Structural Description Logics

(Acknowledgement: Enrico Franconi)

- Description Logics
  - The need for a formalism
    - Ambiguities of object-oriented representations
  - A structure upon FOL
    - A predicate level language
- Examples from object-oriented programming
- $\mathcal{FL}^-$: The simplest structural description logic
  - Syntax and Semantics
  - Reasoning problems and reasoning procedures

Description Logics

- A logical reconstruction and *unifying* formalism for (object-oriented) representation tools as
  - Semantic networks
  - Frame-based systems
  - Object-oriented representations
  - Semantic data models
  - Type systems
  - Feature Logics
- A *structured* fragment of predicate logic
- Provide theories and systems for *expressing* structured information and for *accessing* and *reasoning* with it in a principled way.

Applications

Description logics based systems are used in many applications:

- Conceptual modeling
- Terminologies and formal ontologies
- Query optimization and view maintenance
- Natural language semantics representation
- Information access and “intelligent” interfaces
- “Intelligent integration of information”
- Configuration
- Formal specifications in engineering
- Software management
- Planning
- . . .

Object-Oriented Representations

- Identifier
- Class (vs. prototype in frame systems)
- Instance
- Attribute (“slot”)
  - Value:
    Identifier; default
  - Value restriction:
    Type; concrete domain; cardinality; attached method

Ambiguities: Classes and Instances

Person :	AGE :	Number
	SEX :	M, F
	HEIGHT :	Number
	WIFE :	Person

john :	AGE :	29
	SEX :	M
	HEIGHT :	76
	WIFE :	mary

Ambiguities: Classes and Instances

29er :	AGE :	29
	SEX :	M
	HEIGHT :	Number
	WIFE :	Person

john :	AGE :	29
	SEX :	M
	HEIGHT :	Number
	WIFE :	Person

Ambiguities: IS-A

Person :	AGE :	Number
	SEX :	M, F
	HEIGHT :	Number
	WIFE :	Person

has subclass

Male :	AGE :	Number
	SEX :	M
	HEIGHT :	Number
	WIFE :	Female

Ambiguities: IS-A

Male :	AGE :	Number
	SEX :	M
	HEIGHT :	Number
	WIFE :	Female

has instance

john :	AGE :	35
	SEX :	M
	HEIGHT :	76
	WIFE :	mary

Ambiguities: IS-A

29er :	AGE :	29
	SEX :	M
	HEIGHT :	Number
	WIFE :	Person

has instance

john :	AGE :	29
	SEX :	M
	HEIGHT :	76
	WIFE :	mary

Ambiguities: Relations

Implicit relation:

john :	AGE :	35
	SEX :	M
	HEIGHT :	76
	<u>WIFE</u> :	mary
mary :	AGE :	32
	SEX :	F
	HEIGHT :	59
	<u>HUSBAND</u> :	john

Ambiguities: Relations

Explicit relation:

john :	AGE :	35
	SEX :	M
	HEIGHT :	76
mary :	AGE :	32
	SEX :	F
	HEIGHT :	59
m-j-family :	<u>WIFE</u> :	mary
	<u>HUSBAND</u> :	john

Ambiguities: Relations

Special relation: (⇒ Mereology)

$$\texttt{Car} \xrightarrow{\texttt{HAS-PART}} \texttt{Engine}$$

$$\texttt{Engine} \xrightarrow{\texttt{HAS-PART}} \texttt{Valve}$$

$$\Longrightarrow$$

$$\texttt{Car} \xrightarrow{\texttt{HAS-PART}} \texttt{Valve}$$

Normal relation:

$$\text{John} \xrightarrow{\texttt{HAS-CHILD}} \text{Ronald}$$

$$\text{Ronald} \xrightarrow{\texttt{HAS-CHILD}} \text{Bill}$$

$$\not\Longrightarrow$$

$$\text{John} \xrightarrow{\texttt{HAS-CHILD}} \text{Bill}$$

Ambiguities: Defaults

The “Nixon diamond”

$$\begin{tikzcd}
 & \texttt{Pacifist} & \\
\texttt{Quaker} \arrow[ur] & & \texttt{Republican} \arrow[ul, "/" marking] \\
 & \text{Nixon} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Quakers are pacifists, Republicans are not.
⇒ Is Nixon a pacifist?

Remark: Consider depth-first search vs. breadth-first search!

. . . Sceptical inference

Ambiguities: Quantification

What is the exact meaning of $\texttt{Frog} \xrightarrow{\texttt{HAS-COLOR}} \texttt{green}$ **?**

- Every frog is just green
- Every frog is also green
- Every frog is of some green
- There is a frog, which is just green
- . . .
- Frogs are typically green, but there may be exceptions

False Friends

- The meaning of object-oriented representations is logically very ambiguous.
- The appeal of the graphical nature of object-oriented representation tools has led to forms of reasoning that do not fall into standard logical categories, and are not yet well understood.
- Unfortunately, it is much easier to develop some algorithm that appears to reason over structures of a certain kind, than to *justify* its reasoning by explaining what the structures are saying about the domain.

A Structured Logic

- Any (basic) Description Logic is a fragment of FOL.
- The representation is done at the *predicate level*: there are no variables in the formalism.
- A Description Logic theory consists of two parts:
  - definitions of predicates (“TBox”)
  - assertions over constants (“ABox”)
- Any (basic) Description Logic is a subset of $\mathcal{L}_3$, i.e. the function-free FOL using only at most three variable names.

Why not FOL?

If FOL is directly used without any restrictions then

- the structure of the knowledge is destroyed, and it can not be exploited for driving the inference;
- the expressive power is too high for obtaining decidable and efficient inference problems;
- the inference power may be too low for expressing interesting, but still decidable theories.

Structured Inheritance Networks: KL-ONE

- Structured Descriptions
  - corresponding to the complex relational structure of objects,
  - built using a restricted set of epistemologically adequate constructs;
- Distinction between conceptual (*terminological*, T-Box) and instance (*assertional*, A-Box) knowledge;
- Central role of automatic classification for determining the subsumption — i.e., universal (material) implication — lattice;
- Strict reasoning, no defaults.

Types of the TBox Language

- **Concepts** — denote *entities*
  (unary predicates, classes)

  Example: $\texttt{Student}, \texttt{Married}$
  $\{x \mid \texttt{Student}(x)\}, \{x \mid \texttt{Married}(x)\}$
- **Roles** — denote *properties*
  (binary predicates, relations)

  Example: $\texttt{FRIEND}, \texttt{LOVES}$
  $\{\langle x, y\rangle \mid \texttt{FRIEND}(x, y)\}, \{\langle x, y\rangle \mid \texttt{LOVES}(x, y)\}$;

Concept Expressions

Description Logics organize the information in classes — *concepts* — gathering homogeneous data, according to the relevant common properties among a collection of instances.

Example:

$\texttt{Student} \sqcap \exists \texttt{FRIEND}.\texttt{Married}$

$\{x \mid \texttt{Student}(x) \wedge \bigvee_y .\texttt{FRIEND}(x, y) \wedge \texttt{Married}(y).\}$

λ to Define Predicates

Predicates are special case of functions: truth functions.

$\lambda x.P(x)$ denotes a function from domain individuals to truth values.

E.g., we can write for the *unary* predicate Person:

$$\texttt{Person} \doteq \lambda x.\texttt{Person}(x)$$

which is equivalent to say that Person denotes the *set* of persons:

$$\begin{gathered}\texttt{Person} \Longrightarrow \{x \mid \texttt{Person}(x)\}\\ \texttt{Person}^{\mathcal{I}} = \{x \mid \texttt{Person}(x)\}\\ \text{Person(john)} \quad \text{IFF} \quad \text{john}^{\mathcal{I}} \in \texttt{Person}^{\mathcal{I}}\end{gathered}$$

In the same way for the *binary* predicate FRIEND:

$$\begin{gathered}\texttt{FRIEND} \doteq \lambda x, y.\texttt{FRIEND}(x, y)\\ \texttt{FRIEND}^{\mathcal{I}} = \{\langle x, y\rangle \mid \texttt{FRIEND}(x, y)\}\end{gathered}$$

The functions we are defining with the λ operator may be parametric:

$$\texttt{Student} \sqcap \texttt{Worker} = \lambda x.(\texttt{Student}(x) \land \texttt{Worker}(x))$$

$$(\texttt{Student} \sqcap \texttt{Worker})^{\mathcal{I}} = \{x \mid (\texttt{Student}(x) \land \texttt{Worker}(x))\}$$

$$(\texttt{Student} \sqcap \texttt{Worker})^{\mathcal{I}} = \texttt{Student}^{\mathcal{I}} \cap \texttt{Worker}^{\mathcal{I}}$$

(Verify as an exercise)

Concept Expressions

$(\texttt{Student} \sqcap \exists\texttt{FRIEND.Married})^{\mathcal{I}}$

$=$

$(\texttt{Student})^{\mathcal{I}} \cap (\exists\texttt{FRIEND.Married})^{\mathcal{I}}$

$=$

$\{x \mid \texttt{Student}(x)\} \cap \{x \mid \bigvee_y(\texttt{FRIEND}(x, y) \land \texttt{Married}(y))\}$

$=$

$\{x \mid \texttt{Student}(x) \land \bigvee_y(\texttt{FRIEND}(x, y) \land \texttt{Married}(y))\}$

Objects: Classes

Student

Person	
name:	[String]
address:	[String]
enrolled:	[Course]

$$\begin{aligned}&\{x \mid \texttt{Student}(x)\} =\\ &\ \{x \mid \quad \texttt{Person}(x) \land\\ &\qquad \textstyle\bigvee_y(\texttt{NAME}(x, y) \land \texttt{String}(y)) \land\\ &\qquad \textstyle\bigvee_z(\texttt{ADDRESS}(x, z) \land \texttt{String}(z)) \land\\ &\qquad \textstyle\bigvee_w(\texttt{ENROLLED}(x, w) \land \texttt{Course}(w))\}\end{aligned}$$

$$\begin{aligned}\texttt{Student} \doteq\ & \texttt{Person} \sqcap\\ & \exists\texttt{NAME.String} \sqcap\\ & \exists\texttt{ADDRESS.String} \sqcap\\ & \exists\texttt{ENROLLED.Course}\end{aligned}$$

Objects: Instances

s1: Student	
name:	"John"
address:	"Abbey Road"
enrolled:	cs415

$$\begin{aligned}\texttt{Student}(\texttt{s1}) = {} & \texttt{Person}(\texttt{s1})\wedge\\ & \texttt{NAME}(\texttt{s1}, \text{“}\texttt{john}\text{''})\wedge\\ & \texttt{String}(\text{“}\texttt{john}\text{''})\wedge\\ & \texttt{ADDRESS}(\texttt{s1}, \text{“}\texttt{AbbeyRoad}\text{''})\wedge\\ & \texttt{String}(\text{“}\texttt{AbbeyRoad}\text{''})\wedge\\ & \texttt{ENROLLED}(\texttt{s1}, \texttt{cs415})\wedge\\ & \texttt{Course}(\texttt{cs415})\end{aligned}$$

Semantic Networks

$$\begin{tikzcd}
\boxed{\text{Student}} \arrow[r, "\text{enrolled}"] & \boxed{\text{Course}} & \boxed{\text{Professor}} \arrow[l, "\text{teaches}"'] \\
& \boxed{\text{Teaching-assistant}} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

$\bigwedge_x .\texttt{Student}(x) \rightarrow \bigvee_y(\texttt{ENROLLED}(x, y) \wedge \texttt{Course}(y))$.
$\bigwedge_x .\texttt{Professor}(x) \rightarrow \bigvee_y(\texttt{TEACHES}(x, y) \wedge \texttt{Course}(y))$.
$\bigwedge_x .\texttt{Teaching} - \texttt{assistant}(x) \rightarrow (\texttt{Student}(x) \wedge \texttt{Professor}(x))$.

$\texttt{Student} \sqsubseteq \exists\texttt{ENROLLED.Course}$
$\texttt{Professor} \sqsubseteq \exists\texttt{TEACHES.Course}$
$\texttt{Teaching} - \texttt{assistant} \sqsubseteq \texttt{Student}$
$\texttt{Teaching} - \texttt{assistant} \sqsubseteq \texttt{Professor}$

Quantification

$\texttt{Frog} \xrightarrow{\texttt{HAS-COLOR}} \texttt{Green}$

- $\texttt{Frog} \sqsubseteq \exists\texttt{HAS} - \texttt{COLOR.Green}$:
  Every frog is also green
- $\texttt{Frog} \sqsubseteq \forall\texttt{HAS} - \texttt{COLOR.Green}$:
  Every frog is just green
- $\texttt{Frog} \sqsubseteq \forall\texttt{HAS} - \texttt{COLOR.Green}$
  $\texttt{Frog}(x), \texttt{HAS} - \texttt{COLOR}(x, y)$:
  There is a frog, which is just green

$\texttt{Frog} \xrightarrow{\texttt{HAS-COLOR}} \texttt{Green}$

Every frog is also green

$\texttt{Frog} \sqsubseteq \exists\texttt{HAS} - \texttt{COLOR.Green}$

$\bigwedge_x .\texttt{Frog}(x) \rightarrow \bigvee_y(\texttt{HAS} - \texttt{COLOR}(x, y) \wedge \texttt{Green}(y))$.

Exercise: *Is this a model?*

$\texttt{Frog}$(oscar), $\texttt{Green}$(green),
$\texttt{HAS} - \texttt{COLOR}$(oscar,green),
$\texttt{Red}$(red),
$\texttt{HAS} - \texttt{COLOR}$(oscar,red).

$\texttt{Frog} \xrightarrow{\texttt{HAS-COLOR}} \texttt{Green}$

Every frog is only green

$\texttt{Frog} \quad \sqsubseteq \quad \forall\texttt{HAS}-\texttt{COLOR.Green}$

$\bigwedge_x .\texttt{Frog}(x) \rightarrow \bigwedge_y (\texttt{HAS}-\texttt{COLOR}(x,y) \rightarrow \texttt{Green}(y))$.

Exercise: *Is this a model?*

$\texttt{Frog}$(oscar), $\texttt{Green}$(green),
$\texttt{HAS}-\texttt{COLOR}$(oscar,green),
$\texttt{Red}$(red),
$\texttt{HAS}-\texttt{COLOR}$(oscar,red).

. . . and this?

$\texttt{Frog}$(sing), $\texttt{AGENT}$(sing,oscar).

Analytic reasoning

(Let's see this intuitively, by now.)

Person
subsumes
(Person **with every** male friend **is a** doctor)
subsumes
(Person **with every** friend **is a**
(Doctor **with a** specialty **is** surgery))

(Person **with** $\geq$ 2 children) *subsumes* (Person **with** $\geq$ 3 male children)

(Person **with** $\geq$ 3 young children) *disjoint* (Person **with** $\leq$ 2 children)

$\mathcal{FL}^-$

- For the rest of this chapter, we will focus to the simplest conceivable *structural* description logic: $\mathcal{FL}^-$
- We will regard $\mathcal{FL}^-$ as a logical language:
  - Syntax
  - Semantics
  - Reasoning problems:
    decidability and complexity
  - Reasoning procedures
    soundness, completeness, and asymptotic complexity

The Grammar

$C, D \quad \rightarrow A \;|\; C \sqcap D \;|\; \forall R.C \;|\; \exists R$

$A \in$ *atomic-concept*
$R \in$ *atomic-role*
$C, D \in$ *concept*

concept ::= ⟨*atomic-concept*⟩ |
⟨*concept*⟩ ⊓ ⟨*concept*⟩
∃⟨*atomic-role*⟩
∀⟨*atomic-role*⟩.⟨*concept*⟩

Intuitive Model-theoretic Semantics

Intuitively (as we already know):

- *Concepts* represent classes, i.e., sets of individuals.
- *Roles* represent relations between pairs of individuals.
- Atomic concepts are the names of primitive (undefined) concepts.
- $\sqcap$-constructions represent conjoined concepts, so, e.g., `Adult` $\sqcap$ `Male` $\sqcap$ `Person` would represent the concept of something that is at the same time an adult, a male, and a person.
- This allows us to combine several properties (i.e. *super-concepts* or attribute restrictions) in the definition of a concept.

Quantifiers

- The $\forall$-construct provides a concept restriction on the values of an attribute (x is an $\forall$`R.C` if and only if each `R` of x is a `C`). Thus, $\forall$`CHILD.Doctor` corresponds to the concept of something all of whose children are doctors. It is a way to *restrict* the value of a slot at a frame.
- The $\exists$-operator guarantees that there will be at least one value for the attribute named. x is a $\exists$`R` if and only if x has at least one `R`. E.g., `Person` $\sqcap$ $\exists$`CHILD` would represent the concept of a parent. This is a way to *introduce* a slot at a frame.

Formal Semantics

An *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ consists of:

- a nonempty set Δ (the *domain*)
- a function $\cdot^{\mathcal{I}}$ (the *interpretation function*) that maps
  - every *concept* to a subset of Δ
  - every *role* to a subset of $\Delta \times \Delta$
  - every *individual* to an element of Δ

Extension of Concepts

An interpretation function $\cdot^{\mathcal{I}}$ is an extension function iff:

$$\begin{aligned}(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}\\ (\forall R.C)^{\mathcal{I}} &= \{x \in \Delta \mid \bigwedge_y .(x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}.\\ (\exists R)^{\mathcal{I}} &= \{x \in \Delta \mid \bigvee_y (x,y) \in R^{\mathcal{I}}\}\end{aligned}$$

Recall that $C^{\mathcal{I}}$ is the set of all individuals in the extension of C: so, $x \in C^{\mathcal{I}}$ has the same truth value as $C(x)$.

Analogously, $(x,y) \in R^{\mathcal{I}}$ is the same as $R(x,y)$.

The Subsumption Problem

$\mathtt{C} \sqsubseteq \mathtt{D}$, $\mathtt{C}$ *is subsumed* by $\mathtt{D}$ iff
for any domain Δ and any extension function $\cdot^{\mathcal{I}}$ over Δ:
$\mathtt{C}^{\mathcal{I}} \subseteq \mathtt{D}^{\mathcal{I}}$, i.e., $\bigwedge_x . C(x) \rightarrow D(x)$.

Simple examples:

- $\mathtt{Adult} \sqcap \mathtt{Male} \sqsubseteq \mathtt{Adult}$
- $\mathtt{Adult} \sqcap \mathtt{Male} \sqcap \mathtt{Rich} \sqsubseteq \mathtt{Adult} \sqcap \mathtt{Male}$
- $\forall\mathtt{CHILD.Adult} \sqcap \mathtt{Male} \sqsubseteq \forall\mathtt{CHILD.Adult}$
- $\forall\mathtt{CHILD.Adult} \sqcap \exists\mathtt{CHILD} \sqsubseteq \forall\mathtt{CHILD.Adult}$
- $\forall\mathtt{CHILD.Adult} \not\sqsubseteq \exists\mathtt{CHILD}$
- $\exists\mathtt{CHILD} \not\sqsubseteq \forall\mathtt{CHILD.Adult}$

Computational Properties of Subsumption

Subsumption for $\mathcal{FL}^-$ has the following computational properties (constructive proofs sketched):

decidable and **in P**

The *structural* subsumption algorithm

- The algorithm for computing subsumption is based on structural comparisons between concept expressions.
- The central idea of structural comparison is that if the two concept expressions to be compared are made of subexpressions, one can compare separately one subexpression of a concept with all subexpressions of the other concept.

Normal Form

The algorithm works in two phases: first, concepts are rewritten into a normal form, then their structures are compared.

Normal Form:

1. All nested conjunctions are flattened, i.e.
   $A \sqcap (B \sqcap C) \Longrightarrow A \sqcap B \sqcap C$.
2. All conjunctions of universal quantifications are factorized, i.e.
   $\forall R.C \sqcap \forall R.D \Longrightarrow \forall R.(C \sqcap D)$.

The rewritten concepts are logically equivalent to the previous ones, hence subsumption is preserved by this transformation.

(Proof: exercise)

The Core Algorithm: SUBS?[C, D]

Let $C = C_1 \sqcap \ldots \sqcap C_n$ and $D = D_1 \sqcap \ldots \sqcap D_n$ (in normal form).

Then SUBS?[C, D] returns TRUE if and only if for all C_i:

1. if C_i is either an atomic concept, or is a concept of the form $\exists R$, then there exists a D_j such that $C_i = D_j$;
2. if C_i is a concept of the form $\forall R.C'$, then there exists a D_j of the form $\forall R.D'$ (same atomic role R) such that SUBS?[C', D'].

Exercise: Check the examples given above with the structural algorithm.

Asymptotic Complexity

- By induction on the nesting of $\forall$-quantifiers, one can prove that the complexity of the structural algorithm is $O(|C| \times |D|)$ (i.e., quadratic in the length of the longest argument).
- If the subexpressions of each concept are ordered (e.g. lexicographically), it can be shown that the complexity is only linear, so the dominant factor becomes the complexity of ordering concepts.

Soundness

Remember: Whenever a **sound** reasoning procedure claims to have found a solution for a given instance of the problem, then this is actually a solution.

The structural subsumption algorithm is sound, since, when it says that a concept `C` subsumes a concept `D` — i.e., SUBS?$[C, D]$ returns TRUE — then it holds that $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$ for all interpretations.

Observation:
The part of the algorithm computing the normal form does not change the extension of the concepts for any interpretation; thus it does not affect the soundness (and the completeness) of the algorithm.

Informal Proof

- Suppose that SUBS?$[C, D]$ returns TRUE and consider one of the conjuncts of C — call it C_i.
- Either C_i is among the D_j, or it is of the form $\forall R.C'$
- In the latter case, there is a $\forall R.D'$ among the D_j , where SUBS?$[C', D']$.
- Then, by induction, any extension of D' must be a subset of C', and so any extension of D_j must be a subset of C_i's.
- So, no matter what C_i is, the extension of D — which is the conjunction of all the D_j's — must be a subset of C_i.

- Since this is true for every C_i, the extension of D must also be a subset of the extension of C — which is the intersection of all the extensions of C_i.
- So, whenever SUBS?$[C, D]$ returns TRUE, C subsumes D, i.e., $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$

Completeness

Remember: Whenever an instance of the problem has a solution, a **complete** reasoning procedure computes the solution for that instance.

The structural subsumption algorithm is complete, since, whenever it holds that $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$ for all interpretations, then the algorithm says that `C` subsumes `D`.

Proof Idea

- The proof is done by showing that anytime SUBS?$[C, D]$ returns FALSE, there exists an interpretation assigning an element to D but not to C , i.e., in that interpretation the extension of C is not a superset of the extension of D.
- This shows a counter-example, i.e., anytime SUBS?$[C, D]$ returns FALSE it is not true that $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$ for that particular interpretation, and so $D \not\sqsubseteq C$.
- The proof relies on the fact that anytime SUBS?$[C, D]$ returns FALSE it is possible to find a conjunct C_i of C which has no corresp. conjunct in D.
- It is shown that in this case there exists an interpretation which assigns an object to any primitive concept, but not to the factorized one C_i.
- Thus, it is not possible that $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$.

Structural Algorithms or Normalize-and-Compare

What if we enrich the expressivity?

- CLASSIC
- BACK
- LOOM

These systems have incomplete algorithms, due to the interactions between constructors, which cannot be taken into account by structural algorithms which are based on syntactical comparisons between subexpressions of the concepts.

Example: $A \sqcup \neg A$ subsumes every concept, even if such a concept does not mention at all the atomic concepts A in its definition.

Basic References

- Baader, F. et al. (Ed.): *The Description Logic Handbook. Theory, Implementation and Applications.* Cambridge: Cambridge University Press, 2002
- Baader, F.: *Logic-Based Knowledge Representation.*
  In: Artificial Intelligence Today — Recent Trends and Developments.
  Ed. Wooldridge, M.J., Veloso, M. Berlin: Springer (LNCS 1600), 1999, 13–41
- Donini, F. et al.: *Reasoning in Description Logics.*
  In: Principles of Knowledge Representation and Reasoning. Ed. Brewka, G.
  Studies in Logic, Language, and Information.
  Stanford: CSLI Publications, 1996, 193–238
- Levesque, H.J., Brachman, R.J.: *Expressiveness and Tractability in Knowledge Representation and Reasoning.* Computational Intelligence 3 (1987), 78–93.
- Nebel, B.: *Reasoning and Revision in Hybrid Representation Systems.*
  Lecture Notes in Artificial Intelligence 422, Berlin: Springer, 1990. Chapters 1–6.
- Woods, W., Schmolze, J.: *The KL-ONE Family.*
  Computers and Mathematics with Applications, Special Issue: Semantic Networks in Artificial Intelligence, Vol. 2-5 (1992), 133–177.