Visualization of Description Logic Models*

Fernando Náufel do Amaral and Carlos Bazílio Martins

Depto. de Ciência e Tecnologia, Pólo Universitário de Rio das Ostras,
Universidade Federal Fluminense, Rio das Ostras, RJ, Brazil
{fnaufel,bazilio}@ic.uff.br

Abstract. Many visualization frameworks for ontologies in general and for concept expressions in particular are too faithful to the syntax of the languages in which those objects are represented (e.g., RDF, OWL, DL). *Model outlines* depart from this tradition in that they consist of diagrams characterizing the class of *models* of a given concept expression. We hope this semantically-oriented visualization strategy will allow users to obtain deeper insights about the meaning of such expressions, thereby preventing errors of design or of interpretation.

1 Introduction

The need to provide graphical representations of ontologies has motivated the development of a variety of visualization frameworks, upon which several browsers have been based. By default, many browsers present at once *almost all* of the information contained in the ontology, relying on the user to filter out data that is not relevant to his/her objectives. However, users are not always able to set up filters that are suitable for their purposes, and may end up with cluttered screens, where they spend valuable time searching for wanted information.

Users may benefit from a more task-oriented visualization strategy, where a browser or editor would present only the information necessary for them to achieve their objectives. Here, we present such a visualization framework, dedicated to the task of interpreting (and possibly editing) DL concept descriptions, such as may be used as necessary/sufficient conditions for classes in an ontology.

In many browsers the level of abstraction is not much higher than the (textual) languages commonly used to represent ontologies, such as RDF and OWL. We believe that a visualization framework should emphasize the *meaning* of expressions and abstract from their *syntax*, enabling users to perceive semantic similarities between expressions that are syntactically very different, and vice-versa. Our proposed framework — *model outlines* — is based on diagrams characterizing the class of models of a given concept expression, which present simple visual objects that are equivalent to (possibly more complicated) syntactical objects in the ontology.

Model outlines may be useful in knowledge engineering, in the task of modeling a domain, where the *meaning* of concept expressions must be well-understood

* Work supported by research grant APQ1 E-26/170.503/2007 from FAPERJ.

DL	Manchester OWL	Meaning
$C, D \to A$	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. 1. $\mathcal{ALCN}$ concept expressions and their meaning

both by authors and users of the ontology. Another area where understanding the meaning of concept expressions is important is *proof explanation* [1, 2], where the user may have to deal with many such expressions comprising the intermediate steps in a long proof.

For simplicity (and because this is work in progress) we concentrate on the description logic $\mathcal{ALCN}$. After the basic definitions, we present the visual items that appear in a model outline, along with an algorithm to produce the concept expression corresponding to a given model outline. This establishes a precise semantics for our proposed visualization framework. We then give an algorithm to construct the model outline corresponding to a given concept expression, illustrate with some examples and close with a discussion about further work.

2 Model Outlines

2.1 Definitions

Fig. 1 defines the language of $\mathcal{ALCN}$ concept expressions, both in the DL syntax and in the Manchester OWL syntax [3]. In the grammar, 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 expressions is given by a nonempty set Δ (the *universe* or *domain*) along with an interpretation $\mathcal{I}$ mapping each concept expression C to a set $\mathcal{I}(C) \subseteq \Delta$, and each role term R to a binary relation $\mathcal{I}(R) \subseteq \Delta^2$. An interpretation $\mathcal{I}$ must map each expression in the first column to the set given in the last column. $\#S$ denotes the cardinality of a set S.

A *literal* is an expression of the form A or of the form $\neg A$, where A is an atomic concept term.

We use "R **NOT** n" as an abbreviation for "**NOT** (R **EXACTLY** n)"; we also use "**FROM** m **THRU** n" as an abbreviation for the conjunction "(R **MIN** m) **AND** (R **MAX** n)" when $m < n$.

2.2 Visual Items and Their Semantics

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 individuals in our model outlines are not named, since their identity is not important and since we do not want to clutter the display with unnecessary information. Instead, we label each cluster of individuals with a concept expression C consisting of a conjunction of literals or of a disjunction of literals. (If C is $\top$, the label is omitted. If C is $\bot$, the symbol "$\varnothing$" is used.) This situation is depicted in Fig. 2(a), with the obvious meaning that the individuals in the cluster satisfy the concept expression C.

In order to display more complex concept expressions, we use *labeled arrows* and *boxes* to show how individuals are related to their role fillers, and we use *case widgets* to show how individuals may satisfy the different cases of a disjunction.

Fig. 2(b) shows a cluster x satisfying the expression $\forall R.C$. An almost universally accepted graphical convention is that dashed or grayed-out lines invoke the idea of absence; we use this convention to stress the fact that x will still satisfy $\forall R.C$ even if x has no fillers for R. We surround such possible fillers with a box to show that they all must satisfy C.

Fig. 2(c) shows a cluster x satisfying the expression $\exists R.C_1 \sqcap \ldots \sqcap \exists R.C_n$. The fact that x may have other fillers for R (besides those satisfying $C_1, \ldots, C_n$) is represented by the unlabeled, dashed/grayed-out cluster at the bottom. For uniformity of notation, we have decided to surround the set of fillers with a box, regardless of whether such fillers come from existential or universal constraints. This requires fewer arrows to be drawn, reducing clutter in the display, as the alternative would be to have separate arrows from x to each filler. The presence of the unlabeled cluster and the fact that the box is unlabeled should make it clear that there are no universal constraints placed on the fillers.

Fig. 2(d) shows a cluster x satisfying $\exists R.C_1 \sqcap \ldots \sqcap \exists R.C_n \sqcap \forall R.D$, with the universal constraint on the fillers indicated by the label under the box.

Fig. 2(e) shows a cluster x satisfying $\exists R.C_1 \sqcap \ldots \sqcap \exists R.C_n \sqcap \forall R.(C_1 \sqcup \ldots \sqcup C_n)$. It is the absence of unlabeled clusters that shows that all the fillers must satisfy at least one of $C_1, \ldots, C_n$.

Fig. 2(f) shows a cluster x satisfying $\exists R.C_1 \sqcap \ldots \sqcap \exists R.C_n \sqcap \forall R.(C_1 \sqcup \ldots \sqcup C_n) \sqcap \forall R.D$. All fillers must satisfy $(C_1 \sqcup \ldots \sqcup C_n) \sqcap D$, where D itself may be a complex expression. Instead of showing such a possibly complicated expression to the user, we distribute its components among the labels in an intuitive way.

Fig. 2(g) shows a cluster x satisfying the expression $\exists R.C_1 \sqcap \ldots \sqcap \exists R.C_n \sqcap \forall R.(C_1 \sqcup \ldots \sqcup C_n) \sqcap \exists R.D_1 \sqcap \ldots \sqcap \exists R.D_n$. The requirement that x have fillers for R satisfying each of $D_1, \ldots D_m$ is indicated by the oval with the word "including", meaning that these added fillers are already included in $C_1 \sqcup \ldots \sqcup C_n$. Note that

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

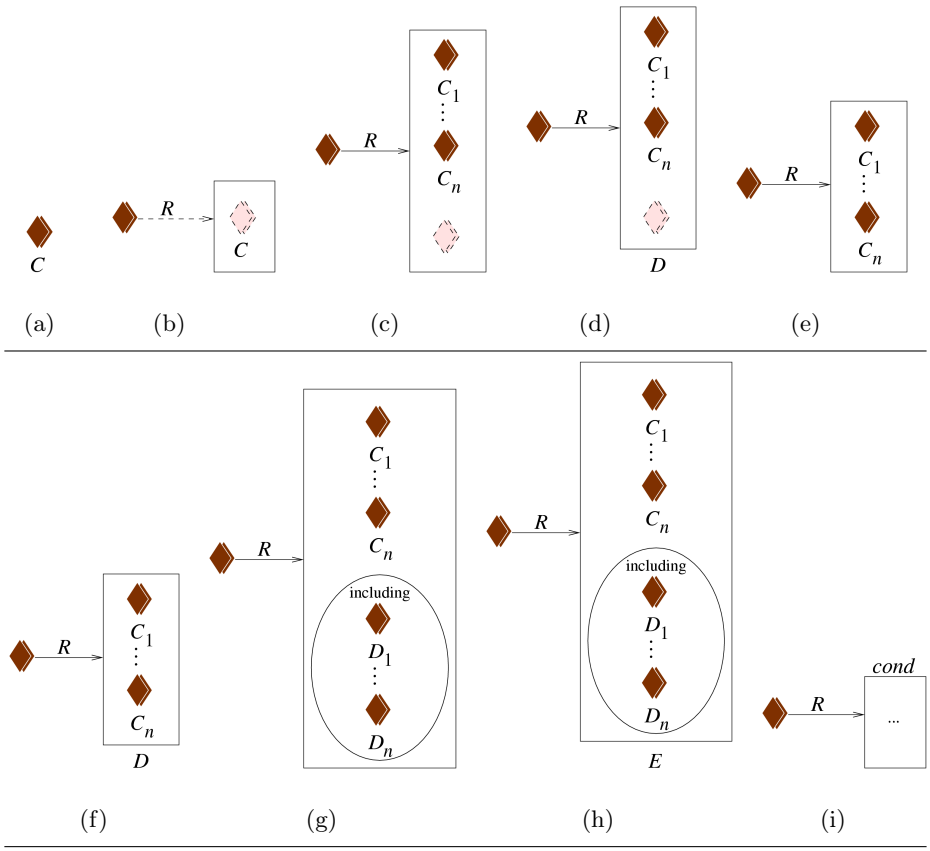

Fig. 2. Visual items that may occur in model outlines

it is not required that all fillers satisfy any of $D_1, \ldots D_m$. In this respect, the oval serves to "isolate" $D_1, \ldots, D_m$, preventing them from participating in the universal constraints on the fillers.

Fig. 2(h) shows a cluster x satisfying the expression $\exists R.C_1 \sqcap \ldots \sqcap \exists R.C_n \sqcap \forall R.(C_1 \sqcup \ldots \sqcup C_n) \sqcap \exists R.D_1 \sqcap \ldots \sqcap \exists R.D_n \sqcap \forall R.E$.

Fig. 2(i) shows the way cardinality restrictions on a role R are represented: a condition *cond* is added as a label above the target box of the arrow for R. Such a condition is of the form "**MIN** n", "**MAX** n", "**EXACTLY** n", "**NOT** n", "**FROM** m **THRU** n", or a list of these, representing a disjunction of cardinality restrictions on R. We think that the use of Manchester OWL syntax makes cardinality restrictions more legible for the nonspecialist user; in particular, "**FROM** m **THRU** n" is more readily understood than, say, "$m \leq x \leq n$", in which the number of individuals in the box would have to be represented, somewhat cryptically, by a variable x. The contents of the box and the nature of the arrow (solid or dashed/grayed out) depend on the minimum number of fillers

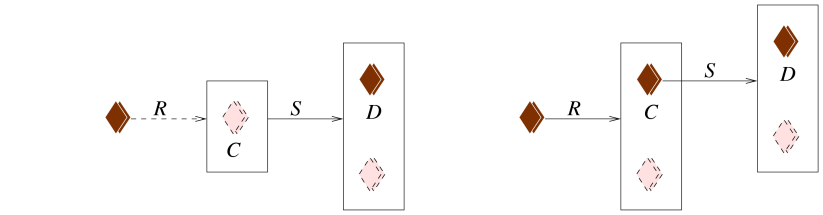

Fig. 3. Universal and existential restrictions

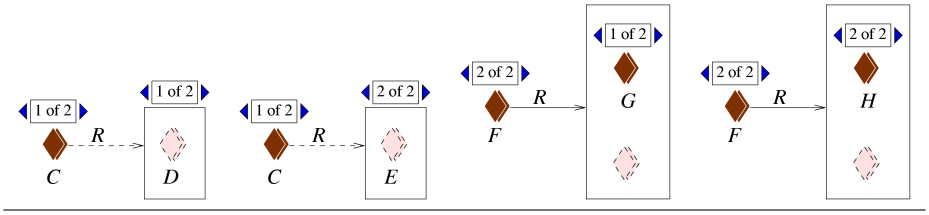

Fig. 4. Complex disjunctions

specified by the cardinality restrictions, as well as on additional quantification constraints for R.

Constraints on the fillers for R may include quantifiers besides (or instead of) conjunctions or disjunctions of literals. Examples would be $\forall R.(C \sqcap \exists S.D)$ and $\exists R.(C \sqcap \exists S.D)$, depicted in Fig. 3. A restriction involving a quantifier is represented by an arrow: if it is a universal restriction (acting upon all fillers in the box), then the arrow leaves the frame of the box, as in (a); if it is an existential restriction (acting upon only one cluster of individuals in the box), then the arrow leaves that cluster, as in (b). The nature of the arrow (solid or dashed), the labels on it and the construction of its target box follow the exact same rules explained above with respect to Figs. 2(a) through (i).

Finally, we show how to represent disjunctions involving more than literals. As explained in Sect. 2.3 below, the algorithm to construct a model outline takes as input a concept expression in disjunctive normal form. The idea is to present one disjunct (i.e., one case) at a time for each cluster, along with a "case widget" for viewing the other cases upon demand. Each case is displayed following the rules explained in the preceding paragraphs. When the user switches to another case for a cluster, the whole subtree rooted at that cluster may change.

As an example, the expression $(C \sqcap \forall R.(D \sqcup E)) \sqcup (F \sqcap \exists R.(G \sqcup H))$ (where C, D, E, F, G and H are conjunctions of literals) gives rise to 4 cases, all of which are represented in Fig. 4. Note that the case widget is placed above clusters introduced by existential restrictions (the clusters labeled by G and by H in

expression(x)

1. $exp \leftarrow \bot$
2. For each case rooted at x (if there is no case widget above x, then x is the root of exactly one case):
   (a) $disjunct \leftarrow L(x)$
   (b) For each arrow of which x is the source:
      i. Let R be the role name labeling the arrow, let b be the target box of the arrow, let K be the disjunction of cardinality restrictions above b, let $y_1, \ldots, y_m$ ($m > 0$) be the clusters directly enclosed by b, and let $z_1, \ldots, z_n$ ($n \geq 0$) be the clusters enclosed by the "including" oval in b
      ii. $disjunct \leftarrow disjunct \sqcap \forall R.(expression(y_1) \sqcup \ldots \sqcup expression(y_m))$
      iii. Let $y_{i_1}, \ldots, y_{i_k}$ be the items directly enclosed by b that are not dashed or grayed out. Then $disjunct \leftarrow disjunct \sqcap \exists R.expression(y_{i_1}) \sqcap \ldots \sqcap \exists R.expression(y_{i_k})$
      iv. $disjunct \leftarrow disjunct \sqcap \exists R.expression(z_1) \sqcap \ldots \sqcap \exists R.expression(z_n)$
      v. $disjunct \leftarrow disjunct \sqcap K$
      vi. $disjunct \leftarrow disjunct \sqcap \forall R.expression(b)$
   (c) $exp \leftarrow exp \sqcup disjunct$

Fig. 5. Algorithm to construct $\mathcal{ALCN}$ expression corresponding to visual item x

the figure), and above boxes corresponding to universal restrictions (the boxes containing the clusters labeled by D and by E in the figure).

Formally, a model outline can be defined as a tree with labeled nodes (corresponding to clusters) and labeled edges (corresponding to arrows). The labels contain information as to how the corresponding clusters and edges should be rendered. A box is just a visual artifact to gather together sibling nodes. In the diagram, an arrow leaving a box is just a visual artifact to represent a set of edges leaving all the children of some node.

To make the semantics of model outlines precise, Fig. 5 gives a recursive algorithm to produce a concept expression corresponding to an item x (a cluster or a box). In the algorithm, the expression $L(i)$ for an item i (a cluster or a box) denotes the label under i (if i is labeled by a concept expression), $\top$ (if i is unlabeled), or $\bot$ (if i is labeled by "$\varnothing$").

2.3 From Concept Expressions to Model Outlines

We now show how to build a model outline for any given $\mathcal{ALCN}$ concept expression C. We start by converting C to disjunctive normal form (DNF), applying simplification rules in the process.[2] This will yield a disjunction of the form $D_1 \sqcup \ldots \sqcup D_n$, where each disjunct D_i is a conjunction. To each D_i we then apply the simplification rule

$$\forall R.C_1 \sqcap \ldots \sqcap \forall R.C_m \;\rhd\; \forall R.(C_1 \sqcap \ldots \sqcap C_m)$$

[2] The set of simplifications applied will be discussed in Sect. 4.

outline(C', v)

1. If $C' = \bot$, then label v with the symbol $\varnothing$ and return.
2. If $C' = \top$, then remove all labels from v and return.
3. If $C' = L_1 \sqcup \ldots \sqcup L_s$, with each L_i a literal, then label v by C' and return.
4. If $C' = L_1 \sqcap \ldots \sqcap L_s$, with each L_i a literal, then label v by C' and return.
5. If none of the above applies, then if the number n of disjuncts in C' is greater than 1, introduce a case widget above v reading "1 of n".
6. For each disjunct D'_i in C' (recall that D'_i has the form shown in (1)): add to v a label and arrows as shown in Fig. 7(a). The resulting diagram is not yet a meaningful model outline: some of the labels may be more complex than conjunctions or disjunctions of literals. So, for each j with $1 \leq j \leq r$, modify the R_j arrow and its target box as follows:
   (a) If K_j is not present in the conjunction, or if $K_j = \top$, then omit the label K_j above the box.
   (b) If $\forall R_j.F_j$ is not present in the conjunction, or if $F_j = \top$, then omit the label F_j below the box.
   (c) If $q_j = 0$ (i.e., there are no conjuncts of the form $\exists R_j.G_{jk}$) and if the disjunction of cardinality restrictions K_j does not imply $> 0.R_j$, then make the R_j arrow dashed/grayed out and move the F_j label from below the box to below the single cluster inside the box, as shown in Fig. 7(b).
   (d) If $q_j > 0$ and if F_j is of the form $(G_{t_1} \sqcap H) \sqcup \ldots \sqcup (G_{t_u} \sqcap H)$ for some conjunction H, with $\{t_1, \ldots, t_u\} \subseteq \{j1, \ldots, jq_j\}$ (see Fig. 7(c)), then
      i. Delete the unlabeled cluster in the box.
      ii. If $\{t_1, \ldots, t_u\} \subsetneq \{j1, \ldots, jq_j\}$, then create an "including" oval in the box and move all clusters with indexes in $\{j1, \ldots, jq_j\} \setminus \{t_1, \ldots, t_u\}$ into the oval.
      iii. Change the label under the box to H.

      Fig. 7(d) shows the result, where $\{w_1, \ldots, w_{q_j - u}\} = \{j1, \ldots, jq_j\} \setminus \{t_1, \ldots, t_u\}$.
   (e) For each visual element x (cluster or box) in the resulting diagram, call *outline*(ℓ, x), where ℓ is the label under x.

Fig. 6. Algorithm to construct model outline rooted at visual item v for $\mathcal{ALCN}$ expression C'. More details are given in the text.

As a result, we obtain C', which is a disjunction $D'_1 \sqcup \ldots \sqcup D'_n$, where each D'_i is of the form

$$L_1 \sqcap \ldots \sqcap L_p \sqcap \forall R_1.F_1 \sqcap \exists R_1.G_{11} \sqcap \ldots \sqcap \exists R_1.G_{1q_1} \sqcap K_1 \sqcap \ldots \\ \sqcap \forall R_r.F_r \sqcap \exists R_r.G_{r1} \sqcap \ldots \sqcap \exists R_r.G_{rq_r} \sqcap K_r \tag{1}$$

where each L_i is a literal, all the F_i and all the G_{ij} are in DNF and each K_i is a disjunction of cardinality restrictions[3] over role R_i. Any (or all) of these elements may be absent.

[3] Obtained by simplification using a suitable set of rewrite rules for intervals of natural numbers. Strictly speaking, this is not DNF, but we believe cardinality restrictions are more readable as a disjunction than as a conjunction of intervals.

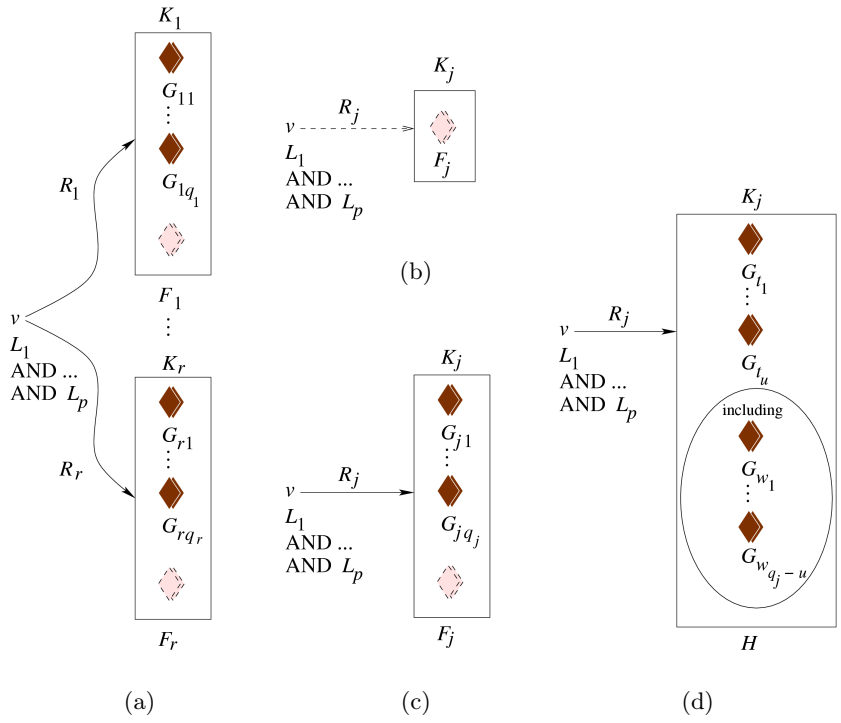

Fig. 7. Situations referenced by algorithm *outline*

Fig. 6 shows the recursive algorithm $outline(C', v)$, where v is the visual item corresponding to C'. Initially, we create a cluster of individuals x (the root of the model outline) and pass it to the algorithm along with C'. Recursive calls will pass as v either a cluster of individuals or a box, according to the case, along with the concept expression that v is supposed to satisfy. Item (d) of step 6 corresponds to the creation of an "including" oval, as illustrated in Fig. 2(g).

Algorithms *expression* and *outline* are related by the following commutation result:

Theorem 1. *Given an $\mathcal{ALCN}$ concept expression C in DNF and a cluster of individuals x, we have that $expression(outline(C, x)) \equiv C$.*

The proof is by induction on the structure of C.

In order to make model outlines more concise, we can refine algorithm *outline* to apply some graphical simplifications, while still preserving the truth of Theorem 1. For example, if there are two or more unlabeled clusters in the same box, we can delete all but one of them. We can also delete unlabeled clusters inside "including" ovals, as well as empty "including" ovals.

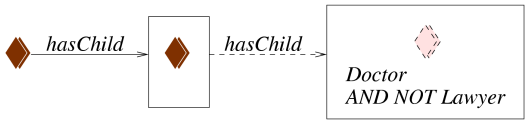

Fig. 8. Model outline for (2)

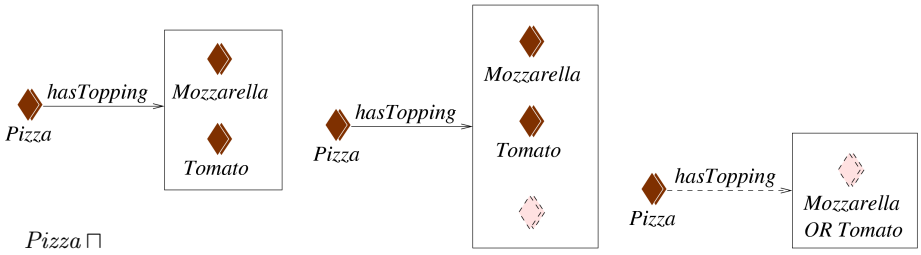

Fig. 9. Correct and incorrect Margheritta pizza specifications

3 Examples

Example 1. To illustrate the use of model outlines in helping users understand the meaning of a complex expression, compare the example in DL syntax (which appears in [1] in the context of proof explanation)

$$\exists hasChild.\top \sqcap \forall hasChild.\neg((\exists hasChild.\neg Doctor) \sqcup (\exists hasChild.Lawyer)) \quad (2)$$

with the corresponding model outline in Fig. 8.

Example 2. To show how model outlines can help prevent common modeling errors, compare the 3 attempts to specify a Margheritta pizza shown in Fig. 9 (see the discussion in [4]). The correct specification is (a). In (b), the presence of the unlabeled cluster will signal that a closure axiom is missing. In (c), the dashed/grayed out objects will alert the user to the fact that this specification can be vacuously satisfied by a pizza with no toppings, and the label "*Mozzarella OR Tomato*" will make it clear that a pizza having only one of those toppings still satisfies this specification.

4 Conclusion

We have presented model outlines, a graphical formalism for characterizing the class of models of a given $\mathcal{ALCN}$ concept expression. We have provided the pre-

cise semantics of model outlines by means of an algorithm (*expression*) yielding an $\mathcal{ALCN}$ concept expression corresponding to a given outline. We have also provided an algorithm (*outline*) for building a model outline for a given $\mathcal{ALCN}$ concept expression. We have shown that given an $\mathcal{ALCN}$ concept expression C in DNF and a cluster of individuals x, we have that $expression(outline(C, x)) \equiv C$.

Our original motivation was the visualization of concept expressions. However, Example 2 in Sect. 3 shows that model outlines can also be useful for *constructing* such expressions. We envision a graphical editor where the user can manipulate elements such as clusters, arrows, boxes and class names to build a model outline embodying the condition he/she intends to specify. Then an algorithm like *expression* can automatically generate such conditions in DL or OWL syntax, for example. This functionality is similar to the one provided by visual query languages (e.g., [5]).

Disjunction is a source of high complexity, as a concept expression in DNF containing n disjunctions may give rise to 2^{n-1} cases. When converting a concept expression to DNF, we can apply a minimization algorithm to try to reduce the number of disjuncts, but the exponential length of DNF is unavoidable in the general case. DNF minimization is an NP-hard problem, which may justify the use of heuristics instead of exact algorithms if the input expression is long.

As for visualization of disjunction, besides showing the total number of cases in a case widget over the root cluster of each case, we plan to use several specific interaction techniques to help the user find his/her way around all the combinations. These techniques include (1) highlighting an entire case when the mouse hovers over one of the visual items that compose the case; in the event the item is "buried" in a chain of nested cases, we plan to use degree-of-interest visualization techniques [6] to make the highlighting of each of the nested cases dependent on its position in the chain; (2) providing statistics about all cases rooted at a cluster selected by the user; (3) providing an overview map, as well as global statistics on the totality of cases in the model outline; and (4) allowing the user to query the set of all cases (e.g., to search for occurrences of some literal in cluster and box labels).

In Sect. 2.3, we mentioned that the construction of a model outline for an input expression C involves the simplification of C. The definition of the set of simplification rules depends on how much reasoning power we want model outlines to have, or, equivalently, how close to the syntactical structure of C we want the model outline to be. For instance, we may want to detect unsatisfiable concept expressions (e.g., $\forall R.\neg C \sqcap \exists R.C \ \rhd \ \bot$). In cases like this, it might be interesting to present to the user the sequence of rewrite steps performed, possibly in animated form.

Our presentation in this paper has considered the meaning of concept expressions with respect to *empty* TBoxes, RBoxes and ABoxes. If the user is working in the context of an ontology, a visualization tool based on model outlines would certainly be more useful if it took into account the axioms about concepts, roles and individuals present in the ontology. For example, the TBox could be used to detect inconsistent boxes (i.e., those where the box label is inconsistent with the

label of some cluster in the box)[4]. The tool could also detect situations where different clusters in a box can be merged (because the clusters' labels represent concepts that are not necessarily disjoint according to the TBox), leading to the construction of a *minimal* model outline in a certain sense.

Work in progress on model outlines includes the investigation of all the points discussed in this section, as well as (1) the conduction of experiments with users from different backgrounds to elicit additional requirements that a visualization/editing tool should meet; (2) the development of a prototype of such a visualization/editing tool, maybe in the form of a plug-in for the Protégé-OWL ontology editor; (3) the extension of the technique to deal with concept languages more expressive than $\mathcal{ALCN}$, such as the one associated with OWL 1.1; and (4) the conduction of deeper comparative studies with other visualization frameworks and tools, such as visual query languages for ontologies (e.g., [5]) and ontology browsers that attempt to follow a more semantically-oriented style (e.g., [7, 8]).

References

1. Borgida, A., Franconi, E., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: Explaining ALC subsumption. In: International Workshop on Description Logics. (1999)
2. McGuinness, D.L., da Silva, P.P.: Explaining answers from the semantic web: the inference web approach. Journal of Web Semantics **1**(4) (2004) 397–413
3. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wan, H.: The Manchester OWL syntax. In: OWL: Experiences and Directions. (2006)
4. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In: Engineering Knowledge in the Age of the SemanticWeb. Volume 3257 of LNCS. (2004)
5. Fadhil, A., Haarslev, V.: OntoVQL: A graphical query language for OWL ontologies. In: International Workshop on Description Logics. (2007)
6. Card, S., Nation, D.: Degree-of-interest trees: A component of an attention-reactive user interface. In: International Conference on Advanced Visual Interfaces (AVI02). (2002)
7. Krivov, S., Williams, R., Villa, F.: GrOWL: A tool for visualization and editing of OWL ontologies. Journal of Web Semantics **5**(2) (2007) 54–57
8. Bosca, A., Bonino, D.: OntoSphere3D: A multidimensional visualization tool for ontologies. In: 7th International Conference on Database and Expert Systems Applications (DEXA), Los Alamitos, CA, USA, IEEE Computer Society (2006)

[4] The tool would have to invoke an external reasoner to obtain such information.