Boolean Reasoning with Graphs of Partitions

Daniel Goossens

Boolean Reasoning with Graphs of Partitions

Daniel Goossens
Laboratoire d'Informatique Avancee de Saint-Denis
University Paris 8, 2 rue de la Liberte, F-93526 Saint-Denis,
Email: goo@ai.univ-paris8.fr

***Abstract*—This paper presents an implemented architecture for easy learning, reorganizing and navigation into a Boolean knowledge base. As the base grows with new definitions and constraints, it is normalized by the closure of a completion operator. This normalization allows arbitrary formats for Boolean expressions. It ensures basic reasoning abilities and spontaneously organizes intermingled taxonomies of concepts. User interaction is done with three operators : activation, inhibition and forced contradiction. Complex constraints, such as instantiations of first-order expressions, may be programmatically added. A complete illustrative scenario is developed around a knowledge base on elementary school geometry and the approach is situated in the AI field.**

I. Introduction

Boolean knowledge is mainly used in AI with the objective of solving decision problems. As SAT solvers have become more efficient, grounding and then solving is often preferable than reasoning with first-order encodings.

However, Boolean knowledge may also be organized to allow for interactive construction of Boolean Knowledge Bases (BKB), navigation in virtual lattices, specification and, eventually, problem solving.

In this purpose, this article proposes a BKB architecture based on graphs of partitions. The emphasis is not on the class of queries that are answered but on the way the base is updated to correct a wrong answer to a query. Teaching and querying the base is done with three simple operators : activation, inhibition and forced contradiction. Complex constraints, such as instantiations of first-order expressions, may be programmatically added. As definitions and constraints are added to the base, it is updated through a normalization mechanism based on the closure of a simple completion operator. Through user interaction, normalization spontaneously and progressively organizes overlapping taxonomies of concepts which allow to answer "which" and "what" questions.

The objective of this paper is to illustrate the use of the architecture and situate it in the AI field, at the expense of algorithmic details and formal presentation.

The architecture is illustrated with a simple but structured example of elementary school geometry : a classification of quadrilaterals. Starting empty, the base is progressively taught, updated, generalized, reorganized and permanently queried. It is emphasized that many different learning sequences will lead to the same result.

The paper is organized as follows : Section II presents basic definitions. Section III develops the BKB architecture and machinery. Section IV illustrates its use. Section V situates the present proposal relatively to other approaches. Section VI concludes.

II. Basic Definitions

Graphs of partitions are oriented hypergraphs. Hyperlinks are called partitions. A *partition* p is a couple $<t, \{f_1 \dots f_n\}>$, where t and $f_1 \dots f_n$ are nodes. t is the *head* of p and $f_1 \dots f_n$ are its *leaves*. A *graph of partitions* G is a couple $<N, L>$. N is a set of nodes. L is a set of partitions whose nodes are members of N. It is drawn as in Figure 1.

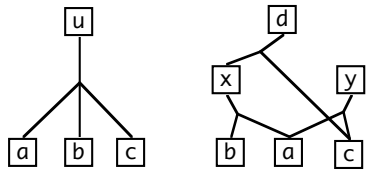

Fig. 1. A partition is drawn as an oriented hyperlink. Partitions may be connected so as to form arbitrarily intermingled hierarchies. The left figure is the partition $u = a + b + c$, with head u and leaves a, b, c. The right graph contains the bipartitions $x = a + b$, $y = a + c$, $d = x + c$.

Each node of a graph of partitions is a leaf of its *top links* and the head of its *bottom links*.

Each node may be interpreted as a Boolean variable and each partition as a Boolean constraint on its nodes. A bipartition $<x, \{y, z\}>$ is a Boolean constraint $(x \leftrightarrow (y \vee z)) \wedge \neg(y \wedge z)$ on the Boolean variables x, y, z. It is abbreviated as $x = y + z$. This interpretation generalizes to an n-ary partition $<t, \{f_1 \dots f_n\}>$, abbreviated as $t = f_1 + \dots + f_n$. The f_i are mutually disjoint and under this constraint, t is indifferently the OR or the XOR of the f_i. When its head is true, a partition reduces to an "Exactly one" clause, true when exactly one of its literals is true and false otherwise.

Under this Boolean interpretation, a graph of partitions is a conjunction of partitions. A *valuation* of a graph of partitions G is a conjunction of constraints $x = 0$ or $x = 1$, where x is a node of G and 0 and 1 are the Boolean values $false$ and $true$. A *valuated* graph of partitions is a conjunction $G \wedge Val$, where G is a graph of partitions and Val is a valuation of G.

III. Representation and Use of Boolean Knowledge

A. Representation of Boolean Expressions

While a true propositional clause prohibits a single of the 2^n valuations of its n variables, a partition prohibits all but n valuations of its n variables. Partitions are thus less precise than clauses for prohibiting valuations. Moreover, graphs of partitions are always 0-valid (satisfied by valuating all nodes to 0), while conjunctions of clauses may be contradictory.

But partitions show an original feature. Any of the nodes of a partition is expressible from the others as a Boolean expression. The Boolean translation $(x \leftrightarrow (y \vee z)) \wedge \neg(y \wedge z)$ of the partition $x = y + z$ implies $x \leftrightarrow (y \vee z)$, $y \leftrightarrow (x \wedge \overline{z})$ and $z \leftrightarrow (x \wedge \overline{y})$ (the dependency is even linear : $x \leftrightarrow (y \oplus z)$, $y \leftrightarrow (x \oplus z)$ and $z \leftrightarrow (x \oplus y)$, that is, $\neg(x \oplus y \oplus z)$, are all implied as well. This property is exploited in [3]).

This small difference with clauses has the consequence that partitions allow to *organize* functional dependencies. For any Boolean basis, it is possible to build *Boolean graphs* (graphs of partitions all of whose nodes are Boolean expressions of the basis), including the complete Boolean lattice generated by the basis (Figure 2).

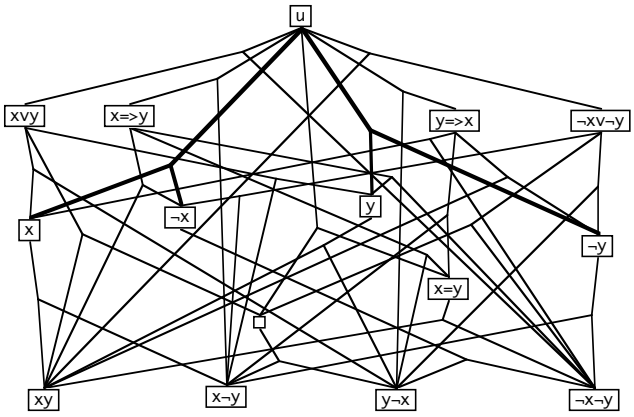

Fig. 2. The complete Boolean lattice generated by the basis $\{x, y\}$. Names are optional. The blank node, for instance, is $x \oplus y$. The value 0 (false) is not represented.

The leaves of a partition may be seen as a basis of atoms from which the complete Boolean lattice may be generated. As a meta construction, one can start with a set of nodes $x_1 \dots x_n$ seen as a free Boolean basis and build nodes interpreted as Boolean expressions of the basis. The starting graph contains the partitions $(u = x_1 + y_1) \dots (u = x_n + y_n)$. Negation is interpreted as complementation relatively to u. Each y_i is $u - y_i$, thus $\overline{x_i}$. Let x and y be two nodes of a Boolean graph G. The graph $G \wedge (x = a+b) \wedge (y = b+c) \wedge (d = a+b+c)$, where a, b, c, d are nodes out of G, is a Boolean graph. It implies $(a \leftrightarrow (x \wedge \overline{y})) \wedge (b \leftrightarrow (x \wedge y)) \wedge (c \leftrightarrow (\overline{x} \wedge y)) \wedge (d \leftrightarrow (x \vee y))$. This is easily verifiable from the Boolean translations of the partitions.

Nodes may then be interpreted as Boolean expressions of a subset of the nodes, considered as a basis of Boolean variables. In this setting, nodes may be *contradictory* or *equivalent*. Without simplification, these constructions introduce contradictory and equivalent nodes. The construction of the Boolean graph of Figure 2 needs simplification mechanisms. However, since the construction includes negation and conjunction, any Boolean expression of the basis may be built in linear size. By this construction, the expression is *reified* (symbolized) as a single node.

B. Boolean Reasoning

Boolean Reasoning on a graph of partitions is based on propagations of Boolean values, deduction of new partitions, suppression of contradictory nodes, merging of equivalent nodes and on a normalization based on a completion operator.

Propagation of Boolean values corresponds to the application of Boolean Constraint Propagation (BCP) [1] on the translation of partitions into propositional clauses.

Normalization is the closure of a completion operator. Let a *hierarchy* be a connected graph such that each node has at most one top partition and at most one bottom partition. The node with only a bottom partition is its head and the nodes with only a top partition are its leaves. If in a graph G, a hierarchy h contains all the nodes but one of a partition p of G not in h, the operator *completes* G with a partition q such that p and q are the partitions of a hierarchy h' parallel to h, that is, with the same head and leaves. The right graph of Figure 1, for instance, would be completed with the partition $d = b+y$. Normalization computes the closure of this operator (see Figure 3).

After each completion, the graph is simplified : redundant partitions (a parallel hierarchy with the same head and leaves already exists) are suppressed or reduced. Contradictory nodes detectable by propagation are suppressed and the remaining nodes of their top bipartitions are merged.

More powerful completion operators have all shown exponential worst case behaviour. The worst case complexity of this closure is still an open problem but it remains efficient in practice, as long as graphs do not get too dense. The normalization algorithm is a control of the closure with avoids many redundancies due to the great number of symetries of the problem.

This reasoning gradually builds *set structures* (partial powerset of the set of leaves of a hierarchy) where Boolean reasoning is complete and efficient and a uniqueness principle is maintained : locally contradictory nodes are suppressed and locally equivalent nodes are merged. The whole graph of Figure 5, for instance, is a single set structure.

This is the main characteristic which makes navigation practical. Within a set structure, nodes are directly connected to their local extension, transitively following their bottom links, and to their local intension, via their top links.

The intension of a node n is defined here as its properties accessible by propagation. If $n = 1$ is propagated, the nodes valuated to 1 are properties it possesses and those valuated to 0 are properties it does not possess. The extension of n contains the nodes valuated to 0 after $n = 0$ has been propagated.

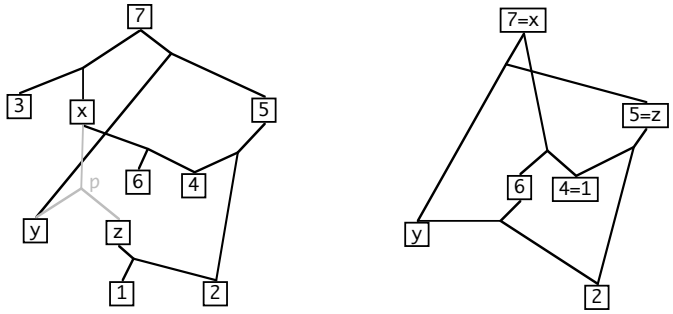

Fig. 3. After partition p is added, normalization of the left graph gives the right graph. The hierarchy $\{7 = 3 + x, p\}$ and the partition $7 = y + 5$ are completed with a partition q: $5 = 3 + z$. Then $\{q,\ z = 1 + 2\}$ and $5 = 4 + 2$ are completed with r: $4 = 3 + 1$. 3 is inferred contradictory by propagation and suppressed. This entails the fusions of the heads and brothers of its top links : $7 = x$, $5 = z$, $4 = 1$. Then $\{7 = 5 + y, 5 = 4 + 2\}$ and $x = 4 + 6$ are completed with $6 = y + 2$.

Intensions and extensions are sets of nodes and they may be intersected, added and complemented to answer queries.

The set structures may get dense. To keep them sparse, a *variable elimination operator* is available. It is an analog of propositional variable elimination, as it is used in the preprocessing phase of some SAT solvers [8]. Any node n with top and bottom links may be suppressed. The node is eliminated and each pair $\{p, q\}$ of a top link p and a bottom link q of n is replaced by a partition containing the nodes of p and q except n (see Figure 4). The operator is multiplicative. If n has α top links and β bottom links, it may add $\alpha \times \beta$ new partitions. It may thus trigger a combinatorial explosion if repetitively used without precaution.

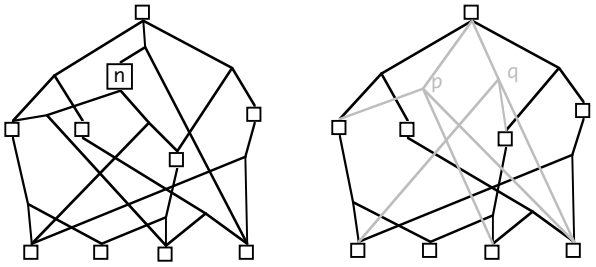

Fig. 4. Elimination of node n from the left graph gives the right graph. The grey partitions p and q are then suppressed because they are redundant.

This makes graphs of partitions a formalism for representing and reasoning with Boolean knowledge. The expressions built as nodes are permanently inter-classified in a dynamic network of implications. The network is densified as new definitions and constraints are added and sparsified using the variable elimination operator.

C. Construction of a Boolean Knowledge Base

A Boolean Knowledge Base (BKB) as considered here is a graph of partitions. It is initially empty and is grown solely by adding definitions. A definition is either a Boolean variable or a Boolean equivalence between a Boolean variable and a Boolean expression. The Boolean expression is reified as the Boolean variable.

For instance, a square is a rhombus with a right angle. If $rhombus$ and $right\_angle$ have been previously defined, $square$ is defined as : $square \leftrightarrow (rhombus \wedge right\_angle)$.

Such a definition is a constraint on its variables which may be *activated* (set to $true$), *inhibited* (set to $false$), *unactivated* (left non valuated) or declared contradictory (*forced contradiction*). A node which is declared contradictory is suppressed from the graph. If G is the graph and n is the node, the suppression is done by propagating $n = 0$ on G and simplifying partitions as would be done on their Boolean translation, and normalizing G. To definitely impose a Boolean constraint c on the variables of the BKB, add a definition $v \leftrightarrow \overline{c}$, where v is a new variable, and declare v contradictory. It definitely makes $\overline{c}$ false, thus, c true.

A definition is activated by valuating its reified variable to $true$. The constraint expressed by its Boolean expression is then imposed ont its variables. When $square$ is $true$, for instance, $rhombus$ and $right\_angle$ are $true$. To inhibit the definition, its reified variable is valuated to $false$. When $square$ is $false$, $rhombus$ and $right\_angle$ may not be both $true$. It is as if the constraint $\neg square$ is active. When $square$ is not valuated, $\{rhombus,\ right\_angle\}$ is a free Boolean basis. If $square$ is declared contradictory, the BKB definitely reflects a world where $rhombus$ and $right\_angle$ are incompatible properties.

Some frequent constructions may be abbreviated :

Let x and y be two nodes of the BKB. To make x imply y, construct $x \wedge \overline{y}$ as a node and declare it contradictory. To make x and y incompatible, construct $x \wedge y$ as a node and declare it contradictory. Each of these two constraints may be added more directly with a single bipartition, the first as $y = x + d$ and the other as $d = x + y$, where d is a new node.

D. Construction of Boolean Expressions

If a valuated node $x = 1$ is written x and $x = 0$ is written $\overline{x}$, a valuation is a conjunction of literals, that is, a Boolean expression. Thus, as any Boolean expression, it may be reified as a single node.

Suppose the current valuation Val of the graph is reified as a node x such that $x = 1$ propagates Val. If a non valuated node y is then valuated, the whole valuation is reified as a node z such that if $y = 1$, $z = (x \wedge y)$ and if $y = 0$, $z = (x \wedge \overline{y})$ using a construction which preserves the fact that $z = 1$ propagates the whole valuation.

In this way, a valuated graph always contains a *source* node s such that $s = 1$ propagates the whole valuation. In a graph with a node u valuated to $true$ (the $universe$) Boolean expressions are then built using only two operators : *activation* (valuating a node to $true$) and *inhibition* (valuating a node to $false$).

All Boolean expressions of a given Boolean basis using $\wedge$ and $\neg$ may be constructed, with negation relative to u. The expression $x \vee y$, for instance, is constructed as $\neg(\overline{x} \wedge \overline{y})$. Since

conjunction and negation are a complete set of operators for Boolean algebra, all Boolean functions of the basis may be expressed.

Each expression is built in linear size if the construction of each gate is finite, and in a time which depends on the complexity of the propagation algorithm used. If no propagation is done, the time is also linear but no simplification or recognition of existing equivalent expressions is done. Multiple contradictory and equivalent nodes are produced. If a complete propagation is used (thus actually exponential in the worst case), then a *uniqueness principle* is maintained on the graph : two different nodes reify two different Boolean functions and no node is contradictory. The actual solution uses an incomplete propagation algorithm, described in [3]. It corresponds to the application of Boolean Constraint Propagation (BCP) [1] on the translation of partitions into propositional clauses. In addition to this propagation, the graph is normalized after each modification. This has the consequence that no clear algebraic characterization is available of which contradictory nodes are suppressed and which equivalent nodes are merged. It can only be illustrated how this normalisation accounts for commonsense intuition in a narrow but structured context.

IV. A Boolean Knowledge Base on Quadrilaterals

A. Taxonomies

The BKB of Figure 5 may be learnt by first accumulating "sort-of" assertions. The assertion "A rectangle is a sort of parallelogram" may be entered by creating two variables $rectangle$ and $parallelogram$ and connecting them with an implication, that is, a bipartition.

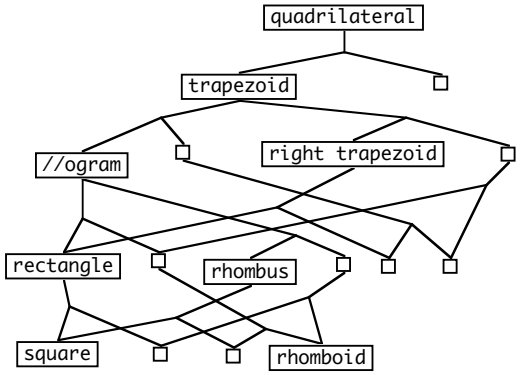

Fig. 5. A possible classification of common quadrilaterals.

If only implications are entered, the BKB will answer questions like "Is a rectangle a trapezoid ?" (activate $rectangle$ and read the value of $trapezoid$) or "What are the properties of a rhomboid ?" ($parallelogram$, $\neg rectangle$, $\neg rhombus$), but not "What is a parallelogram which is also a right trapezoid ?". Here, we need to enter a definition like $rectangle \leftrightarrow (parallelogram \wedge right\_trapezoid)$. This may be done in many different ways. We may enter the definition directly. We may also define a figure that is a $parallelogram$ and a $right\_trapezoid$ and not a $rectangle$ and then declare it contradictory, like in Figure 6, with x a $parallelogram$, y a $right\_trapezoid$ and z a $rectangle$. We may also define a non-rectangular parallelogram and a non-rectangular right trapezoid, then define their conjunction and declare it contradictory. Just as Boolean functions have many Boolean expressions, learning a Boolean definition may follow many different sequences. The elementary Boolean reasoning needed to keep the BKB close to that of Figure 5 in all such circumstances is afforded by the normalization mechanism presented in Subsection III-B.

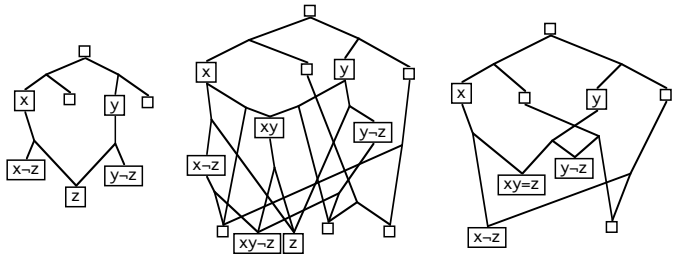

Fig. 6. The middle graph shows a construction of $x \wedge y$ from the left graph. The node xy is built with a finite construction and connected to z and the graph is normalized. Then, the node $xy\neg z$ is forced as a contradiction and suppressed, giving the right graph, where z and xy have been merged.

B. Abstract Properties

The BKB of Figure 5 may answer a question like "What is a rectangle which is also a rhombus ?" (a square) but not simpler ones like "Does a square have a right angle ?" or "Does a square have parallel sides ?". Intuitively, this requires "part-of" assertions in addition to the "sort-of" assertions but this may be done up to some extent at the Boolean level. We have to abstract the properties we want to distinguish, and redefine everything from this more general basis.

The BKB of Figure 7 starts from the basis $\{polygon,\ 4sides,\ 2\|sides,\ 2{\times}2\|sides,\ 2{=}sides,\ 4{=}sides,\ 1right\_angle, 2right\_angles\}$ of self documented properties. All properties but $polygon$ and $4sides$ are intended to mean "at least".

The implication constraints between these properties ($2{\times}2\|sides \rightarrow 2\|sides$, $4sides \rightarrow 2{=}sides$, $2right\_angles \rightarrow 1right\_angle$) are entered as bipartitions. Trapezoid, parallelogram, rectangle, etc, are defined as Boolean combinations of the properties :

$$trapezoid \leftrightarrow (4sides \wedge 2\|sides)$$
$$\|ogram \leftrightarrow (4sides \wedge 2{\times}2\|sides)$$
$$rectangle \leftrightarrow (\|ogram \wedge 1right\_angle)$$
$$right\_trapezoid \leftrightarrow (trapezoid \wedge 1right\_angle)$$
$$rhombus \leftrightarrow (\|ogram \wedge 4{=}sides)$$
$$square \leftrightarrow (rectangle \wedge rhombus)$$
$$rhomboid \leftrightarrow (\|ogram \wedge (\neg 1right\_angle \wedge \neg 4{=}sides))$$

The definition of a rectangle as a parallelogram with at least one right angle does not imply that it necessarily has

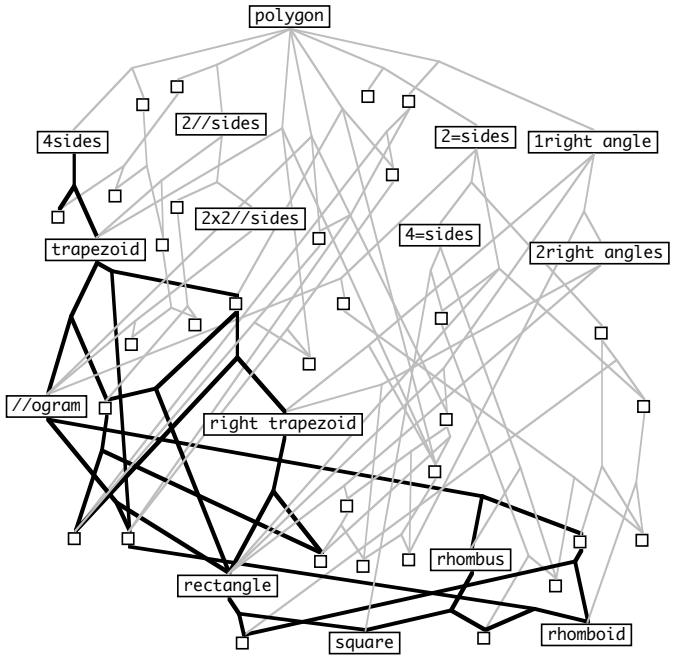

Fig. 7. A classification of quadrilaterals constructed from more general properties. The previous BKB still appears (highlighted with thick links) as a single set structure, where questions are answered simply by propagation.

four right angles and thus at least two. This reasoning needs a formalization of geometry outside the scope of Boolean algebra, but as before, we may simply add Boolean implications : $rectangle \rightarrow 2right\_angles$ and $rectangle \rightarrow 2{=}sides$.

Similarly, the question "Is there a trapezoid with exactly one right angle ?" is answered "yes" after these definitions are entered. To change with "no", a simple way is to build the figure ($trapezoid \ \wedge 1right\_angle \wedge \neg 2right\_angles$) and declare it contradictory.

The BKB of Figure 7 was built using the activation and inhibition operators, the forced contradiction operator, the variable elimination operator and the graph was normalized after each operation. Its two dimensional drawing needed much manual layout.

One characteristic is that it includes the graph of the BKB of Figure 5 as a subgraph, with minor differences, even though it was not explicitly constructed. This has the consequence that queries that would be answered by the BKB of Figure 5 will be answered by this new BKB with the same efficiency. There is no formal guaranty that for every learning sequence this set structure will be present. The general problem contains NP-complete and co-NP-complete subproblems. We can only verify experimentally that "reasonable" sequences such as those illustrated here, do lead to the expected set structure.

When it is queried, this BKB gives the illusion that concepts may be decomposed into parts. For instance, a rectangle has four sides, at least two right angles, two equal sides and two pairs of parallel sides. This illusion may be extended at will by adding more abstract properties to the initial basis. The limit here is the cost of Boolean encoding.

As the basis is generalized, concepts have more equivalent definitions. For instance,

$$square \leftrightarrow (rectangle \wedge rhombus)$$
$$square \leftrightarrow (rectangle \wedge 4{=}sides)$$
$$square \leftrightarrow (rhombus \wedge 1right\_angle)$$
$$square \leftrightarrow (rhombus \wedge right\_trapezoid)$$

The Boolean reasoning needed to verify those equivalences is simple but the problem of maintaining the knowledge base in a suitable form is not, because the equivalences must be spontaneously discovered rather than simply verified. Since the base is finite, it may be solved by testing all possible equivalences of a new node with all the nodes of the base. Such a solution is practical with small improvements, using a modern complete SAT solver, but on relatively small graphs. Moreover, it gives no hint as to how to organize the graph after each new construction. The solution adopted here is to let the normalization mechanism discover some equivalences and contradictions. Whatever definition of $square$ is entered first, the others will automatically be discovered as equivalent by mere propagation or normalization. If some equivalence is outside the scope of normalization, then it will eventually be detected later as the base exhibits more set structures through subsequent normalizations in response to user interactions. The recognition of already existing concepts is thus incomplete but incremental and never unreachable.

C. Instantiation of First Order Rules

As facts such as $\neg(trapezoid \ \wedge \ 1right\_angle \ \wedge \ \neg 2right\_angles)$ are entered, the BKB seems to possess deeper knowledge about the geometry of Euclidean quadrilaterals but these facts have a limited power of generalization. This knowledge looks like the apparently sharp but fatuitous and superficial knowledge we start with as beginners, without the insight of experience. Despite the generality of its basis and the amount of constraints concerning quadrilaterals, the BKB of Figure 7 will never deduce something as simple as the fact that no triangle has two parallel sides. If $3sides$ is added to the basis, $\neg(3sides \wedge 2 \parallel sides)$ will have to be added as a new constraint.

To overcome the need for this indefinite addition of constraints, a first-order axiomatisation of parallelism, intersection and perpendicularity seems unavoidable. However, it turns out that such an axiomatisation may be grounded as a set of small interconnected graphs like the graph of Figure 8.

In the graph of Figure 8, the facts $a \parallel b$, $b \parallel c$ and $a \parallel c$ are connected in such a way that the first-order constraint $\forall xyz \ ((x \parallel y) \wedge (y \parallel z)) \rightarrow (x \parallel z)$ applies on the set $\{a, b, c\}$. The graph is an efficient grounding of the first-order rule. Whenever two of the facts are true, the third is true by mere propagation (this may be verified by applying BCP on the Boolean translation of the partitions), accounting for transitivity of the $\parallel$ relation. Its symmetry is accounted for by considering that $a \parallel b$ and $b \parallel a$ are the same node. Reflexivity is ignored.

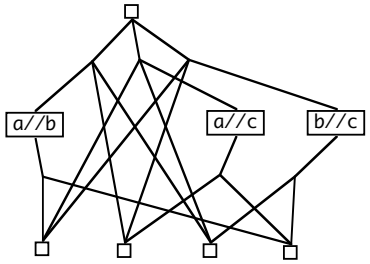

Fig. 8. Symmetry and transitivity of parallelism.

The graph may be directly built in the form shown in Figure 8, but a handy construction which is general for equivalence relations is the following : For each triple of elements, build the three instances $i1, i2, i3$ of the relation. Reify with a linear construction the Boolean function which is true iff two of $i1, i2, i3$ are true and the other false and declare it contadictory. The resulting normalized graph is the graph of Figure 8.

For a geometric figure involving n lines, such a graph will be needed for each of the $(n(n-1)(n-2))/6$ triples of lines. Next, their basis of facts must be connected to the corresponding abstract properties of the BKB of Figure 7. The property $2\|sides$ is no longer a postulate. It is defined as the disjunction of the six possible $\|$ relations between four lines a, b, c, d used to build a quadrilateral, without deciding in advance which ones intersect. Similarly, $2\times2\|sides$ is the disjunction $((a\,\|\,b) \wedge (c\,\|\,d)) \vee ((a\,\|\,c) \wedge (b\,\|\,d)) \vee ((a\,\|\,d) \wedge (b\,\|\,c))$.

A similar grounding into all non-equivalent instances may also be produced for the first-order rule connecting parallelism and perpendicularity : $\forall xyz\ (((x \perp y)\ \wedge\ (y \perp z)) \rightarrow (x\,\|\,z)$. The $\perp$ relation instances of this grounding now define the formerly postulated properties $1right\_angle$ and $2right\_angles$ in an obvious way. Corresponding instances of $\|$ and $\perp$ must also be made incompatible (using a bipartition).

It may be noted that in this formalization, lines and sides are not distinguished. The equivalence relation $=$ between sides may be grounded in the same way, defining the properties $2{=}sides$ and $4{=}sides$.

At this stage, if the constraint $\neg(trapezoid \wedge 1right\_angle \wedge \neg 2right\_angles)$ has not been added to the base, the BKB still admits the concept $trapezoid \wedge 1right\_angle \wedge \neg 2right\_angles$. A satisfying model is the trapezoid of Figure 9, where two opposite non parallel sides cross perpendicularly out of the figure.

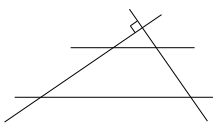

Fig. 9. A trapezoid with exactly one right angle.

To exclude this concept without explicitly declaring it contradictory, we may define the intersection relation between lines. Each instance of the $\|$ relation has as relative complement the corresponding instance of the intersection relation and the $\perp$ instance implies the intersection instance (Figure 10), this being produced for each instance.

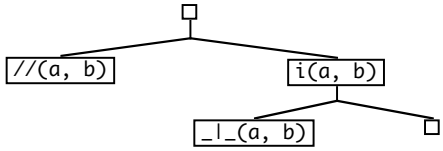

Fig. 10. Three instances of relations $\|$, intersection (i) and $\perp$ for a given pair of lines.

A triangle may then be defined as three intersecting lines, excluding any $\|$ relation between its sides. A quadrilateral is a combination of four lines a, b, c, d such that the pairs $\{a, b\}, \{b, c\}, \{c, d\}, \{a, d\}$ intersect as the vertices. The definitions of the abstract properties concerning $\|$ and $\perp$ may be simplified. We may now require that the $1right\_angle$ property concerns only the pairs of lines declared intersecting in the definition of a quadrilateral, excluding the case of Figure 9.

D. Discussion

The construction of Figure 9 is clearly a subtle trick. It appears because of an axiomatization of quadrilaterals with some shortcomings. It suggests that such incomplete, partial, provisional Boolean axiomatizations, far from being undesirable, are at the root of some form of insight, a capacity of "cheating" intelligently.

The BKB obtained at each stage is partial. It knows some facts and ignores others but it may permanently be taught with a simple user interaction. The BKB of Figure 7, for instance, still ignores that a quadrilateral with four equal sides is a rhombus. We may tell it directly by constructing the concept of a quadrilateral with four equal sides which is not a rhombus and declaring it contradictory. After suppression of the node and normalization, the BKB will keep the relative simplicity of Figure 7. This is the way geometry is first taught. We may then less simply instantiate some first-order theorems about circles and angles, to make more such geometrical facts deducible.

At the end of this, we get a BKB which may not be displayed but which may answer a large class of questions, those expressible as Boolean expressions of its postulated properties. Questions such as :
"Can a non-square rectangle be a rhombus ?",
"Can a non-rectangular parallelogram have a right angle ?",
"Can a non-square right trapezoid be a rhombus ?",
"Can a non-rectangular right trapezoid be a parallelogram or have four equal sides ?",
once translated in Boolean expression, will be answered (negatively) by mere propagation. Using an actual complete SAT solver on the translation of the BKB into propositional clauses,

any question expressed as a Boolean expression of some nodes of the BKB may be answered.

In addition, a navigation service is available, through the use of "which", "what is" and "what are the properties of" questions, which browse the intension and extension of nodes : "what are the properties of a rhomboid ?", "what are the properties of a rhombus which is not a square ?", "Which quadrilaterals have a right angle ?", "Which trapezoids have at least two equal sides ?", "what is a parallelogram which is also a right trapezoid ?", etc. Unlike decision problems, these questions may not have a canonical answer. The quality of the answer depends on the taxonomies which appeared spontaneously through basic automatic reasonings, such as the taxonomy of quadrilaterals spontaneously built in the BKB of Figure 7.

Querying a BKB and entering Boolean definitions and simple constraints is done with the activation, inhibition and forced contradiction operators. Here, everything may be considered as empirically learned through an interaction with a user. The only a-priori knowledge is Boolean algebra, mechanized under the form of graphs of partitions and their machinery.

Another building operation consists in grounding first-order rules. The BKB architecture is used as a library and complete freedom is left to the programmer for writing code that instantiates the first-order rules. It is an open problem how this grounding could be separated into a-priori knowledge extending the architecture and knowledge that may be taught by an interaction.

A recurring difficulty is that instantiating first-order rules into Boolean expressions, which should avoid combinatorial explosions, is an algebraic problem which involves arbitrary reasoning sequences. Furthermore, the classical limitations of first-order expression instantiation apply here. Many applications are out of reach because they may not be easily grounded at the Boolean level.

The architecture consists of a programmable library and an interactive graphical editor (Figures were built using the editor). It is written in C++ under the Carbon and Cocoa environments on MAC OS X.

V. Situation Relative to Other Trends

The most explicit use of graphs of partitions dates back to [7]. Lenhart Schubert used graphs of partitions as a data structure to answer part-of and sort-of questions most efficiently. Computational logic has developed mainly around clause-based formalisms. The present approach uses partitions as a logical formalism rather than a data structure, as an alternative to propositional clauses.

The following subsections are restricted to technologies specialized in propositional knowledge. This excludes higher order systems of representation and reasoning, like Description Logics, Semantic Networks or Conceptual Graphs.

A. SAT Solving

Problem solving is not the main functionality of the architecture proposed here for the management of Boolean knowledge. It includes a complete satisfiability test and a complete propagation operator, based on the complete satisfiability test, which valuates all the nodes implied by the current valuation, but they are used as extra tools. The foundation tools are the incomplete propagations, simplifications, normalization, the Boolean operators (activation, inhibition, forced contradiction) and the possibility to use the architecture as a programmable library.

Because SAT solving has been made so efficient on propositional clauses, the best actual solution for SAT testing on valuated graphs of partitions is to translate the valuated graph into propositional clauses and use any modern SAT solver, such as zchaff [6] or minisat2 [5].

For problem solving, graphs of partitions may be used as an interactive specification formalism. Problems are specified through user interaction and external programs that directly produce graphs as non-learnable constraints (for instance grounding of first-order expressions). The specification is then translated into cnfs and submitted to a SAT solver.

The merging of equivalent nodes and suppression of contradictory nodes is, as experiments show, a tiny advantage on average graphs for performing SAT tests. The exploitation of linear systems implied by graphs of partitions with Gaussian elimination in [3] showed huge improvements because DPLL-based solvers do not behave well on linear relations. Here, however, the computation times of zchaff and minisat2 before and after normalization of a graph show no important difference. One better motivation for normalization is navigation. Normalization is the mechanism that progressively builds set structures. These set structures are dynamic intermingled taxonomies of concepts. As illustrated in this article, set structures are an adequate feature for answering "what" and "which" questions. As a detail, the fact that four different definitions of a square are merged into a single node avoids that four squares appear in the answer of a "which" question.

B. Automatic Extraction of Logical Gates and Instance Preprocessing

The automatic extraction of logical gates from propositional clauses [9] is supposed to simplify a formula for accelerating the SAT test. The extraction time is thus deliberately limited. It must be lower than the time spared for the solving phase. Logical dependencies in graphs of partitions may be organized as Boolean graphs, where each node is a Boolean expression of a set of nodes called the basis. Normalization may be seen as a basic tool for extracting logical gates. No restriction on computation times is needed here since normalizations are useful for all subsequent reasonings within the BKB.

The preprocessing of problem instances [4] produces instances that are not equivalent to the original formula, but only equisatisfiable. Here again, the objective is to spend less time than is gained for the solving phase. In contrast, normalization preserves equivalence with the original base. It is not a simplification process that seeks to accelerate the solving of a single instance, but all subsequent reasonings. Thus, even high normalization times may be useful.

C. Knowledge Compilation

One objective of Knowledge Compilation is to structure knowledge so that testing for contradiction and deduction be polynomially bounded. In this objective, it imposes specialized formats to Boolean knowledge, such as NNF, DNNF, BDD, FBDD, etc [2].

If graphs of partitions are kept in a normal form, no particular format is imposed on the reified Boolean expressions they contain. The advantage for problem specification may be illustrated with the encoding of Constraint Satisfaction Problems (CSP) with finite domain variables. The variables are easily encoded as set structures representing their domain, with arbitrary clustering. Then, the relation constraints on tuples of variables are represented using Boolean conjunction on free copies of domains, as cartesian product. Arbitrary Boolean expressions then offer complete freedom for mixing prohibited and allowed values for tuples of variables.

D. Formal Concept Analysis

In the domain of Formal Concept Analysis (FCA), and since graphs of partitions allow arbitrary Boolean expressions, the Hasse diagram of a relation on finite sets may be built from the cartesian product represented as the conjunction of two independent hierarchies. It would be interesting to study what role normalization could play in the incremental construction of the Galois lattice.

E. Characteristic Vectors

Deduction locally to a set structure could be most efficiently implemented with bit vectors and Boolean operations on these vectors. This would require to explicitly delimit set structures as they dynamically appear. However, a graph may contain an exponential number of different crossing set structures and using vectors looks tedious. The normalization does the same, less efficiently, but without the need to isolate set structures in advance.

F. Possible Direct Applications

The BKB illustrated here are neither an axiomatization of Euclidean plane geometry for proving subtle theorems nor a system for computer assisted teaching of geometry. They are a generalistic architecture for learning and reasoning with Boolean knowledge. It could be used as a library for managing type, sort and part-of virtual lattices for artificial or natural language understanding. Another direction is the conception of graphical interactive *knowledge browsers*, which would usefully supplement, for structured informations, the traditional tree-structured file browsers associated with keyword search (such as Column View and Spotlight in Mac OS X).

VI. Conclusion

This article has presented and illustrated an architecture for managing Boolean knowledge. Even though it includes operators for a complete satisfiability test and complete propagation of Boolean values, its main motivation is not problem solving but to allow for easy specification, knowledge reorganization and navigation.

Boolean concepts and simple constraints are learnt via a simple user interaction, using only three operators : activation, inhibition and forced contradiction. Complex constraints, like instantiations of first-order expressions, are programmatically built in the same way as decision problems are specified with propositional clauses.

Thanks to a flexible normalization mechanism, and because no rigid format is imposed on expressions, the architecture provides enhanced freedom for reorganizing knowledge, such as abstracting a new basis of properties or constraining previously free generators.

Normalization spontaneously organizes intermingled taxonomies of concepts. Navigation explores the extension and intension of concepts in these dynamic taxonomies.

Two direct applications are the management of type, sort and part-of virtual lattices and knowledge browsers for structured information. The most interesting long term objective is the study of systematic ways of tractably instantiating first-order expressions into Boolean expressions, using graphs of partitions instead of propositional clauses. This would crucially enlarge the specification possibilities of the architecture.

Acknowledgment

The author would like to thank Professor M. J. Lebon for her generous attention and fruitful discussions.

References

[1] Apt K. (2000). Some remarks on boolean constraint propagation. In *Lecture Notes in Computer Science*, p. 91–107: Springer-Verlag.

[2] Darwiche A. & Marquis P (2002). A Knowledge Compilation Map. In *Journal of Artificial Intelligence Research 17*, p. 229–264.

[3] Goossens D. (2007). Spontaneous simplifications in graphs of partitions. In F. Fages, F. Rossi & S. Soliman, Eds., *Proceedings of CSCLP 2007, Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming*, p. 17–31: INRIA.

[4] Lynce I. & Marques-Silva J. (2001). The puzzling role of simplification in propositional satisfiability. In *EPIA'01 Workshop on Constraint Satisfaction and Operational Research Techniques for Problem Solving (EPIA-CSOR)*.

[5] Een Niklas & Biere Armin (2005). Effective preprocessing in sat through variable and clause elimination. In *proc. SAT 05, volume 3569 of LNCS, Springer*, p. 61–75.

[6] Moskewicz M. W., Madigan C. F., Zhao Y., Zhang L. & Malik S. (2001). Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*.

[7] Papalaskaris M. & Schubert L. (1981). Parts Inference: Closed and Semi-closed Partitioning graphs. In *Proceedings of the 7th IJCAI*, p. 304–309, Vancouver.

[8] Subbarayan S. & Pradhan D. (2004). Niver: Non increasing variable elimination resolution for preprocessing sat instances. *In Selected Revised Papers of International Conference on Theory and Applications of Satisfiability Testing Conference (Springer LNCS)*, p. 276–291.

[9] Grégoire E. & Ostrowski R. & Mazure B. & Sais L. (2004) Automatic Extraction of Functional Dependencies. *In SAT 2004 - 7th International Conference on Theory and Applications of Satisfiability Testing.* http://www.satisfiability.org/SAT04/programme/64.pdf