Title

Antoine Gréa

Abstract

Acknowledgements

Table of Content

1 Introduction

1.1 Thesis Context

1.2 Motivations

1.2.1 Observation

1.2.2 Abstraction

1.2.3 Cognition

1.3 Issues

1.4 Contributions

1.5 Plan

2 Knowledge representation

Knowledge representation is at the intersection of maths, logic, language and computer sciences. Its research begins in the end of the 19th century, with Cantor inventing set theory (Cantor 1874). Then after a crisis in the begining of the 20th century with Russel's paradox and Gödel's incompletude theorem, revised versions of the set theory become one of the fundations of mathematics. The most accepted version is the Zermelo–Fraenkel axiomatic set theory with the axiom of Choice (ZFC) (Fraenkel *et al.* 1973, vol. 67; Ciesielski 1997). This effort was to represent mathematics in itself as a way to formalize its own expression.

A similar process happened in the 1970's, when logic based knowledge representation gained popularity among computer scientists (Baader 2003). Systems at the time explored notions such as rules and networks to try and organize knowledge into a rigourous structure. At the same time other systems were built arround First Order Logic (FOL). Then, around the 1990's, the research began to merge in search of common semantics in what led to the developpement of Description Logics (DL). This domain is expressing knowledge as a hierarchy of classes containing individuals.

From there and with the advent of the world wide web, engineers were on the lookout for standardization and interoperability of computer systems. One such standardization took the name of "semantic web" and aimed to created a widespread network of connected services sharing knowledge between one another in a common language. At the begining of the 21st century, several languages were created, all based around W3C specifications called Resource Description Framework (RDF) (Klyne and Carroll 2004). This language is based around the notion of statements as triples. Each can store a unit of knowledge at a time. All the underlying theoritical work of DL continued with it and created more efficient and lighter derivatives. One such derivative is the family of languages called Web Ontology Language (OWL) (Horrocks *et al.* 2003). Ontologies and knowledge graphs are more recent names for the representation and definition of categories (DL classes), properties and relation between concepts, data and entities.

All these knowledge description systems rely on a syntax to interoperate systems and users to one another. The base of such language comes from the formalization of automated grammars by Chomsky (1956). It mostly consist in a set of hierarchical rules aiming to deconstruct an input string into a sequence of terminal symbols. This deconstruction is called parsing and is a common operation in

computer science. More tools for the formalization of computer language emerged soon after thanks to Backus (1959) while working on a programming language at IBM. This is how the Backus–Naur Form (BNF) meta-language was created ontop of Chomsky's formalization.

All these tools are the base for all modern knowledge representations. In the rest of this chapter, we discuss the fundamentals of each of the aspects of knowledge description, then we propose a knowledge description framework that is able to adapt to its usage.

2.1 Fundamentals

First and foremost, we present the list of notations in this document. While trying to stick to traditional notations, we also aim for an unambiguious notation across several domains while remaining concise and precise. In table 2.1 we present the list of all symbols and relations. When possible we use the classical mathematical operator and otherwise we prefer lower case greek letters for relations. Table 2.2 lists all sets and general notions that we represent with uppercase letters.

2.1.1 Fundation of maths and logic systems

In order to understand knowledge representation, some mathematical and logical tools need to be presented.

2.1.1.1 Logic

The first mathematical notion we define is logic. More precisely First Order Logic (FOL) in the context of DL.

Introducing the two boolean values $\top$ *true* and $\bot$ *false* along with the classical boolean operators $\neg$ *not*, $\wedge$ *and* and $\vee$ *or*. These are defined the following way :

- $\neg\top = \bot$
- $a \wedge b \vDash (a = b = \top)$
- $\neg(a \vee b) \vDash (a = b = \bot)$

With $a \vDash b$ being the logical implication also called entailement.

Table 2.1: List of symbols and relations. The symbol $\pm$ shows when the notation has signed variants.

Symbol	Description
$var : exp$	The colon is a separator to be read as "such that".
$[exp]$	Iverson's brackets: $[false] = 0 \wedge [true] = 1$.
$\{e : exp(e)\}$	Set builder notation, all e such that $exp(e)$ is true.
$\langle e_1, e_2, e_n \rangle$	n-Tuple of various elements.
$e_s \xrightarrow{e_p} e_o$	Link or edge from e_s to e_o having e_p.
$l \downarrow a$	Link l partially supports step a.
$\pi \Downarrow a$	Plan π fully supports a.
$\varnothing$	Empty set, also notted $\{\}$.
$\subset, \cup, \cap$	Set inclusion, union and intersection.
$e \in E$	Element e belongs in set E also called choice operator.
$=$	Equality relation.
$\neg, \wedge, \vee$	Negation (not), conjonction (and), disjonction (logical or).
$\lvert E \rvert$	Cardinal of set E.
$t_1 \prec t_2$	t_1 subsumes t_2. Also used for order : t_1 precedes t_2
$\top, \bot$	Top and bottom symbols used as true and false respectively.
$\models$	Entails, used for logical implication.
$f \circ g$	Function composition.
$\otimes$	Flaws in a partial plan. Several variants :
$\otimes_{\ddagger}$	• unsupported subgoal.
$\otimes_{\dagger}$	• causal threat to an existing causal link.
$\otimes_{\circlearrowleft}$	• cycle in the plan.
$\otimes_{*}$	• decomposition of a compound action.
$\otimes_{\multimap\bullet}$	• alternative to an existing action.
$\otimes_{\circledcirc}$	• orphan action in the plan.
$\oplus, \ominus$	Positive and negative resolvers.
¢	Cost of an action.
$\mathbb{i}, \mathbb{g}$	Initial and Goal step.
$\mathbb{P}(E)$	Powerset, set of all subsets of E.
$\mathcal{P}(X)$	Probability of event X.
pre, *eff*	Preconditions and effects of an action.
δ	Duration of an action.
$\mu^{\pm}(e)$	Signed meta-relation. μ^{+} abstracts and μ^{-} reifies. μ alone gives the abstraction level.
$\nu(e)$	Name of entity e.
π	Plan or method of compound action.
$\rho(e)$	Parameters of entity e.
$\sigma(e)$	Scope of entity e.
$\tau(e)$	Type of entity e.
$\phi^{\pm}(e)$	Signed incidence (edge) and adjacence (vertice) function for graphs. ϕ^{-} gives the anterior (subject), ϕ^{+} the posterior (object) and ϕ^{0} gives the property of statement or cause of a causal link. ϕ alone gives a triple corresponding to the edge.
$\chi^{*}(g)$	Transitive cover of graph g.

Table 2.2: List of important named sets.

Name	Description
A	Actions.
C	Causal links.
D	Domain of knowledge.
E	Entities.
F	Fluents.
L	Literals.
O	Observations.
P	Properties.
Q	Quantifiers.
S	Statements.
T	Types.
Π	Plans or method of compound action.

2.1.1.2 Set theory

We also need to define the notion of set. At the begining of his funding work on set theory, Cantor wrote:

> *A set is a gathering together into a whole of definite, distinct objects of our perception or of our thought—which are called elements of the set.* – George Cantor (1895)

We then define a set the following way:
Definition 1 (Set)**.** A collection of *distinct* objects, considered as an object in its own right. We define a set one of two way (always using braces):

- In extension by listing all the elements in the set: $\{1, 2, 4\}$
- In intension by specifying the rule that all elements follow: $\{n : n \in \mathbb{N} \land (n \mod 2 = 0)\}$

This definition alone gives some properties of sets. By being able to distinguish elements in the set from one another we assert that elements have an identity and we can derive equality from there. We can also give the empty set $\varnothing$ as the set with no elements. Since two sets are equals if and only if they have precisely the same elements, the empty set is unique. The member relation is noted $e \in E$ to indicate that e is an element of E. We note $S \subset G : (e \in S \vDash e \in G) \land S \neq G$, that a set S is a proper subset of a more general set G.

We also define the union and intersection as following:

- $E \cup F : \{e : e \in E \lor e \in F\}$
- $E \cap F : \{e : e \in E \land e \in F\}$

2.1.1.3 Description Logics

On of the most standard and flexible way of representing knowledge for databases is by using ontologies. They are based mostly on the formalism of Description Logics (DL). It is common when using DLs to store statements into three boxes (Baader 2003):

- The TBox for terminology (statements about types)
- The RBox for rules (statements about properties) (Bürckert 1994)
- The ABox for assertions (statements about individual entities)

These are used mostly to separate knowledge about general facts (intentional knowledge) from specific knowledge of individual instances (extensional knowledge). The extra RBox is for "knowhow" or knowledge about entities behaviour. It restrict usages of roles (properties) in the ABox. The terminology is often hierarchically ordered using a subsumption relation noted $\prec$. If we represent classes or type as a set of individuals then this relation is akin to the subset relation of set theory.

2.1.1.4 Graphs

Next in line, we need to define a few notion of graph theory.
Definition 2 (Graph)**.** A graph is a mathematical structure $g = (V, E)$ consisting of vertices V (also called nodes) and edges E (arcs) that links two vertices together. Each edge is basically a pair of vertices ordered or not depending if the the graph is directed or not.

In the following, the signed symbols only applies to dirrected graphs.

We provide graphs with an adjacence function ϕ over any vertex $v \in V$ such that:

- $\phi(v) = \{e \in E : v \in e\}$
- $\phi^+(v) = \{\langle v, v' \rangle \in E : v' \in V\}$
- $\phi^-(v) = \{\langle v', v \rangle \in E : v' \in V\}$

And an incidence function using the same name over any edges $e = \langle v, v' \rangle \in E$ such that:

- $\phi(e) = \langle v, v' \rangle$
- $\phi^-(e) = v$
- $\phi^+(e) = v'$

Properties of graphs are better explained relative to their *transitive cover* χ^* of a graph $g = (V, E)$ defined as follow:

- $\chi(g) = (V, E') : E' = E \cup \{\langle v_1, v_3 \rangle : \{\langle v_1, v_2 \rangle, \langle v_2, v_3 \rangle\} \subset E\}$
- $\chi^2 = \chi \circ \chi$

- $\chi^n = \bigcirc_{i=0}^{n} \chi$
- $\chi^* = \bigcirc^{\infty} \chi$

Now we present two additional notions on graphs.
Definition 3 (Path)**.** We say that vertices v_1 and v_2 are *connected* if it exists a path from one to the other. Said otherwise, there is a path from v_1 to v_2 if and only if $\langle v_1, v_2 \rangle \in E_{\chi^*(g)}$.

Similarly we define *cycles* as the existence of a path from a given vertex to itself. Some graphs can be strictly acyclical, enforcing the absence of cycles.

2.1.1.5 Hypergraphs

A generalization of graphs are **hypergraphs** where the edges are allowed to connect more than two vertices. An hypergraph is n-*uniform* when the edges are restricted to connect only n vertices together. In that regard, classical graphs are 2-uniform hypergraphs.

Another variant of graphs are graphs with the special case where $E \subset V$. This means that edges are allowed to connect to other edges. These kind of graphs are a generalization of hypergraphs. Informations about these kind of structures for knowledge reare hard to come by and rely mostly on a form of "folk wisdom" where knowledge is rarely published and mostly transmitted orally during lessons. One of the closest information is this forum post (Kovitz 2018) that associated this type of graph to port graphs (Silberschatz 1981). Additional information was found in the form of a contribution of Vepstas (2008) on an encyclopedia article about hypergraphs. In that contribution, he says that a generalization of hypergraph allowing for edge-to-edge connections violates the axiom of fundation of ZFC by allowing edge-loops.

This shows the limits of standard mathematics especially on the field of knowledge representation. Some strucutres needs higher dimensions than allowed by the one dimensional structure of ZFC and FOL.

However, it is important to not be mistaken: such non-standard set theories are more general than ZFC and therefore contains ZFC as a subset. All is a matter of restrictions.

2.1.1.6 Sheaf

In order to understand sheaves, we need to present a few auxiliary notions. Most of these definitions are adapted from (Vepštas 2008). The first of which is a seed.
Definition 4 (Seed)**.** A seed correspond to a vertex along with the set of adjacent edges. Formally we note a seed $(v, \phi_g(v))$ that means that a seed build from vertex v in the graph g contains a set of adjacent edges $\phi_g(v)$. We call the vertex v the *germ* of the seed.

Now we can build a kind of partial graphs from seeds called sections.
Definition 5 (Section)**.** A section is a set of seeds that have their common edges connected. This means that if two seeds have an edge in common connecting both germs, then the seeds are connected in the section and the edges are merged.

This tool was originally mostly meant for big data and categorization over large graphs. This is the reason for the next notion.
Definition 6 (Graph Quotient)**.** A quotient over a graph is the act of reducing a subgraph into a node while preserving the external connections. All internal structure becomes ignored and the subgraph now acts like a regular node.

Porting that notion to sections instead of graphs allows us to define stalks.
Definition 7 (Stalk)**.** Given a projection $p : V \to V'$ over the germs of a section s, the stalk above the vertex $v' \in V'$ is the quotient of all seeds that have their germ follow $p(v) = v'$.

Now we can add the final definition of sheaves.
Definition 8 (Seaf)**.** A seaf is a collection of sections, together with a projection function p and gluing axioms that the projection should respect depending on the application.

2.1.2 Grammar and Parsing

Grammar is an old tool that used to be dedicated to liguists. With the funding works of Chomsky and his Contex-Free Grammars (CFG), these tools became available to mathematicians and shortly after to computer scientists.

A CFG is a formal grammar that aims to generate a formal language given a set of hierarchical rules. Each rule is given a symbol as a name. From any finite input of text in a given alphabet, the grammar should be able to determine if the input is part of the language it generates.

In computer science, popular meta-language called BNF was created shortly after Chomsky's work on CFG. The syntax is of the following form :

```
1 <rule> ::= <other_rule> | <terminal_symbol> | "literals"
```

A terminal symbol is a rule that does not depend of any other rule. It is possible to use recursion, meaning that a rule will use itself in its definition. This actually allows for infinite languages. Despite its expressive power, BNF is often used in one of its extended form.

In this context, we present a widely used form of BNF syntax that is meant to be human readable despite not being very formal. We add the repetition operators $*$ and $+$ that respectively repeat 0

and 1 times or more the preceeding expression. We also add the negation operator ~ that matches only if the following expression does not match. We also add parentesis for grouping expression and brackets to group literals.

A regular grammar is static, it is set once and for all and will always produce the same language. In order to be more flexible we need to talk about dynamic grammars and their associated tools.

One of the main tool for both static and dynamic grammar, is a parser. It is the program that will interprete the input into whatever usage it is meant for. Most of the time, a parser will transform the input into another similarily structured language. It can be a storage inside objects or memory, or compiled into another format, or even just for syntax coloration. Since a lot of usage requires the same kind of function, a new kind of tool emerged to make the creation of a parser simpler. We call those tools parser or compiler generators (Paulson 1982). They take a grammar description as input and gives the program of a parser of the generated language as an output.

For dynamic grammar, these tools can get more complicated. There are a few ways a grammar can become dynamic. The most straight-forward way to make a parser dynamic is to introduce code in the rule handling that will tweak variables affecting the parser itself (Souto *et al.* 1998). This allows for handling context in CFG without needing to rewrite the grammar.

Another kind of dynamic grammar are grammar that can modify themselves. In order to do this a grammar is valuated with reified objects representing parts of itself (Hutton and Meijer 1996). These parts can be modified dynamically by rules as the input gets parsed (Renggli *et al.* 2010). This approach uses Parsing Expression Grammars (PEG)(Ford 2004) with Packrat parsing that Packrat parsing backtracks by ensuring that each production rule in the grammar is not tested more than once against each position in the input stream (Ford 2002). While PEG are easier to implement and more efficient in practice than their classical conterparts (Loff *et al.* 2018; Henglein and Rasmussen 2017), it offset the computation load in memory making it actually less efficient in general (Becket and Somogyi 2008).

Some tools actually just infer entire grammars from inputs and softwares (Höschele and Zeller 2017; Grünwald 1996). However, these kind of approaches require a lot of input data to perform well. They also simply provide the grammar after expensive computations.

2.1.3 Ontologies and their Languages

(RDF, OWL, Atomese, and Their limitations)

2.2 WORLD

2.2.1 Knowledge Structure

2.2.2 Dynamic Grammar

2.2.3 Contextual Interpretation

2.2.4 Structure as a Definition

2.2.5 Extended Inference Mechanisms

2.3 Perpectives

2.3.1 Literal definition using Peano's axioms

2.3.2 Advanced Inference

3 General Planning Framework

3.1 Existing Languages and Frameworks

État de l'art

3.2 Taxonomy

3.2.1 Action type

(Définition)

3.2.2 Plan type

(Définition)

3.2.3 Problem type

(Définition)

3.3 Color

(Framework)

4 Online and Flexible Planning Algorithms

4.1 Existing Algorithms

État de l'art

4.2 Lollipop

4.2.1 Operator Graph

4.2.2 Negative Refinements

4.2.3 Usefullness Heuristic

4.2.4 Algorithm

4.2.5 Theoretical and Empirical Results

4.3 HEART

4.3.1 Domain Compilation

4.3.2 Abstraction in POP

4.3.3 Planning in cycle

4.3.4 Properties of Abstract Planning

4.3.5 Computational Profile

4.4 Planning Improvements

4.4.1 Heuristics using Semantics

4.4.2 Macro-Action learning

4.5 Recognition

4.5.1 Existing approcahes

4.5.2 Rico

4.5.2.1 Probabilities and approximations

We define a probability distribution over dated states of the world. That distribution is in part given and fixed and the rest needs computation. **TODO : that's super bad...**

Here is the list of given prior probabilities and asumptions :

- $P(O) = \prod_{o \in O} P(o)$
- $P(\mathcal{G}) = \sum_{G \in \mathcal{G}} P(G) = 1$ since we assume that the agent must be pursuing one of the goals.

- $P(G|\pi) = 1$ for a plan π appliable in I that achieves G.

From dirrect application of Bayes theorem and the previous assomptions, we have :

$$P(\pi|O) = \frac{P(O|\pi)P(\pi)}{P(O)} = \frac{P(O|\pi)P(\pi|G)P(G)}{P(O)} \tag{4.1}$$

$$P(G|O) = \frac{P(O|G)P(G)}{P(O)} \tag{4.2}$$

From the total probability formula :

$$P(O|G) = \sum_{\pi \in \Pi_G} P(O|\pi)P(\pi|G) \tag{4.3}$$

$$P(O|G) = \sum_{\pi \in \Pi_G} P(O|\pi)P(\pi|G) \tag{4.4}$$

5 Conclusion

Apendix

References

Baader, F.
The description logic handbook: Theory, implementation and applications, Cambridge university press, 2003.

Backus, J. W.
The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference, *Proceedings of the International Comference on Information Processing, 1959*, 1959.

Becket, R., and Z. Somogyi
DCGs+ memoing= packrat parsing but is it worth it?, *International Symposium on Practical Aspects of Declarative Languages*, Springer, 2008, 182–196.

Bürckert, H.-J.
Terminologies and rules, *Workshop on Information Systems and Artificial Intelligence*, Springer, 1994, 44–63.

Cantor, G.
On a Property of the Class of all Real Algebraic Numbers., *Crelle's Journal for Mathematics*, 77, 258–262, 1874.

Cantor, G.
Beiträge zur Begründung der transfiniten Mengenlehre, *Mathematische Annalen*, 46 (4), 481–512, 1895.

Chomsky, N.
Three models for the description of language, *IRE Transactions on information theory*, 2 (3), 113–124, 1956.

Ciesielski, K.
Set Theory for the Working Mathematician, Cambridge University Press, August 1997.

Ford, B.
Packrat parsing:: Simple, powerful, lazy, linear time, functional pearl, *ACM SIGPLAN Notices*, ACM, 2002, 37,36–47.

Ford, B.
Parsing expression grammars: A recognition-based syntactic foundation, *ACM SIGPLAN Notices*, ACM, 2004, 39,111–122.

Fraenkel, A. A., Y. Bar-Hillel, and A. Levy
Foundations of set theory, vol. 67Elsevier, 1973.

Grünwald, P.
A minimum description length approach to grammar inference, in S. Wermter, E. Riloff, and G. Scheler (eds.), *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, Lecture notes in computer science; Springer Berlin Heidelberg, 1996, 203–216.

Henglein, F., and U. T. Rasmussen
PEG parsing in less space using progressive tabling and dynamic analysis, *Proceedings of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, ACM, 2017, 35–46.

Horrocks, I., P. F. Patel-Schneider, and F. V. Harmelen
From SHIQ and RDF to OWL: The Making of a Web Ontology Language, *Journal of Web Semantics*, 1, 2003, 2003.

Höschele, M., and A. Zeller
Mining input grammars with AUTOGRAM, *Proceedings of the 39th International Conference on Software Engineering Companion*, IEEE Press, 2017, 31–34.

Hutton, G., and E. Meijer
Monadic parser combinators, 1996.

Klyne, G., and J. J. Carroll
Resource Description Framework (RDF): Concepts and Abstract Syntax, Language Specification. Ed. B. McBride W3C, 2004.

Kovitz, B.
Terminology - What do you call graphs that allow edges to edges?, *Mathematics Stack Exchange*, January 2018.

Loff, B., N. Moreira, and R. Reis
The Computational Power of Parsing Expression Grammars, *International Conference on Developments in Language Theory*, Springer, 2018, 491–502.

Paulson, L.
A semantics-directed compiler generator, *Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '82*, Albuquerque, Mexico: ACM Press, 1982, 224–233. doi:10.1145/582153.582178.

Renggli, L., S. Ducasse, T. Gîrba, and O. Nierstrasz
Practical Dynamic Grammars for Dynamic Languages, *Workshop on Dynamic Languages and Applications*, Malaga, Spain, 2010, 4.

Silberschatz, A.
Port directed communication, *The Computer Journal*, 24 (1), 78–82, January 1981. doi:10.1093/comjnl/24.1.78.

Souto, D. C., M. V. Ferro, and M. A. Pardo
Dynamic Programming as Frame for Efficient Parsing, *Proceedings SCCC'98. 18th International Conference of the Chilean Society of Computer Science (Cat. No.98EX212)(SCCC)*, November 1998, 68. doi:10.1109/SCCC.1998.730784.

Vepstas, L.
Hypergraph edge-to-edge, *Wikipedia*, May 2008.

Vepštas, L.
Sheaves: A Topological Approach to Big Data, 2008.