_1 \sqcup \ldots \sqcup D'_n$, where each D'_i can be written as

$$L_1 \sqcap \ldots \sqcap L_m \sqcap C_1 \sqcap \cdots \sqcap C_p$$

where each L_i is a literal, and each C_j can be written as

$$\forall R.F \sqcap \exists R.G_1 \sqcap \ldots \sqcap \exists R.G_q \sqcap K$$

where F and all the G_i are in mDNF and K is a collection of intervals of natural numbers representing cardinality restrictions over role R. Any (or all) of these elements may be absent. Note that we have grouped the conjuncts according to the role R they refer to. Later, when the model outline is built, each of these groups will originate an arrow labeled by R (this is why at most one arrow labeled by R may originate from any given cluster or any given box).

Following these guidelines, the simplified mDNF of the concept description corresponding to the example

model outline in Figure 4 is found to be

$$
\begin{aligned}
&\textit{GraduateCourse} && \text{(2a)}\\
\sqcap\, &\forall \textit{hasLecturer}. && \text{(2b)}\\
&\quad [\forall \textit{holdsPhDIn}.(\textit{CompSci} \sqcup \textit{Math}) && \text{(2c)}\\
&\quad \sqcap\, \exists \textit{holdsPhDIn}.(\textit{CompSci} \sqcup \textit{Math})] && \text{(2d)}\\
\sqcap\, &\exists \textit{hasLecturer}.(\textit{DeptProfessor} \sqcap \textit{Tenured} && \text{(2e)}\\
&\quad \sqcap\, \exists \textit{supervises}.\textit{GradStudent} && \text{(2f)}\\
&\quad \sqcap\, \{[2,\infty]\}.\textit{supervises}) && \text{(2g)}\\
\sqcap\, &\{[2,2]\}.\textit{hasLecturer} && \text{(2h)}\\
\sqcap\, &\forall \textit{hasAttendance}.[(\textit{Student} \sqcap \textit{Enrolled}) && \text{(2i)}\\
&\quad \sqcup\, (\textit{Guest} \sqcap \textit{Enrolled})] && \text{(2j)}\\
\sqcap\, &\exists \textit{hasAttendance}.\textit{Student} && \text{(2k)}\\
\sqcap\, &\exists \textit{hasAttendance}.\textit{GradStudent} && \text{(2l)}\\
\sqcap\, &\{[10,50]\}.\textit{hasAttendance} && \text{(2m)}\\
\sqcap\, &\forall \textit{usesLab}.(\textit{DeptLab} \sqcap \neg \textit{Closed}) && \text{(2n)}\\
\sqcap\, &\{[0,2]\}.\textit{usesLab} && \text{(2o)}
\end{aligned}
$$

Note how the constraints have been grouped by the roles they act upon. Note also how the cardinality constraints in lines (2g), (2h), (2m) and (2o) have been written with (singleton) collections of intervals of natural numbers.

Two transformations must be effected before the model outline can be built.

The first one concerns lines (2c)–(2d), where the set of objects related to the lecturers through *holdsPhDIn* is *closed*: i.e., the lecturers must hold *some* PhD degree in CompSci or Math and *only* PhD degrees in CompSci or Math.

The algorithm detects such a closure whenever it finds conjuncts of the form

$$\forall R(C_1 \sqcup \cdots \sqcup C_n) \sqcap \exists R.C_1 \sqcap \cdots \sqcap \exists R.C_n$$

Here, we have $n = 1$ and $C_1 = \textit{CompSci} \sqcup \textit{Math}$. Then, to indicate the closure, the algorithm refrains from adding a dashed, unlabeled cluster to the target box of the *holdsPhDIn* arrow (see Figure 4).

The second transformation is similar: in lines (2i)–(2l), we can see there is some sort of closure related to the role *hasAttendance*, but the situation is more complicated. In fact, this is the general case, which also includes the first transformation. Whenever the conjuncts for role R are of the form

$$
\begin{aligned}
&\forall R[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D)\\
&\qquad \sqcup\, (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)]\\
&\sqcap\, \exists R.C_1 \sqcap \cdots \sqcap \exists R.C_n \sqcap \exists R.F_1 \sqcap \cdots \sqcap \exists R.F_q
\end{aligned}
$$

where D is a conjunction (with $D = \top$ as the trivial case) it proceeds as follows:

- Solid clusters for $C_1, \ldots, C_n$ are created in the main target box for the R-arrow.
- If D is a conjunction of literals, then the main target box for the R-arrow gets D as a label. If $D = \top$, this label is not shown.
- If D is not a conjunction of literals (i.e., if one of the conjuncts is not a literal), then the main target box for the R-arrow gets the literal conjuncts as a label and the nonliteral conjuncts as arrows originating from the box.
- The main target box for the R-arrow gets an "among which" inner box containing solid clusters for $F_1, \ldots, F_q$.
- Dashed clusters for $C_{n+1}, \ldots, C_p$ are created in the main target box for the R-arrow.

In our example description, in lines (2i)–(2l), we have that $n = 1$, and $C_1 = \textit{Student}$, and $D = \textit{Enrolled}$, and $p = 1$, and $C_2 = \textit{Guest}$, and $q = 1$, and $F_1 = \textit{GradStudent}$.

Algorithms 2–4 show the detailed pseudocode for converting an $\mathcal{ALCN}$ concept description C in modified DNF to the abstract syntax of its corresponding model outline. Next, we present remarks about the behavior of each procedure in the algorithms.

In order to build the entire model outline for an $\mathcal{ALCN}$ concept description C, one should call the procedure as BUILDCLUSTERCASES(**solid**, C).

The remarks below will help the reader follow the code of BUILDCLUSTERCASES in Algorithm 2:

- Lines 1–8 treat the base cases, where the description in modified DNF is $\top$, $\bot$, a disjunction of literals, or a conjunction of literals, respectively.
- In line 2, this piece of abstract syntax should be rendered visually in such a way as to make it clear that C is inconsistent (i.e., C denotes the empty set).
- If the algorithm does not exit before or at line 8, then we know C is of the form $L_1 \sqcup \cdots \sqcup L_m \sqcup D_1 \sqcup \cdots \sqcup D_n$, with $m \geq 0$, $n \geq 1$, where each L_i is a literal, and each D_i is a nonliteral.

BUILDCLUSTERCASES($stroke, C$)

```
1  if C = ⊥
2     then return ( cases ( cluster stroke ( NOTHING ) ( ) ) )
3  if C = ⊤
4     then return ( cases ( cluster stroke ( ) ( ) ) )
5  if C = L_1 ⊔ ··· ⊔ L_n        ▷ A disjunction of literals
6     then return ( cases ( cluster stroke ( OR L_1 ··· L_n ) ( ) ) )
7  if C = L_1 ⊓ ··· ⊓ L_n        ▷ A conjunction of literals
8     then return ( cases ( cluster stroke ( AND L_1 ··· L_n ) ( ) ) )
9  ▷ From here on, C is of the form L_1 ⊔ ··· ⊔ L_m ⊔ D_1 ⊔ ··· ⊔ D_n, with m ≥ 0, n ≥ 1,
10 ▷ all of the L_i literals, and all of the D_i nonliterals
11 cases ← ( cluster stroke ( OR L_1 ··· L_m ) ( ) )
12 for each D_i in D_1 ··· D_n
13 ▷ D_i is of the form L_1 ⊓ ··· ⊓ L_m ⊓ C_1 ⊓ ··· ⊓ C_p, with m + p > 0
14    do classLabel ← ( AND L_1 ··· L_m )
15       arrows ← ( )
16       for each C_j in C_1 ··· C_p
17          do arrows ← arrows + BUILDARROW(C_j)
18       cases ← cases + ( cluster stroke classLabel arrows )
19 return ( cases cases )
```

Algorithm 2. Conversion from $\mathcal{ALCN}$ to model outlines: BUILDCLUSTERCASES

– Line 11 gathers all the literals in $L_1 \sqcup \cdots \sqcup L_m$ into one single case.
– The loop in lines 12–18 produces a case rooted in the present cluster for each of the remaining disjuncts D_i, each of which is of the form

$$L_1 \sqcap \cdots \sqcap L_m \sqcap C_1 \sqcap \cdots \sqcap C_p$$

subject to all of the following conditions:

  * $m + p > 0$ (i.e., there must be at least one conjunct; otherwise, C would be $\top$, and would have been treated in one of the base cases of the algorithm).
  * If $p = 0$ then $m \geq 2$ (i.e., if there are only literals in the conjunction, there must be more than one literal; otherwise, D_i would be a literal, and would have been treated in line 11 above, which gathers all literals into a single case).

– In line 14, if $m = 0$, then *classLabel* should be empty. If $m = 1$ then, it should get L_1, with no **AND**.
– Now, each C_j consists of a conjunction of all restrictions over a single role R, including cardinality restrictions.
– Then, in the loop consisting of lines 16–17, each C_j gives rise to an arrow leaving the present case, and in line 18 the present case is added to the set of cases associated to the present cluster.

Each arrow (along with its target box or box cases) is built by procedure BUILDARROW, whose pseudocode is detailed in Algorithm 3.

The following remarks are intended to help the reader follow the steps involved:

– Strictly speaking, the BUILDARROW procedure is non-deterministic, as there may be more than one way to parse the input concept description C in the form outlined in lines 1– 6.
– More precisely, there may be more than one conjunct D common to all disjuncts in the $\forall R$ universal restriction.
– Concept descriptions $C_1 \ldots C_n$ will correspond to solid clusters in the target box.
– Concept descriptions $C_{n+1} \ldots C_{n+p}$ will correspond to dashed clusters (i.e., optional individuals) in the target box.
– Concept description D will correspond to the target box's label and to arrows leaving the target box.
– Concept descriptions $F_1 \ldots F_q$ will correspond either to solid clusters in the "among which" inner box or to solid clusters in the target box, depending on circumstances explained below.
– The procedure is structured in 3 cases:

BUILDARROW(C)

1 ▷ C is of the form
2 ▷ $\forall R.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D) \sqcup (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)]$
3 ▷ $\sqcap \exists R.C_1 \sqcap \cdots \sqcap \exists R.C_n$
4 ▷ $\sqcap \exists R.F_1 \sqcap \cdots \sqcap \exists R.F_q$
5 ▷ $\sqcap K(R)$
6 ▷ with $K(R)$ a (possibly empty) list of natural number intervals representing cardinality restrictions over R
7 **if** $n = 0 \wedge p = 0$
8 **then if** $q > 0$
9 **then** $opt \leftarrow$ **(cluster dashed ( ) ( ))**
10 $existentials \leftarrow$ **(** $F_1 \ldots F_q$ **)**; $stroke \leftarrow$ **solid**
11 **else if** $K(R)$ has an interval containing 0
12 **then** $opt \leftarrow$ **(cluster dashed ( ) ( ))**
13 $existentials \leftarrow$ **( )**; $stroke \leftarrow$ **dashed**
14 **else** $opt \leftarrow$ **( )**; $existentials \leftarrow$ **(** $\top$ **)**; $stroke \leftarrow$ **solid**
15 $boxCase \leftarrow$ BUILDBOXCASE($\top$, $existentials$, opt)
16 **return (arrow** $stroke$ **(** R **) (** $K(R)$ **) (cases** $boxCase$ **))**
17 **if** $n > 0$
18 **then** $existentials \leftarrow$ **(** $C_1 \ldots C_n$ **)**
19 **if** $q = 0$
20 **then** $opt \leftarrow$ **( )**
21 **else** $opt \leftarrow$ **(innerBox** BUILDCLUSTERCASE(**solid**, F_1) ... BUILDCLUSTERCASE(**solid**, F_q) **)**
22 **if** $p > 0$
23 **then** $opt \leftarrow opt +$ **(cases** BUILDCLUSTERCASE(**dashed**, C_{n+1}) ... BUILDCLUSTERCASE(**dashed**, C_{n+p}) **)**
24 $boxCase \leftarrow$ BUILDBOXCASE(D, $existentials$, opt)
25 **return (arrow solid (** R **) (** $K(R)$ **) (cases** $boxCase$ **))**
26 **if** $n = 0 \wedge p > 0$
27 **then** $existentials \leftarrow$ **( )**; $opt \leftarrow$ **( )**; $literals \leftarrow$ **( )**; $disjuncts \leftarrow$ **(** $C_{n+1} \ldots C_{n+p}$ **)**
28 **for** each C_i in $C_{n+1} \ldots C_{n+p}$
29 **do if** C_i is a literal
30 **then** $literals \leftarrow literals \sqcup C_i$; $disjuncts \leftarrow disjuncts - C_i$
31 $boxCases \leftarrow$ **(cases)**
32 **if** $q = 0$
33 **then if** $literals$ is nonempty
34 **then** $opt \leftarrow$ BUILDCLUSTERCASE(**dashed**, $literals$)
35 $boxCases \leftarrow boxCases +$BUILDBOXCASE($D$, $existentials$, opt)
36 **for** each C in $disjuncts$
37 **do** $opt \leftarrow$ BUILDCLUSTERCASE(**dashed**, C)
38 $boxCases \leftarrow boxCases +$BUILDBOXCASE($D$, $existentials$, opt)
39 **if** $K(R)$ is absent or has an interval containing 0
40 **then return (arrow dashed (** R **) (** $K(R)$ **) (** $boxCases$ **))**
41 **else return (arrow solid (** R **) (** $K(R)$ **) (** $boxCases$ **))**
42 **else** $existentials \leftarrow F_1 \ldots F_q$
43 **if** $literals$ is nonempty
44 **then** $opt \leftarrow$ BUILDCLUSTERCASE(**dashed**, $literals$)
45 $boxCases \leftarrow boxCases +$ BUILDBOXCASE(D, $existentials$, opt)
46 **for** each C in disjuncts
47 **do** $opt \leftarrow$ BUILDCLUSTERCASE(**dashed**, C)
48 $boxCases \leftarrow boxCases +$ BUILDBOXCASE(D, $existentials$, opt)
49 **return (arrow solid (** R **) (** $K(R)$ **) (** $boxCases$ **))**

Algorithm 3. Conversion from $\mathcal{ALCN}$ to model outlines: BUILDARROW

BUILDBOXCASE(D, E, opt)

```
1  ▷ D is of the form L_1 ⊓ ··· ⊓ L_n ⊓ C_1 ⊓ ··· ⊓ C_m, with each C_i over a role name
2  boxLabel ← ( AND L_1 ··· L_n )
3  boxArrows ← ( )
4  for each C_i in C_1 ··· C_m
5      do boxArrows ← boxArrows + BUILDARROW(C_i)
6  allClusterCases ← ( )
7  for each element G in E
8      do clusterCases ← ( cases ) + BUILDCLUSTERCASES(solid, G)
9         allClusterCases ← allClusterCases + clusterCases
10 return ( box ( boxLabel ) allClusterCases opt boxArrows )
```

Algorithm 4. Conversion from $\mathcal{ALCN}$ to model outlines: BUILDBOXCASE

1. $n = 0 \wedge p = 0$ (starting on line 7): there is only one target box case. If present, concept descriptions $F_1 \ldots F_q$ give rise to solid clusters. As there are no universal restrictions, the target box will contain a dashed, unlabeled cluster, unless the cardinality restrictions K force the existence of individuals in the target box, in which case, the unlabeled cluster will be solid.
2. $n > 0$ (starting on line 17): there is only one target box case. Concept descriptions $C_1 \ldots C_n$ will give rise to solid clusters. If present, descriptions $C_{n+1} \ldots C_{n+p}$ will give rise to dashed, labeled clusters corresponding to optional individuals. If present, concept descriptions $F_1 \ldots F_q$ will give rise to solid clusters in the "among which" inner box. D will give rise to the target box's label and outgoing arrows.
3. $n = 0 \wedge p > 0$ (starting on line 26): multiple box cases may be produced here. Each of $C_{n+1} \ldots C_{n+p}$ that is not a literal will give rise to a box case (literals will be gathered into one single box case and appear as a disjunction in the dashed cluster's label). If present, concept descriptions $F_1 \ldots F_q$ will give rise to solid clusters in the "among which" inner box, which will be common to all box cases. D will participate in the target box's label and outgoing arrows.

Finally, procedure BUILDBOXCASE in Algorithm 4 is responsible for building each box case. Concept description D gives rise to the box's label and outgoing arrows. The contents of the box case are generated from parameters E (solid clusters) and opt (dashed clusters and/or an "among which" inner box). Note that the stroke of the box (dashed or solid) and the cardinality restrictions are actually associated to the arrow incident to the box, which is computed by the BUILDARROW procedure.

5. Usability Evaluation

We have conducted a usability study in order to evaluate our proposed diagrammatic notation. The main aim was to *test the usefulness of model outlines for the understanding of complex concept descriptions*.

Note that it is the model outline *notation* itself that is being evaluated, not a specific graphical user interface (GUI) implementing the notation. Thus, the focus of the study is on understanding, not on interaction. We find this to be an advantage, as changes can be made to the notation before we are committed to a specific GUI, and problems can be identified in relation to specific features of the notation, so that special attention can be given to these problems in order to solve or mitigate them through the use of appropriate human interaction techniques. From a practical point of view, this potentially reduces the need for radical, costly changes after implementation.

Likewise, we have chosen model outlines for the simpler $\mathcal{ALCN}$ language so we could find out early if something needs to be changed in our most basic assumptions. The result of this test will help us design the extensions of model outlines to deal with more expressive concept languages.

Following [9], our *main goal* is to show that *model outlines can help users with little or no training in Logic to understand complex concept descriptions. In particular, model outlines are more effective than Manchester OWL for this task.*

Manchester OWL (see Figure 3 and also [12]) is a textual notation for DL which uses keywords for logical symbols (e.g., "**SOME**" for "$\exists$"), infix notation for restrictions (e.g., "*hasChild* **SOME** *Man*" for "$\exists$*hasChild*.*Man*"), syntax highlighting and indentation in order to make descriptions more readable for nonspecialists. So, we are comparing our diagrammatic notation with a textual notation designed for the same target audience. As the test participants were all Brazilians, we used Brazilian Portuguese translations of the Manchester OWL keywords, as listed below:

NOT	NÃO
AND	E
OR	OU
SOME	ALGUM
ONLY	SOMENTE
ONLYSOME	SOMENTEALGUM
MIN	MIN
MAX	MAX
EXACTLY	EXATAMENTE

The decision to use Brazilian Portuguese translations of Manchester OWL keywords may have introduced an extra variable in the usability study, as one must now worry about how faithful to the original keywords those translations are. However, educated Portuguese speakers will notice that the translations reflect the semantics of the original keywords, including the potential for undesired interpretations (i.e., those that do not agree with the formal semantics of DL operators).

For example, Brazilian Portuguese "OU" is as ambiguous as English "**OR**", as many speakers tend to interpret it as *exclusive or* instead of *inclusive or*.

Also, the mistake of interpreting universal restrictions as implying existence is as common with English "**ONLY**" as with Portuguese "SOMENTE".

As noted in [12], users "*still needed training to re-align their 'natural interpretation' with the correct OWL/DL interpretation. [...] However, it is arguable that explanations of OWL semantics would be required whatever syntax was chosen.*" In our usability study, participants took a tutorial on Manchester OWL (in Portuguese) containing many examples before they had to answer questions to test their understanding of concept descriptions.

The material used in the tutorials and the tests is available online at `http://www.professores.uff.br/fnaufel/mo/us01.html`.

Next, we defined a set of *concerns*, in the form of questions like: *Can users understand the meaning of X?*, where X is one of the elements present in model outlines (solid clusters, dashed clusters, arrows, boxes, inner boxes, case widgets, etc.). Specific concerns were also formulated (e.g., "Can users understand that individuals in "among which" inner boxes are mandatory?").

We selected 10 participants for our study (note that [9] recommends 6 to 12). These participants come from several backgrounds and occupations, as detailed below. All received detailed information on the procedures and on their rights as participants. All signed terms of informed consent.

One session of the study consisted of:

1. a pre-test questionnaire,
2. a tutorial on notation A,
3. a specification on domain X using notation A,
4. 15 questions about the specification,
5. a post-task questionnaire,
6. a tutorial on notation B,
7. a specification on domain Y using notation B,
8. 15 questions about the specification,
9. a post-task questionnaire,
10. a post-test questionnaire.

Notations A and B alternated between model outlines and Manchester OWL. Domains X and Y alternated between graduate courses (which included Figure 4 of this paper) and family relations. Each participant answered 15 questions for each domain. The questions for each domain were fixed, regardless of the notation used. For each domain, half the participants answered questions on model outlines, and half answered questions on Manchester OWL specifications. Half the participants saw model outlines before Manchester OWL, and half saw Manchester OWL before model outlines.

The number of correct answers and the time to answer were measured. Additional information was obtained in the form of comments collected through the "thinking out loud" protocol [9] and through questionnaires. Table 1 shows the occupation and the number of correct answers for each participant.

For the graduate courses domain, we note the following highlights:

Question 8 was related to Figure 4 of this paper, and elicited *5 errors* using Manchester OWL, and *no errors* using model outlines. The question was: "If a course is attended only by students that are *not* graduate stu-

Table 1
Occupation and number of correct answers for each participant

Occupation	Correct answers (model outlines)	Correct answers (Manchester OWL)
Logician	15	14
Theoretical physicist	15	12
Software engineer	15	12
Secretary	15	10
Nurse	13	12
Graphics designer	13	12
Social worker	13	11
Comp. Science undergrad	13	10
Production engineer	13	9
Mathematician	12	14
Totals:	137	116
Percentages:	91.3%	77.3%

dents, does the course meet the specification?" The error was probably induced by the abbreviation recommended in [12]:

$$\textit{hasAttendance}\ \textbf{SOME}\ [\textit{Student}, \textit{GradStudent}]$$

which seems to have evoked the idea that the bracketed list consisted of a set of alternatives. This question was answered correctly by all participants using model outlines, which indicates that users understood the meaning of "among which" inner boxes.

Question 14 elicited *4 errors* with Manchester OWL, and *3 errors* with model outlines. This question was about a specification consisting of 4 cases. The situation proposed in the question satisfied exactly one of the 4 cases. With Manchester OWL, the participants had difficulty in finding their way among multiple parentheses and complex disjunctions. With model outlines, they apparently thought that the proposed situation had to satisfy *all* cases.

For the family relations domain, we note the following highlights:

Question 6 elicited *4 errors* with Manchester OWL, and *no errors* with model outlines. This question asked if a person satisfying the given specification could have jobless children. The specification in Manchester OWL included the sentence

$$\textit{hasChild}\ \textbf{SOME}\ (\textit{Man}\ \textbf{AND}\ \textit{worksAt}\ \textbf{ONLY}\ \textit{Hospital})$$

Apparently, the users forgot the fact that "ONLY" (which stands for "$\forall$") does not imply the existence of objects. In the model outline, the presence of a dashed cluster, a dashed box and a dashed arrow made it clear that existence was not required.

Question 8 elicited *3 errors* using Manchester OWL, and *1 error* using model outlines. This question asked if a person satisfying the given specification had to have a grandchild working as a surgeon. Some users found it confusing to follow the composition of roles (*hasChild–hasChild*), and were again, as in question 8 about graduate courses, confused by the Manchester OWL abbreviation "**SOME** $[\cdots]$". In the model outline, the presence of a solid cluster labeled *Surgeon* inside an "among which" inner box made the correct answer more clear.

One trend was clearly observed in both domains: specifications that involve cases (i.e., complex disjunctions), such as the one in Figure 5 of this paper, are more difficult to understand than those that do not, as Table 2 indicates.

Table 2
Number of correct answers per domain and type of specification

Domain and type of specification	Correct answers (model outlines)	Correct answers (Manchester OWL)
Family, no cases	94%	72%
Courses, no cases	96%	84%
Family, with cases	84%	72%
Courses, with cases	84%	80%

Among the comments offered by the participants, many indicated confusion due to the way cases were presented in model outlines (like in Figure 5 of this paper, the layout used in the test consisted of 4 diagrams on a single page). Some users thought that all 4 diagrams had to be satisfied. This is clearly one weakness of model outlines (on paper) that we must try to eliminate in the GUI implementation. We predict that such confusion will not arise if the user interacts with the model outline (e.g., dynamically expanding and collapsing cases). The GUI should also make clear when clusters in different cases actually correspond to the same cluster, by showing one single cluster which can be expanded in different ways.

As for time: in the courses domain, each user took in average 28 seconds per question, regardless of the notation. In the family relations domain, each user took in average 26 seconds per question with model outlines, but 40 seconds per question with Manchester OWL.

Of the 10 participants, 5 said they preferred model outlines, 4 said they liked both notations equally well, and 1 said both notations were equally bad, remarking that he preferred formal logic notation.

6. Discussion

Among several questions related to our proposed visual framework, we examine here the issue of possible applications and a more specific problem regarding the interaction of concept descriptions with the axioms in the TBox of a formal ontology.

6.1. Applications

Apart from the usability issues discussed in Section 5, we can examine the question of *usefulness*; more precisely, in what applications and tasks can model outlines be of help to users?

The algorithms presented in Sections 3 and 4 provide a translation-based formal semantics to our visual language. Such algorithms also suggest two basic ways of using model outlines:

In a first scenario, an ontology browser can show nonspecialist users model outlines corresponding to complex concept descriptions (such as necessary and sufficient conditions in the definitions of classes). This was our original motivation, and this is the application the usability test was designed to take into account (investigating how well nonspecialist users could understand concept descriptions rendered as model outlines). Here, a concept description is translated into a model outline, and the user takes on a more or less passive stance, interacting with a finished diagram only to collapse and expand elements, switch the focus to certain cases, etc.

Of course, the usefulness of model outlines in this scenario depends on the frequency with which users have to deal with complex concept descriptions such as the one represented by the model outline in Figure 4. We conjecture that in the context of proof explanation, at least, complex concept descriptions arise naturally. The answer to the more general question — how often do complex concept descriptions occur in practice? — requires designing specific tests and surveys targeted at a broad population of ontology users, and is not examined further in this article.

In a second situation, the user may be expected to create and edit model outlines. We can envision a dedicated graphical editor which will guide the user in the task of assembling a diagram from basic elements such as clusters, boxes and arrows, providing autocompletion and other facilities to help the user find the desired class and role names to include in labels. In this scenario, the user takes on a much more active role, and the interaction provided by the graphical editor must be carefully planned in order to prevent the user from building nonsensical diagrams or, even worse, to prevent the user from becoming confused because a certain operation he is trying to perform (and which he finds to be intuitive) is not being allowed by the editor. In other words, the manual construction of model outlines by the user presents interaction and usability problems that seem to be harder than those involved in the mere exhibition of finished model outlines. If such a graphical editor is to be made available, more specific usability tests must be conducted first.

The manual construction of model outlines is an activity that may be required in at least two tasks: the user may want to define concept descriptions to serve as necessary and sufficient conditions in class definitions, or the user may want to construct a diagram to submit as a visual query on the ontology he is working on. In this latter case, the definition of model outlines must be altered to provide features that are typical of query languages, such as clusters marked as variables to be bound by the query results. One possibility that seems worth investigating is the formatting of the query's results themselves as model outlines, allowing the same visual language to be used for queries and for the data returned by them.

The use of model outlines to exhibit instance data leads to a third possible application: model exploration [3], where a user may interact with generated models of the formal ontology he is working on, so as to gain better understanding of the axioms in the ontology and their logical consequences and/or to test conjectures.

6.2. Model outlines and the TBox

Model outlines have been originally designed to denote "standalone" concept descriptions, which means that the atomic concepts and the role names that appear in the diagram are not constrained in any way other than what is shown in the diagram itself. However, in real life, concept descriptions usually appear in the context of a formal ontology, whose TBox axioms affect the interpretation of atomic concepts and role names.

A simple example is the fact that a solid cluster labeled "*Class1* **AND** *Class2*" will denote nonexistent individuals if the ontology's TBox contains the axiom

$$Class1 \sqsubseteq \neg\, Class2$$

which states that the atomic concepts *Class1* and *Class2* are disjoint.

A slightly more complex situation is illustrated by Figure 4, which seems to suggest that *GradStudent* is a subclass of *Student*, but this type of knowledge belongs in the TBox, and should not be conveyed by a model outline! To be exact, the "among which" inner box in Figure 4 is only saying that *if GradStudent and Student are disjoint, then the GraduateCourses at the root of the diagram can have no attendance*.

This is a general phenomenon related to "among which" inner boxes. A question that naturally arises, then, is whether these inner boxes are really necessary in the visual language.

Consider the model outline in Figure 8. It corresponds to the $\mathcal{ALCN}$ description

$$\forall R.(A \sqcup B) \sqcap \exists R.A \sqcap \exists R.B \sqcap \exists R.C$$

The modified model outline obtained by removing the inner box and placing the cluster labeled C in the main box, along with the clusters labeled A and B, shown in Figure 9, corresponds to the $\mathcal{ALCN}$ description

$$\forall R.(A \sqcup B \sqcup C) \sqcap \exists R.A \sqcap \exists R.B \sqcap \exists R.C$$

It can be shown that these two descriptions are equivalent if and only if the interpretation of C is contained in the interpretation of $A \sqcup B$, but this containment is not the kind of information we have designed model outlines to convey.

In the context of a model-outline-based tool used for ontology browsing, one solution would be to invoke a DL reasoner to check the consistency of each cluster label and each box, using the knowledge present in the TBox. Then the tool could somehow visually highlight the inconsistent elements of the diagram, alerting the user.

In the context of a model-outline-based visual query language, it seems that "among which" inner boxes can be eliminated without essentially compromising the range of queries that can be formulated. This is because the user could build a query corresponding to Figure 9 and be informed, in the results, of the possibility that individuals could appear at the same time as instances of $A \sqcup B$ and of C.

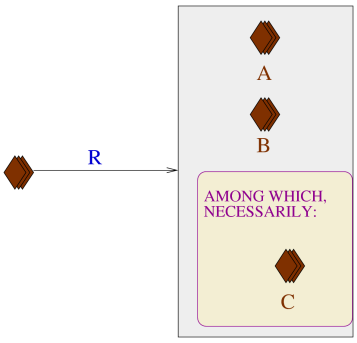

Fig. 8. A model outline with an inner box

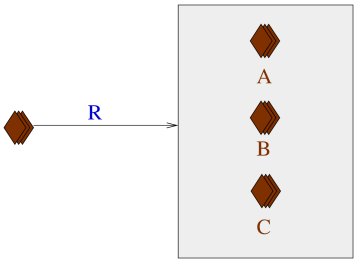

Fig. 9. A similar model outline, without the inner box

7. Conclusions

The main achievements of the work related here are the reformulation of our model outline visual language and its related algorithms, as well as the results of our first usability test, comparing model outlines to Manchester OWL.

Ontology visualization is a very active field of study. The survey [15] discusses over 40 ontology visualization tools, all of them developed in the past 10 years. All of those tools are *general*, in the sense that they use one single visualization framework to show several types of information about the ontology: the subsumption hierarchy, roles, etc. In particular, those tools show concept descriptions either textually (e.g., Protégé) or in the form of abstract syntax trees (as in Figure 1 of this paper).

Model outlines, on the other hand, are *specialized*, having been designed specifically to show concept descriptions. Although the notation used is new, our usability test indicates it is intuitive enough to be un-

derstood by nonspecialists. The specialized nature of model outlines suggests that they can be *integrated* with a more general tool, so that users can easily switch views, e.g., from the subsumption hierarchy as a tree to the definition of a class as a model outline.

To summarize, this article has provided a precise, updated presentation of our visual framework, with rigorous definitions of the syntax and the semantics of model outlines. Furthermore, the results of a carefully conducted usability test indicate that our framework is more useful in helping nonspecialist users understand complex concept descriptions than a text-based notation (Manchester OWL).

We are currently implementing a concept description browser based on model outlines, as a Protégé plugin. We are taking special care to rely on graphical conventions and interaction techniques that profit from the vast body of knowledge related to visual perception and cognitive principles, as described, e.g., in [18].

Work is also under way to extend model outlines to the concept language of OWL 2 DL [13], based on the following remarks:

- A nominal $\{a\}$ is represented by a single diamond (instead of a cluster) labeled "a". This also applies to value restrictions such as $R \ni a$ (or, in Manchester OWL, R **VALUE** a);
- A negated nominal $\neg\{a\}$ is represented by a cluster labeled "**NOT** a";
- A set $\{a_1, \ldots, a_n\}$ is represented by a cluster labeled "**CONTAINED IN** $\{a_1, \ldots, a_n\}$";
- A negated set $\neg\{a_1, \ldots, a_n\}$ is represented by a cluster labeled "**NONE OF** $\{a_1, \ldots, a_n\}$";
- Equivalences are used to convert conjunctions and disjunctions of (possibly negated) nominals or value restrictions into expressions using sets of nominals. For example:

$$\{a\} \sqcup \{b\} \rhd \{a, b\}$$
$$\neg\{a\} \sqcap \neg\{b\} \rhd \neg\{a, b\}$$
$$\neg(R \ni a) \rhd \forall R.\neg\{a\}$$

- Qualified number restrictions correspond to cardinality labels attached to clusters (while non-qualified number restrictions remain attached to boxes);
- Local reflexivity (e.g., $\exists R.$**SELF**) is represented by an arrow from a cluster or a diamond to itself;

While these remarks are rather intuitive, there are more difficult challenges in extending model outlines to OWL 2 DL. Representing concrete domains is one. Deciding how much information from the RBox we can effectively represent in diagrammatic form is another (e.g., inverse roles, role chains). These are issues still to be resolved.

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2nd edition, 2007.

[2] James Bailey, François Bry, Tim Furche, and Sebastian Schaffert. Web and semantic web query languages: A survey. In Norbert Eisinger and Jan Maluszynski, editors, *Reasoning Web*, volume 3564 of *Lecture Notes in Computer Science*, pages 35–133. Springer, 2005.

[3] Johannes Bauer. *Model Exploration to Support Understanding of Ontologies*. PhD thesis, Technische Universität Dresden, 2009. `http://www.tatome.de/uni/d_thesis.pdf`.

[4] Alexander Borgida, Enrico Franconi, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. Explaining ALC subsumption. In *International Workshop on Description Logics (DL'99), Linköping, Sweden*, 1999.

[5] DL query tab. `http://protegewiki.stanford.edu/wiki/DLQueryTab`, last modified on June 4, 2009.

[6] Fernando Náufel do Amaral. Visualizing the semantics (not the syntax) of concept descriptions. In *Proceedings of VI TIL, Vila Velha, ES*, 2008. Available at `http://www.nilc.icmc.usp.br/til/til2008/p336-do_amaral.pdf`.

[7] Fernando Náufel do Amaral. Usability of a visual language for DL concept descriptions. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Proceedings of the Fourth International Conference on Web Reasoning and Rule Systems, Bressanone/Brixen, Italy*, volume 6333 of *Lecture Notes in Computer Science*, pages 27–41. Springer Berlin / Heidelberg, 2010.

[8] Fernando Náufel do Amaral and C. Bazílio. Visualization of Description Logic models. In *The 21st International Workshop on Description Logics (DL2008), Dresden, Germany*, 2008. Available at `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-353/NaufelMartins.pdf`.

[9] Joseph Dumas and Janice Redish. *A Practical Guide to Usability Testing*. Intellect, 1994.

[10] Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In *FLAIRS 2006*, 2006.

[11] Brian R. Gaines. Designing visual languages for description logics. *Journal of Logic, Language and Information*, 18(2):217–250, 2009.

[12] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang. The Manchester OWL syntax. In *OWL: Experiences and Directions*, 2006.

[13] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 57–67. AAAI Press, 2006.

[14] John Howse, Fernando Molina, Sun-Joo Shin, and John Taylor. On diagram tokens and types. In *DIAGRAMS '02: Proceedings of the Second International Conference on Diagrammatic Representation and Inference*, pages 146–160, London, UK, 2002. Springer-Verlag.

[15] Akrivi Katifori, Constantin Halatsis, George Lepouras, Costas Vassilakis, and Eugenia Giannopoulou. Ontology visualization methods—a survey. *ACM Comput. Surv.*, 39(4):Article 10, 2007.

[16] S. Krivov, R. Williams, and F. Villa. GrOWL: A tool for visualization and editing of OWL ontologies. *Journal of Web Semantics*, 5(2):54–57, 2007.

[17] Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining answers from the semantic web: the inference web approach. *Journal of Web Semantics*, 1(4):397–413, 2004.

[18] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

Appendix

A. Proof of correctness

Formally, we can consider model outlines as just another syntax for concept descriptions. As such, the meaning of a model outline M is defined by its translation as computed by the call $\textsc{Descr}(M)$. Analogously, an $\mathcal{ALCN}$ concept description C can be expressed in the form of a model outline by the result of $\textsc{BuildClusterCases}(\textbf{solid}, C)$.

In this appendix, we prove that algorithms DESCR and BUILDCLUSTERCASES work correctly together, in the sense that converting an $\mathcal{ALCN}$ concept description to a model outline and then translating the model outline back to $\mathcal{ALCN}$ yields a concept expression which is equivalent to the original one. More precisely:

Theorem (commutation of algorithms)**.** *For every $\mathcal{ALCN}$ concept description C in modified disjunctive normal form, and for every $s \in \{\textbf{\textit{solid}}, \textbf{\textit{dashed}}\}$:*

$$C \equiv \textsc{Descr}(\textsc{BuildClusterCases}(s, C))$$

Definition (modified disjunctive normal form)**.** *The set of $\mathcal{ALCN}$ concept descriptions in modified disjunctive form (mDNF) is defined inductively. A concept description C is in mDNF iff*

Base: *$C = L_1 \sqcup \cdots \sqcup L_n$, with each L_i a literal; if $n = 0$, then $C = \bot$;*

Induction: *$C = D_1 \sqcup \cdots \sqcup D_n$, with at least one of the D_i not a literal, where each D_i is of the form*

$$A_1 \sqcap \cdots \sqcap A_m$$

where each A_j is either a literal or a conjunction of the form

$$\begin{aligned}&\forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D)\\ &\qquad \sqcup (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)]\\ &\sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n\\ &\sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q\\ &\sqcap K(R_j)\end{aligned}$$

where D, if present, is of the form

$$L'_1 \sqcap \cdots \sqcap L'_{n'} \sqcap C'_1 \sqcap \cdots \sqcap C'_{m'}$$

and where

- *All indices are greater than or equal to zero;*
- *Each C_i and each F_i is in mDNF;*
- *$K(R_j)$ is a set of "natural number intervals" representing cardinality restrictions over role R_j (see Section 4 for more comments);*
- *For any pair of conjuncts A_x and A_y, we have that $x \neq y$ implies $R_x \neq R_y$ (i.e., all information about one role is in one single conjunct);*
- *Each L'_i is a literal;*
- *Each C'_i has the same general form as the A_j's, and as with the A_j's, $x \neq y$ implies $R_x \neq R_y$.*

Notation. *For typesetting reasons, we will use the following abbreviations and conventions in the proof:*

- *"*$\textsc{BCC}(\ldots)$*" for "*$\textsc{BuildClusterCases}(\ldots)$*"*
- *"*$\textsc{BDescr}(\ldots)$*" for "*$\textsc{BoxDescr}(\ldots)$*"*
- *"*$\textsc{BArrow}(\ldots)$*" for "*$\textsc{BuildArrow}(\ldots)$*"*
- *Whenever algorithms* DESCR *and* BOXDESCR *execute a loop in order to assemble a disjunction (resp. a conjunction), the variable that will hold the result is initialized with $\bot$ (resp. $\top$). Here, we will refrain from writing these initial values, as the resulting disjunction (resp. conjunction) is obviously equivalent to the one produced by the algorithms.*

Proof. The proof of the theorem is by strong induction over the structure of concept description C (in mDNF).

Base case.

In this case, $C = L_1 \sqcup \cdots \sqcup L_n$, where each L_i is a literal.

Then $\text{BCC}(s, C)$ will return the model outline

```
(cases (cluster s (OR L1 ··· Ln) ()))
```

which, when given to DESCR, will produce

$$L_1 \sqcup \cdots \sqcup L_n$$

which is equivalent to (in fact, identical with) C.

Inductive step.

Now, let $C = D_1 \sqcup \cdots \sqcup D_n$, with at least one of the D_i not a literal, as in the inductive case of the definition of mDNF.

We assume, as the *inductive hypothesis*, that for all descriptions C_i, F_i and C'_i in the structure of C and for any $s \in \{\textbf{solid}, \textbf{dashed}\}$ the following equivalences hold:

$$C_i \equiv \text{DESCR}(\text{BCC}(s, C_i))$$
$$F_i \equiv \text{DESCR}(\text{BCC}(s, F_i))$$
$$C'_i \equiv \text{DESCR}(\text{BCC}(s, C'_i))$$

We then show that for any $s \in \{\textbf{solid}, \textbf{dashed}\}$

$$C \equiv \text{DESCR}(\text{BCC}(s, C))$$

Without loss of generality, we rewrite C in such a way that all literals are grouped at the beginning:

$$C = L_1 \sqcup \cdots \sqcup L_m \sqcup D_1 \sqcup \cdots \sqcup D_n$$

Then $\text{BCC}(s, C)$ is

```
(cases (cluster s (OR L1 ... Lm) ())
       (cluster s label(D1) (arrows(D1)))
       ⋮
       (cluster s label(Dn) (arrows(Dn))))
```

where "label(D_i)" is the label produced by BCC for the cluster case associated to disjunct D_i, and "arrows(D_i)" is the list of arrows produced by BCC for the cluster case associated to disjunct D_i.

Applying DESCR gives $\text{DESCR}(\text{BCC}(s, C))$, which is of the form

$$\begin{aligned}
&(L_1 \sqcup \cdots \sqcup L_m) \\
&\quad \sqcup (\text{label}(D_1) \sqcap \prod_{\substack{A \in \\ \text{arrows}(D_1)}} \text{BDESCR}(A)) \\
&\quad \sqcup \cdots \\
&\quad \sqcup (\text{label}(D_n) \sqcap \prod_{\substack{A \in \\ \text{arrows}(D_n)}} \text{BDESCR}(A))
\end{aligned} \tag{3}$$

In order to write this description in more detail (and to prove that it is equivalent to C), let us recall that each disjunct D_i is of the form

$$A_1 \sqcap \cdots \sqcap A_p$$

(please refer to the inductive definition of mDNF for details). Without loss of generality, let us group all conjuncts that are literals, writing

$$D_i = L_1 \sqcap \cdots \sqcap L_q \sqcap A_1 \sqcap \cdots \sqcap A_p$$

Then $\text{labels}(D_i)$ will be

```
(AND L1 ... Lq)
```

and $\text{arrows}(D_i)$ will be

$$(\text{BARROW}(A_1) \ \ldots \ \text{BARROW}(A_p))$$

which lets us rewrite the line associated to disjunct D_i in (3) as

$$\begin{aligned}
&L_1 \sqcap \cdots \sqcap L_q \\
&\quad \sqcap \text{BDESCR}(\text{BARROW}(A_1)) \\
&\quad \sqcap \cdots \\
&\quad \sqcap \text{BDESCR}(\text{BARROW}(A_p))
\end{aligned} \tag{4}$$

Next, we will consider, for each conjunct A_j, first the arrow produced by $\text{BARROW}(A_j)$, and then the description produced by $\text{BDESCR}(\text{BARROW}(A_j))$ for this arrow. Our intention is to show that

$$\text{BDESCR}(\text{BARROW}(A_j)) \equiv A_j$$

as this equivalence, when applied to (4) and then to (3), will finally allow us to conclude that

$$C \equiv \text{DESCR}(\text{BCC}(s, C))$$

First, however, we prove the following simple and useful lemma:

Lemma. *The* BARROW *procedure produces an arrow in such a way that it makes no difference in semantics whether the arrow will have a cluster or a box as its source. More precisely, for any concept description C of the form*

$$\begin{aligned}&\forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D)\\ &\qquad \sqcup (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)]\\ &\sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n\\ &\sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q\\ &\sqcap K(R_j)\end{aligned}$$

and any $s \in \{\textbf{\textit{solid}}, \textbf{\textit{dashed}}\}$, we have that

$$\text{DESCR}(\text{BCC}(s, C)) \equiv \text{BDESCR}(\text{BARROW}(C))$$

Proof of lemma.
For any C of the form given, $\text{BCC}(s, C)$ is

```
(cases (cluster s ()(BARROW(C))))
```

Applying DESCR to this model outline yields

$$\top \sqcap \text{BDESCR}(\text{BARROW}(C))$$

which is equivalent to $\text{BDESCR}(\text{BARROW}(C))$.
(End of proof of lemma.)

In what follows, we split the proof that

$$\text{BDESCR}(\text{BARROW}(A_j)) \equiv A_j$$

into 8 cases, which mirror the structure of the code of the BARROW procedure. Recall that A_j is of the form (again, please refer to the inductive definition of mDNF for details)

$$\begin{aligned}&\forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D)\\ &\qquad \sqcup (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)]\\ &\sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n\\ &\sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q\\ &\sqcap K(R_j)\end{aligned}$$

The cases consider different conditions for the values of n, p and q in the form of A_j.

Case 1: $n = 0, p = 0, q = 0$
In this case, $A_j = K(R_j)$.

Note that $K(R_j)$ must be nonempty, or A_j would be $\top$ (a literal), which we know is not true (as all conjuncts that are literals have been grouped before A_1).

If $K(R_j)$ includes an interval containing 0, then $\text{BARROW}(A_j)$ will be

```
(arrow dashed (R_j)(K(R_j))
  (cases
    (box ()()(cluster dashed ()())()))))
```

and $\text{BDESCR}(\text{BARROW}(A_j))$ will be

$$K(R_j) \sqcap \forall R_j.\top$$

which is equivalent to A_j.

If $K(R_j)$ does not include an interval containing 0, then $\text{BARROW}(A_j)$ will be

```
(arrow solid (R_j)(K(R_j))
  (cases
    (box ()(BCC(solid, ⊤))()()))))
```

and $\text{BDESCR}(\text{BARROW}(A_j))$ will be

$$\begin{aligned}&K(R_j)\\ &\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, \top))\\ &\sqcap \forall R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, \top))\end{aligned}$$

which is equivalent to

$$K(R_j) \sqcap \exists R_j.\top \sqcap \forall R_j.\top$$

and hence to A_j.

Case 2: $n = 0, p = 0, q > 0$
In this case,

$$A_j = \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q \sqcap K(R_j)$$

Then $\text{BARROW}(A_j)$ will be

```
(arrow solid (R_j)(K(R_j))
  (cases
    (box
      ()
      (BCC(solid, F_1) ··· BCC(solid, F_q))
      (cluster dashed ()())()))))
```

and $\text{BDESCR}(\text{BARROW}(A_j))$ will be

$$K(R_j) \sqcap \forall R_j.\top \\ \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\ \sqcap \cdots \\ \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q))$$

which, by the inductive hypothesis, is equivalent to

$$K(R_j) \sqcap \forall R_j.\top \\ \sqcap \exists R_j.F_1 \\ \sqcap \cdots \\ \sqcap \exists R_j.F_q$$

and hence to A_j.

Case 3: $n > 0, p = 0, q = 0$

In this case,

$$A_j = \forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D)] \\ \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \sqcap K(R_j)$$

Then $\text{BARROW}(A_j)$ will be

```
(arrow solid (R_j) (K(R_j))
  (cases
    (box
      (label(D))
      (BCC(solid, C_1) ··· BCC(solid, C_n))
      () (arrows(D)) ) ) )
```

Recalling that $D = L'_1 \sqcap \cdots \sqcap L'_{n'} \sqcap C'_1 \sqcap \cdots \sqcap C'_{m'}$, we define label$(D)$ to be

```
(AND L'_1 ··· L'_n')
```

and arrows(D) to be

```
(BARROW(C'_1) ··· BARROW(C'_m'))
```

So $\text{BDESCR}(\text{BARROW}(A_j))$ will be

$$K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\ \sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n))) \\ \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ \sqcap \cdots \\ \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\ \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_1)) \\ \sqcap \cdots \\ \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_{m'}))$$

which, by the lemma, is equivalent to

$$K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\ \sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n))) \\ \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ \sqcap \cdots \\ \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\ \sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_1)) \\ \sqcap \cdots \\ \sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_{m'}))$$

(for any $s \in \{\textbf{solid}, \textbf{dashed}\}$). This, by the inductive hypothesis, is equivalent to

$$K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\ \sqcap \forall R_j.(C_1 \sqcup \cdots \sqcup C_n) \\ \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\ \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{m'}$$

which can be rewritten as

$$K(R_j) \\ \sqcap \forall R_j.[(C_1 \sqcup \cdots \sqcup C_n) \\ \qquad \sqcap (L'_1 \sqcap \cdots \sqcap L'_{n'} \sqcap C'_1 \sqcap \cdots \sqcap C'_{m'})] \\ \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n$$

which is equivalent to A_j.

Case 4: $n > 0, p = 0, q > 0$

In this case,

$$\begin{aligned} A_j = \forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D)] \\ \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\ \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q \sqcap K(R_j) \end{aligned}$$

Then BARROW(A_j) will be

```
(arrow solid (R_j)(K(R_j))
  (cases
    (box
      (label(D))
      (BCC(solid, C_1) ··· BCC(solid, C_n))
      ((InnerBox
        BCC(solid, F_1) ··· BCC(solid, F_q)))
      (arrows(D)))))
```

As in case 3, we define label(D) to be

```
(AND L'_1 ··· L'_n')
```

and arrows(D) to be

```
(BARROW(C'_1) ··· BARROW(C'_m'))
```

So BDESCR(BARROW(A_j)) will be

$$\begin{aligned} & K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\ & \sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ & \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n))) \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ & \sqcap \cdots \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\ & \sqcap \cdots \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q)) \\ & \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_1)) \\ & \sqcap \cdots \\ & \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_{m'})) \end{aligned}$$

which, by the lemma, is equivalent to

$$\begin{aligned} & K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\ & \sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ & \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n))) \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\ & \sqcap \cdots \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\ & \sqcap \cdots \\ & \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q)) \\ & \sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_1)) \\ & \sqcap \cdots \\ & \sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_{m'})) \end{aligned}$$

(for any $s \in \{\textbf{solid}, \textbf{dashed}\}$). This, by the inductive hypothesis, is equivalent to

$$\begin{aligned} & K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\ & \sqcap \forall R_j.(C_1 \sqcup \cdots \sqcup C_n) \\ & \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\ & \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q \\ & \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{m'} \end{aligned}$$

which can be rewritten as

$$\begin{aligned} & K(R_j) \\ & \sqcap \forall R_j.[(C_1 \sqcup \cdots \sqcup C_n) \\ & \qquad \sqcap (L'_1 \sqcap \cdots \sqcap L'_{n'} \sqcap C'_1 \sqcap \cdots \sqcap C'_{m'})] \\ & \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\ & \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q \end{aligned}$$

which is equivalent to A_j.

Case 5: $n > 0, p > 0, q = 0$

In this case,

$$\begin{aligned} A_j = \forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D) \\ (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)] \\ \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \sqcap K(R_j) \end{aligned}$$

Then BARROW(A_j) will be

(arrow solid (R_j) ($K(R_j)$)
  (cases
    (box
      (label(D))
      (BCC(**solid**, C_1) $\cdots$ BCC(**solid**, C_n))
      ((BCC(**dashed**, C_{n+1})
        $\cdots$ BCC(**dashed**, C_{n+p})))
      (arrows(D)))))

As in case 3, we define label(D) to be

(AND $L'_1 \cdots L'_{n'}$)

and arrows(D) to be

(BARROW(C'_1) $\cdots$ BARROW($C'_{m'}$))

So BDESCR(BARROW(A_j)) will be

$$
\begin{aligned}
& K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
& \sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
& \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
& \qquad \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+1})) \\
& \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+p}))) \\
& \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
& \sqcap \cdots \\
& \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
& \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_1)) \\
& \sqcap \cdots \\
& \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_{m'}))
\end{aligned}
$$

which, by the lemma, is equivalent to

$$
\begin{aligned}
& K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
& \sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
& \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
& \qquad \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+1})) \\
& \qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+p}))) \\
& \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
& \sqcap \cdots \\
& \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
& \sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_1)) \\
& \sqcap \cdots \\
& \sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_{m'}))
\end{aligned}
$$

(for any $s \in \{\textbf{solid}, \textbf{dashed}\}$). This, by the inductive hypothesis, is equivalent to

$$
\begin{aligned}
& K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
& \sqcap \forall R_j.(C_1 \sqcup \cdots \sqcup C_n \sqcup C_{n+1} \sqcup \cdots \sqcup C_{n+p}) \\
& \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\
& \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{m'}
\end{aligned}
$$

which can be rewritten as

$$
\begin{aligned}
& K(R_j) \\
& \sqcap \forall R_j.[(C_1 \sqcup \cdots \sqcup C_n \sqcup C_{n+1} \sqcup \cdots \sqcup C_{n+p}) \\
& \qquad \sqcap (L'_1 \sqcap \cdots \sqcap L'_{n'} \sqcap C'_1 \sqcap \cdots \sqcap C'_{m'})] \\
& \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n
\end{aligned}
$$

which is equivalent to A_j.

Case 6: $n > 0, p > 0, q > 0$

In this case,

$$
\begin{aligned}
A_j = \forall R_j.[& (C_1 \sqcap D) \sqcup \cdots \sqcup (C_n \sqcap D) \\
& (C_{n+1} \sqcap D) \sqcup \cdots \sqcup (C_{n+p} \sqcap D)] \\
& \sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\
& \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q \sqcap K(R_j)
\end{aligned}
$$

Then BARROW(A_j) will be

(arrow solid (R_j) ($K(R_j)$)
  (cases
    (box
      (label(D))
      (BCC(**solid**, C_1) $\cdots$ BCC(**solid**, C_n))
      ((InnerBox
        BCC(**solid**, F_1) $\cdots$ BCC(**solid**, F_q))
        (BCC(**dashed**, C_{n+1})
        $\cdots$ BCC(**dashed**, C_{n+p})))
      (arrows(D)))))

As in case 3, we define label(D) to be

(AND $L'_1 \cdots L'_{n'}$)

and arrows(D) to be

$$(\text{BARROW}(C'_1) \cdots \text{BARROW}(C'_{m'}))$$

So BDESCR(BARROW(A_j)) will be

$$\begin{aligned}
&K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
&\sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
&\qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
&\qquad \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+1})) \\
&\qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+p}))) \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
&\sqcap \cdots \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\
&\sqcap \cdots \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q)) \\
&\sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_1)) \\
&\sqcap \cdots \\
&\sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_{m'}))
\end{aligned}$$

which, by the lemma, is equivalent to

$$\begin{aligned}
&K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
&\sqcap \forall R_j.(\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
&\qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
&\qquad \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+1})) \\
&\qquad \sqcup \cdots \sqcup \text{DESCR}(\text{BCC}(\textbf{dashed}, C_{n+p}))) \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_1)) \\
&\sqcap \cdots \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, C_n)) \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\
&\sqcap \cdots \\
&\sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q)) \\
&\sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_1)) \\
&\sqcap \cdots \\
&\sqcap \forall R_j.\text{DESCR}(\text{BCC}(s, C'_{m'}))
\end{aligned}$$

(for any $s \in \{\textbf{solid}, \textbf{dashed}\}$). This, by the inductive hypothesis, is equivalent to

$$\begin{aligned}
&K(R_j) \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
&\sqcap \forall R_j.(C_1 \sqcup \cdots \sqcup C_n \sqcup C_{n+1} \sqcup \cdots \sqcup C_{n+p}) \\
&\sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\
&\sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q \\
&\sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{m'}
\end{aligned}$$

which can be rewritten as

$$\begin{aligned}
&K(R_j) \\
&\sqcap \forall R_j.[(C_1 \sqcup \cdots \sqcup C_n \sqcup C_{n+1} \sqcup \cdots \sqcup C_{n+p}) \\
&\qquad \sqcap (L'_1 \sqcap \cdots \sqcap L'_{n'} \sqcap C'_1 \sqcap \cdots \sqcap C'_{m'})] \\
&\sqcap \exists R_j.C_1 \sqcap \cdots \sqcap \exists R_j.C_n \\
&\sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q
\end{aligned}$$

which is equivalent to A_j.

Case 7: $n = 0, p > 0, q = 0$

In this case,

$$\begin{aligned}
A_j = &\forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_p \sqcap D)] \\
&\sqcap K(R_j)
\end{aligned}$$

Then BARROW(A_j) will be

```
(arrow s (R_j) (K(R_j))
  (cases
    (box
      (label(D))
      ()
      ( (BCC(dashed, L''_1 ⊔ ··· ⊔ L''_{m''})) )
      (arrows(D)) )
    (box
      (label(D))
      ()
      ( (BCC(dashed, C_{m''+1})) )
      (arrows(D)) )
      ···
    (box
      (label(D))
      ()
      ( (BCC(dashed, C_p)) )
      (arrows(D)) ) ) )
```

where $s =$ **dashed** if $K(R_j)$ includes an interval containing 0, or $s =$ **solid** if $K(R_j)$ does not include an

interval containing 0, and where, without loss of generality, we have rewritten the descriptions

$$C_1, \ldots, C_p$$

so as to group all literals first, followed by all nonliterals:

$$L''_1, \ldots, L''_{m''}, C_{m''+1}, \ldots, C_p$$

As in case 3, we define label(D) to be

$$\texttt{(AND } L'_1 \cdots L'_{n'}\texttt{)}$$

and arrows(D) to be

$$(\textsc{BArrow}(C'_1) \cdots \textsc{BArrow}(C'_{m'}))$$

Note that this is the first time in the proof that more than one box case may be generated. When we apply BDescr, each case will correspond to one disjunct in BDescr(BArrow(A_j)):

$$\begin{array}{l}
[K(R_j) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\\
\qquad\quad \mathrm{BCC}(\textbf{dashed}, L''_1 \sqcup \cdots \sqcup L''_{m''})) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\textsc{BDescr}(\textsc{BArrow}(C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\textsc{BDescr}(\textsc{BArrow}(C'_{n'}))] \\
\sqcup [K(R_j) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{dashed}, C_{m''+1})) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\textsc{BDescr}(\textsc{BArrow}(C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\textsc{BDescr}(\textsc{BArrow}(C'_{n'}))] \\
\sqcup \cdots \sqcup \\
[K(R_j) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{dashed}, C_p)) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\textsc{BDescr}(\textsc{BArrow}(C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\textsc{BDescr}(\textsc{BArrow}(C'_{n'}))]
\end{array}$$

By the lemma, this is equivalent to

$$\begin{array}{l}
[K(R_j) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\\
\qquad\quad \mathrm{BCC}(\textbf{dashed}, L''_1 \sqcup \cdots \sqcup L''_{m''})) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_{n'}))] \\
\sqcup [K(R_j) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{dashed}, C_{m''+1})) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_{n'}))] \\
\sqcup \cdots \sqcup \\
[K(R_j) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{dashed}, C_p)) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_{n'}))]
\end{array}$$

(for any $s \in \{\textbf{solid}, \textbf{dashed}\}$). This, by the inductive hypothesis, is equivalent to

$$\begin{array}{l}
[K(R_j) \\
\quad \sqcap \forall R_j.(L''_1 \sqcup \cdots \sqcup L''_{m''}) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{n'}] \\
\sqcup \\
[K(R_j) \\
\quad \sqcap \forall R_j.C_{m''+1} \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{n'}] \\
\sqcup \cdots
\end{array}$$

$$
\begin{array}{l}
\sqcup \\
[K(R_j) \\
\quad \sqcap \forall R_j.C_p \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{n'}]
\end{array}
$$

which is equivalent to A_j.

Case 8: $n = 0, p > 0, q > 0$

In this case,

$$
\begin{array}{l}
A_j = \forall R_j.[(C_1 \sqcap D) \sqcup \cdots \sqcup (C_p \sqcap D)] \\
\qquad \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q \sqcap K(R_j)
\end{array}
$$

Then BARROW(A_j) will be

```
(arrow solid (R_j)(K(R_j))
  (cases
    (box
      (label(D))
      (BCC(solid, F_1) ··· BCC(solid, F_q))
      ( (BCC(dashed, L''_1 ⊔ ··· ⊔ L''_m'')) )
      (arrows(D)) )
    (box
      (label(D))
      (BCC(solid, F_1) ··· BCC(solid, F_q))
      ( (BCC(dashed, C_{m''+1})) )
      (arrows(D)) )
      ...
    (box
      (label(D))
      (BCC(solid, F_1) ··· BCC(solid, F_q))
      ( (BCC(dashed, C_p)) )
      (arrows(D)) ) ) )
```

where, without loss of generality, we have rewritten the descriptions

$$C_1, \ldots, C_p$$

so as to group all literals first, followed by all nonliterals:

$$L''_1, \ldots, L''_{m''}, C_{m''+1}, \ldots, C_p$$

As in case 3, we define label(D) to be

```
(AND L'_1 ··· L'_n')
```

and arrows(D) to be

(BARROW$(C'_1) \cdots$ BARROW$(C'_{m'})$)

Again, as in case 7, several box cases may be generated. When we apply BDESCR, each case will correspond to one disjunct in BDESCR(BARROW(A_j)):

$$
\begin{array}{l}
[K(R_j) \\
\quad \sqcap \forall R_j.\text{DESCR}(\\
\qquad \text{BCC}(\textbf{dashed}, L''_1 \sqcup \cdots \sqcup L''_{m''})) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_{n'})) \\
\quad \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q))] \\
\sqcup [K(R_j) \\
\quad \sqcap \forall R_j.\text{DESCR}(\text{BCC}(\textbf{dashed}, C_{m''+1})) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_{n'})) \\
\quad \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q))] \\
\sqcup \cdots \sqcup \\
[K(R_j) \\
\quad \sqcap \forall R_j.\text{DESCR}(\text{BCC}(\textbf{dashed}, C_p)) \\
\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'}) \\
\quad \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \forall R_j.\text{BDESCR}(\text{BARROW}(C'_{n'})) \\
\quad \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_1)) \\
\quad \sqcap \cdots \\
\quad \sqcap \exists R_j.\text{DESCR}(\text{BCC}(\textbf{solid}, F_q))]
\end{array}
$$

By the lemma, this is equivalent to

$$
\begin{aligned}
&[K(R_j)\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\\
&\qquad \mathrm{BCC}(\textbf{dashed}, L''_1 \sqcup \cdots \sqcup L''_{m''}))\\
&\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'})\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_1))\\
&\quad \sqcap \cdots\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_{n'}))\\
&\quad \sqcap \exists R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{solid}, F_1))\\
&\quad \sqcap \cdots\\
&\quad \sqcap \exists R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{solid}, F_q))]\\
&\sqcup [K(R_j)\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{dashed}, C_{m''+1}))\\
&\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'})\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_1))\\
&\quad \sqcap \cdots\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_{n'}))\\
&\quad \sqcap \exists R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{solid}, F_1))\\
&\quad \sqcap \cdots\\
&\quad \sqcap \exists R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{solid}, F_q))]\\
&\sqcup \cdots \sqcup\\
&[K(R_j)\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{dashed}, C_p))\\
&\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'})\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_1))\\
&\quad \sqcap \cdots\\
&\quad \sqcap \forall R_j.\textsc{Descr}(\mathrm{BCC}(s, C'_{n'}))\\
&\quad \sqcap \exists R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{solid}, F_1))\\
&\quad \sqcap \cdots\\
&\quad \sqcap \exists R_j.\textsc{Descr}(\mathrm{BCC}(\textbf{solid}, F_q))]
\end{aligned}
$$

(for any $s \in \{\textbf{solid}, \textbf{dashed}\}$).

This, by the inductive hypothesis, is equivalent to

$$
\begin{aligned}
&[K(R_j)\\
&\quad \sqcap \forall R_j.(L''_1 \sqcup \cdots \sqcup L''_{m''})\\
&\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'})\\
&\quad \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{n'}\\
&\quad \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q]\\
&\sqcup\\
&[K(R_j)\\
&\quad \sqcap \forall R_j.C_{m''+1}\\
&\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'})\\
&\quad \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{n'}\\
&\quad \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q]\\
&\sqcup \cdots \sqcup\\
&[K(R_j)\\
&\quad \sqcap \forall R_j.C_p\\
&\quad \sqcap \forall R_j.(L'_1 \sqcap \cdots \sqcap L'_{n'})\\
&\quad \sqcap \forall R_j.C'_1 \sqcap \cdots \sqcap \forall R_j.C'_{n'}\\
&\quad \sqcap \exists R_j.F_1 \sqcap \cdots \sqcap \exists R_j.F_q]
\end{aligned}
$$

which is equivalent to A_j.

□