

Functional roles (denoted by \mathcal{F}) is further restricted from \mathcal{N} such that $n = 1$. Therefore, concepts like $(= 2 \text{ hasParent.People})$ will not be allowed in DLs with \mathcal{F} constructors. In particular, *SHIF* (Horrocks et al., 2000) is important because it, when extended with data types, is the DL language corresponding to OWL Lite.

Several other commonly used DLs of the *SH* family include the following:

- *SHIQ* (*ALCHIQ_{R+}*) (Horrocks et al., 1999) is obtained from *SHOIQ* by disallowing the use of nominals;
- *SHOQ* (*ALCHOQ_{R+}*) (Horrocks and Sattler, 2001) is obtained from *SHOIQ* by disallowing the use of inverse roles;
- *SHIO* (*ALCHIO_{R+}*) (Hladik, 2004) is obtained from *SHOIQ* by disallowing the use of (qualified) number restrictions.

The syntax and semantic of the *SH* family DLs are summarized in the Table 2.1.

2.2 Reasoning with Description Logics

2.2.1 Reasoning Tasks for Description Logics

DLs provide a good trade-off between the expressivity power and computational complexity. Many inference tasks with DLs can be solved efficiently with highly optimized DL reasoners, such as FaCT++ (Tsarkov and Horrocks, 2004), RACER (Haarslev and Möller, 2001) and Pellet (Sirin et al., 2007).

Typical reasoning tasks in DL include the follows:

- Subsumption: to test if a concept C is subsumed by another concept D , i.e., if $C \sqsubseteq D$;
- Satisfiability: to test if a concept C is satisfiable, i.e., if there is an individual in an interpretation such that it is an instance of C ;
- Equivalency: to test if two concepts are equivalent, i.e., $C \equiv D$;
- Disjointness: to test if two concepts must have no shared instances;
- Membership: to test if an individual a is an instance of a concept C , i.e., $C(a)$;

Many reasoning problems can be reduced to other reasoning problems. For example, some of them can be reduced to subsumption:

- C and D are equivalent $\Leftrightarrow C \sqsubseteq D$ and $D \sqsubseteq C$
- C and D are disjoint $\Leftrightarrow C \sqcap D \sqsubseteq \perp$.
- a is a member of $C \Leftrightarrow \{a\} \sqsubseteq C$

Subsumption can also be reduced to satisfiability:

- $C \sqsubseteq D \Leftrightarrow C \sqcap \neg D$ is unsatisfiable

It is true because if $C \sqcap \neg D$ is satisfiable, there must an interpretation \mathcal{I} and an element $x \in C^{\mathcal{I}}$ but $x \notin D^{\mathcal{I}}$ therefore $C \sqsubseteq D$ cannot hold. If no such an interpretation can be found, $C \sqcap \neg D$ is unsatisfiable, hence $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in any \mathcal{I} , therefore $C \sqsubseteq D$ is true.

Hence, all reasoning problems mentioned above can be in effect reduced to concept satisfiability checking. In the following discussion, we will focus on concept satisfiability checking only.

2.2.2 Basic Tableau Algorithms

Modern DLs exploit Tableau Algorithms (Baader and Sattler, 2001; Hollunder et al., 1990) for practical reasoning support. The basic idea of tableau algorithm is to check concept satisfiability (and hence also for concept subsumption) w.r.t. a knowledge base by constructing a common model of the concept and the knowledge base.

A tableau algorithm for a specific DL language contains the following main elements:

- A *completion graph*, or a *tableau* that represents a model of the DL language. Such a completion graph typically has the “tree model” property (Vardi, 1996).
- A set of *tableau expansion rules* to construct a complete and consistent completion graph.
- A set of *blocking rules* to detect infinite cyclic models and ensure termination.
- A set of *clash conditions* to detect logic contradictions.

In this subsection, we will demonstrate the basic process of tableau algorithms with the DL \mathcal{ALC} . For an \mathcal{ALC} TBox \mathcal{T} and an \mathcal{ALC} -concept C_0 , a tableau algorithm will construct a common model for both \mathcal{T} and C_0 to checking the satisfiability of C_0 w.r.t. \mathcal{T} . If one such model (i.e., a completion graph) is found, C_0 is satisfiable, otherwise C_0 is unsatisfiable.

Before the reasoning process starts, the concepts in \mathcal{T} and C_0 should be transformed into the *Negation Normal Form* (NNF), i.e., with negation only occurs in front of atomic concepts. It can be done with rewriting rules in Table 2.2. We use $\dot{\neg}C$ to denote the NNF of $\neg C$.

$$\begin{aligned}
\neg\neg C &\equiv C \\
\neg(C \sqcap D) &\equiv \neg C \sqcup \neg D \\
\neg(C \sqcup D) &\equiv \neg C \sqcap \neg D \\
\neg\exists R.C &\equiv \forall R.\neg C \\
\neg\forall R.C &\equiv \exists R.\neg C \\
\neg\leq nR.C &\equiv \geq (n+1)R.C \\
\neg\geq (n+1)R.C &\equiv \leq nR.C \\
\neg\geq 0R.C &\equiv C \sqcap \neg C
\end{aligned}$$

Table 2.2 Negation Normal Form Transformation

Reasoning w.r.t. a TBox \mathcal{T} can be reduced to reasoning w.r.t. an empty TBox with the *internalization* technique. Given \mathcal{T} , a concept $C_{\mathcal{T}}$ is defined as $C_{\mathcal{T}} = \bigcap_{(C_i \sqsubseteq D_i) \in \mathcal{T}} (\neg C_i \sqcup D_i)$. Any individual x in any model of \mathcal{T} will be an instance of $C_{\mathcal{T}}$.

For an \mathcal{ALC} knowledge base, a completion graph or a **tableau** $T = \langle V, E, \mathcal{L} \rangle$ is a tree, where V is the node set, E is the edge set, \mathcal{L} is a function that assigns labels for each node and edge. Each node x in the tree represents an individual in the domain of the model, and the label $\mathcal{L}(x)$ contains all concepts of which x is an instance. Each edge $\langle x, y \rangle$ represents a set of role instances in the model, and the label $\mathcal{L}(\langle x, y \rangle)$ contains the names of those roles. If $R \in \mathcal{L}(\langle x, y \rangle)$, y is an *R-successor* of x . In an \mathcal{ALC} -tableau:

- if $C \in \mathcal{L}(x)$, then $\neg C \notin \mathcal{L}(x)$,
- if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, then $C_1 \in \mathcal{L}(x)$ and $C_2 \in \mathcal{L}(x)$,
- if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, then $C_1 \in \mathcal{L}(x)$ or $C_2 \in \mathcal{L}(x)$,
- if $\forall R.C \in \mathcal{L}(x)$ and $R \in \mathcal{L}(\langle x, y \rangle)$, then $C \in \mathcal{L}(y)$,

- if $\exists R.C \in \mathcal{L}(x)$, then there is some y such that $R \in \mathcal{L}(\langle x, y \rangle)$ and $C \in \mathcal{L}(y)$.

Given a concept C and a TBox \mathcal{T} , the tableau is a tree expanded from an initial root node x_0 , $\mathcal{L}(x_0) = C \sqcap C_{\mathcal{T}}$, with the following **expansion rules**:

- \sqcap -rule: if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$, then $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$;
- \sqcup -rule: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$, then $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1\}$ or $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2\}$;
- \exists -rule: if $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and x has no R-successor y with $C \in \mathcal{L}(y)$, then create a new node y with $\mathcal{L}(\langle x, y \rangle) = \{R\}$ and $\mathcal{L}(y) = \{C\}$;
- \forall -rule: if $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and there is an R-successor y of x with $C \notin \mathcal{L}(y)$, then $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$;
- CE -rule: if $C_{\mathcal{T}} \notin \mathcal{L}(x)$, x is not blocked, then $\mathcal{L}(x) = \mathcal{L}(x) \cup C_{\mathcal{T}}$.

To ensure termination, a node can be blocked with the **subset blocking** strategy: for any node x , if there is an ancestor node y of x in the tree, and $\mathcal{L}(x) \subseteq \mathcal{L}(y)$, then x is blocked. No expansion rule will be applied to a blocked node. In fact, a blocked node prevents the cyclic application of tableau expansion rules, hence represents infinitely many similar individuals in the model.

An \mathcal{ALC} tableau contains a **clash** if there is $\{C, \neg C\} \in \mathcal{L}(x)$ for some node x and concept C . A tableau is **consistent** (clash-free) if it contains no clash, and is **complete** if no expansion rule can be applied. The given concept is satisfiable if and only if the algorithm finds a consistent and complete tableau.

Note that the \sqcup -rule is *non-deterministic* in that it generates different possible tableaux. The algorithm needs to try multiple choices, i.e., *search* for different possible models. Once a chosen search path leads to a clash, the algorithm needs to track back to the tableau state before the choice, and try other remaining choices.

A tableau algorithm has to meet three requirements:

- Soundness: if a complete and consistent tableau is found by the algorithm, the tableau must satisfies the initial concept C_0 .

- Completeness: if the initial concept C_0 is satisfiable, the algorithm can always find an complete and consistent tableau for it.
- Termination: the algorithm can terminate in finite steps with a result.

It can be proven that the aforementioned algorithm is terminating, sound and complete for \mathcal{ALC} (Baader and Sattler, 2001).

2.2.3 Tableau Algorithms for Expressive DLs

Similar tableau algorithms can be designed for more expressive DL languages. In this subsection, we will briefly introduce such an algorithm for \mathcal{SHOIQ} provided by (Horrocks and Sattler, 2005).

A \mathcal{SHOIQ} completion graph is $T = \langle V, E, \mathcal{L}, \neq \rangle$. The symmetric binary relation \neq is used to keep track of inequalities between nodes of T . The introduction of nominal concepts (the “ \mathcal{O} ” constructor) somehow relaxes the strict tree structure (Horrocks and Sattler, 2005) which is enjoyed by \mathcal{ALC} -tableau: a \mathcal{SHOIQ} completion graph contains two types of nodes, i.e., the *blockable nodes* which still form tree structures, and *nominal nodes* which may be arbitrarily interconnected. A nominal node is a node that has nominal names in its labels; such a node cannot be blocked since a blocked node represents infinitely many individuals while a nominal is only allowed to have singleton instances.

If $R \in \mathcal{L}(\langle x, y \rangle)$, y is said an *R-successor* of x and x is an *R-predecessor* of y . *Ancestor* is the transitive closure of predecessor, and *descendant* is the transitive closure of successor. A node y is called an *R-neighbor* of a node x if y is an *R-successor* of x or if x is an $\text{Inv}(R)$ -successor of y .

A node x is **directly blocked** iff none of its ancestors is blocked, and it has ancestors x', y and y' such that

- 1) x is a successor of x' and y is a successor of y' ,
- 2) y, x and all nodes on the path from y to x are blockable,
- 3) $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$, and
- 4) $\mathcal{L}(\langle x', x \rangle) = \mathcal{L}(\langle y', y \rangle) \neq \emptyset$

A node y is **indirectly blocked** iff one of its safe ancestor is blocked. A node is **blocked** if either it is directly blocked or it is indirectly blocked.

An R -neighbor y of x is **safe** if x is blockable or if x is a nominal node and y is not blocked. It is safe in the sense enough R -neighbors for nominal nodes can be generated (Horrocks and Sattler, 2007).

We define $S^T(x, C) := \{y \in V \mid S \in \mathcal{L}(\langle x, y \rangle) \wedge C \in \mathcal{L}(y)\}$ as the set of S -successor of x with C in their labels. For an RBox \mathcal{R} , we denote $\underline{\mathbb{X}}_{\mathcal{R}}$ as is the transitive-reflexive closure over $\mathcal{R} \cup \{\text{Inv}(R) \sqsubseteq \text{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$.

The set of \mathcal{SHOIQ} tableau expansion rules is given as the follows (Horrocks and Sattler, 2007) (the notions of Merge operation will be introduced later):

- \sqcap -rule: **if** $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$, **then** $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$;
- \sqcup -rule: **if** $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$, **then** $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$;
- \exists -rule: **if** $\exists S.C \in \mathcal{L}(x)$, x is not blocked, and x has no safe S -neighbor y of x with $C \in \mathcal{L}(y)$, **then** create a new node y with $\mathcal{L}(\langle x, y \rangle) = \{S\}$ and $\mathcal{L}(y) = \{C\}$;
- \forall -rule: **if** $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and there is an S -neighbor y of x with $C \notin \mathcal{L}(y)$, **then** $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$;
- \forall_+ -rule: **if** $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and there is some R with $\text{Trans}(R)$, $R \underline{\mathbb{X}} S$ and there is an R -neighbor y of x with $\forall R.C \notin \mathcal{L}(y)$, **then** $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$;
- *choose*-rule: **if** $(\leq nS.C) \in \mathcal{L}(x)$, x is not indirectly blocked, and there is an S -neighbor y of x with $\{C, \dot{C}\} \cap \mathcal{L}(y) = \emptyset$, **then** $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \dot{C}\}$;
- \geq -rule: **if** $(\geq nS.C) \in \mathcal{L}(x)$, x is not blocked, and there are no n safe S -neighbors y_1, \dots, y_n of x with $C \in \mathcal{L}(y_k)$ and $y_k \neq y_j$ for each $1 \leq k \leq j \leq n$, **then** create n new nodes y_1, \dots, y_n with $\mathcal{L}(\langle x, y_k \rangle) = \{S\}$ and $\mathcal{L}(y_k) = \{C\}$ and $y_k \neq y_j$ for $1 \leq k \leq j \leq n$;
- \leq -rule: **if** $(\leq nS.C) \in \mathcal{L}(z)$, z is not indirectly blocked, $|S^T(x, C)| \geq n$ and there are two S -neighbors x, y of z with $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \neq y$, **then**
 1. if x is a nominal node, then $\text{Merge}(y, x)$,
 2. else if y is a nominal node or an ancestor of x , then $\text{Merge}(x, y)$,
 3. else $\text{Merge}(y, x)$;
- *o*-rule: **if** for some nominal o there are two nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ and not $x \neq y$, **then** $\text{Merge}(x, y)$;
- *NN*-rule: **if** (1) $(\leq nS.C) \in \mathcal{L}(x)$, x is a nominal node, and there is a blockable S -neighbor

y of x such that $C \in \mathcal{L}(y)$ and x is a successor of y ; (2) there is no m such that $1 \leq m \leq n$, $(\leq mS.C) \in \mathcal{L}(x)$ and there exists m nominal S -neighbors z_1, \dots, z_m of x with $C \in \mathcal{L}(z_k)$ and $z_k \neq z_j$ for $0 \leq k \leq j \leq m$, **then** (1) guess m with $1 \leq m \leq n$, set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\leq mS.C\}$; (2) create m new nodes y_1, \dots, y_m with $\mathcal{L}(\langle x, y_k \rangle) = \{S\}$ and $\mathcal{L}(y_k) = \{C, o_k\}$ for some new nominal o_k and $y_k \neq y_j$ for $0 \leq k \leq j \leq n$;

- **CE-rule:** if $C_{\mathcal{T}} \notin \mathcal{L}(x)$, x is not indirectly blocked, then $\mathcal{L}(x) = \mathcal{L}(x) \cup C_{\mathcal{T}}$.

The Merge operation is used in “shrinking” rules (\leq - and o -rule) to merge one node into another node. More precisely, it contains the following operations (Horrocks and Sattler, 2007):

Pruning: The operation $\text{Prune}(y)$ removes a node y and all blockable successors of y recursively. Formally, it performs the following operations:

1. for all successors z of y , remove $\langle y, z \rangle$ from E and, if z is blockable, $\text{Prune}(z)$;
2. remove y from V .

Merging: Intuitively, the operation $\text{Merge}(y, x)$ merges y into x by letting x inherits all predecessors and nominal successors of y , while prune y and its blockable sub-trees. More precisely, it has the following steps:

1. for all nodes z such that $\langle z, y \rangle \in E$
 - (a) if $\{\langle x, z \rangle, \langle z, x \rangle\} \cap E = \emptyset$;, then add $\langle z, x \rangle$ to E and set $\mathcal{L}(\langle z, x \rangle) = \mathcal{L}(\langle z, y \rangle)$,
 - (b) if $\langle z, x \rangle \in E$, then set $\mathcal{L}(\langle z, x \rangle) = \mathcal{L}(\langle z, x \rangle) \cup \mathcal{L}(\langle z, y \rangle)$,
 - (c) if $\langle x, z \rangle \in E$, then set $\mathcal{L}(\langle x, z \rangle) = \mathcal{L}(\langle x, z \rangle) \cup \{\text{Inv}(S) \mid S \in \mathcal{L}(\langle z, y \rangle)\}$, and
 - (d) remove $\langle z, y \rangle$ from E ;
2. for all nominal nodes z such that $\langle y, z \rangle \in E$
 - (a) if $\{\langle x, z \rangle, \langle z, x \rangle\} \cap E = \emptyset$;, then add $\langle x, z \rangle$ to E and set $\mathcal{L}(\langle x, z \rangle) = \mathcal{L}(\langle y, z \rangle)$,
 - (b) if $\langle x, z \rangle \in E$, then set $\mathcal{L}(\langle x, z \rangle) = \mathcal{L}(\langle x, z \rangle) \cup \mathcal{L}(\langle y, z \rangle)$,
 - (c) if $\langle z, x \rangle \in E$, then set $\mathcal{L}(\langle z, x \rangle) = \mathcal{L}(\langle z, x \rangle) \cup \{\text{Inv}(S) \mid S \in \mathcal{L}(\langle y, z \rangle)\}$, and
 - (d) remove $\langle y, z \rangle$ from E ;
3. set $\mathcal{L}(x) = \mathcal{L}(x) \cup \mathcal{L}(y)$;

4. add $x \neq z$ for all z such that $y \neq z$; and
5. Prune(y).

A *SHOIQ* tableau T contains a **clash** iff one of the following three situations occurs:

- $\{A, \neg A\} \subseteq \mathcal{L}(x)$, for some concept name A and a node x ;
- for some simple role S and a node x , $(\leq nS.C) \in \mathcal{L}(x)$ and there are $n + 1$ S -neighbours y_0, \dots, y_n of x such that $C \in \mathcal{L}(y_i)$ and $y_i \neq y_j$ for all $0 \leq i < j \leq n$;
- for some nominal name o , there are nodes $x \neq y$ with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$.

let O be the set of nominal names that occur in \mathcal{T} , then the tableau algorithm starts with the initial completion graph $T = (\{r_1, \dots, r_l\}, \emptyset, \mathcal{L}, \emptyset)$ such that for every $o \in O$, there is a $r_i \in V$ with $\mathcal{L}(r_i) = \{o\}$. Then T is repeatedly expanded using the expansion rules introduced in the above, until no rule can be applied or a clash occurs.

2.3 Web Ontology Language - OWL

There has been a significant body of recent work on languages for specifying ontologies on the semantic web, including activities on the development of OIL (Ontology Inference Layer) (Fensel et al., 2001), DAML (DARPA Agent Markup Language) (Ouellet and Ogbuji, 2002), their combination DAML+OIL (Horrocks, 2002), and the recent OWL (Web Ontology Language) (Schreiber and Dean, 2004). In particular, OWL has been released as a W3C (World Wide Consortium) recommendation in February 2004, and the last three years witnesses the rapid development and adaption of OWL in a wide range of tools and services.

From the modeling point view, OWL has a strong correspondence to description logics (Horrocks et al., 2003). Concepts and roles in DLs correspond to *classes* and *properties* in OWL, respectively. Table 2.3 summaries OWL class constructors and axioms and their DL correspondences (except for features involving data types). The abstract syntax given in the table can be encoded in RDF/XML as its exchange syntax (Bechhofer et al., 2004). Hence, OWL shares many common features with RDF (Resource Description Framework), such as the use of Universal Resource Identifiers (URI) for the unambiguous reference of web resources. An alternative syntax, the Manchester Syntax (Horridge et al., 2006), is recently proposed to obtain a less verbose syntax that is more friendly for non-logician users.