# 1 Introduction

Constraint satisfaction problems (CSPs) arise in a wide range of automated reasoning tasks including design, diagnosis, truth maintenance, logic programming, spatio-temporal reasoning, and a variety of combinatorial optimisation tasks including timetabling and scheduling. Because of their importance, CSPs and methods for solving them have been studied extensively. Also extensively studied is the resolution rule of inference, which can be used to decide the satisfiability of a set of propositional clauses, a problem that specialises the CSP.

This paper reports on initial explorations into the relationship between resolution and the concepts developed in the CSP literature. The key observation, on which the results of this paper are built, is that semantic trees, used in the study of resolution, and backtrack search spaces, used in the study of constraint satisfaction, are the same objects.

This paper presents three main results. Section 3 proves that if a set of propositional clauses is closed under the usual resolution inference rule then the search space for the corresponding constraint satisfaction problem is backtrack free for *every* variable ordering. This result is useful for Boolean problems—those in which the variables can take on values from a domain of size two. However, many naturally-arising problems are non-Boolean.

Section 4 generalises the results of Section 3 to the non-Boolean case. Non-Boolean clauses (NB-clauses), which contain variables ranging over domains of arbitrary finite size, are introduced as is NB-resolution, an inference rule for NB-clauses. The main result of this section is a proof that if a set of NB-clauses is closed under NB-resolution then the search space for the corresponding constraint satisfaction problem is backtrack free for *every* variable ordering.

The results of Sections 3 and 4 show how one can remove backtracking from all search spaces, regardless of variable ordering. However, one would normally just need to eliminate backtracking from the search space that is to be searched—that is, for a particular variable ordering. Section 5 introduces a severely-restricted version of NB-resolution, called ordered NB-resolution, and proves that for any given variable ordering, it eliminates all backtracking from the search space generated by that variable ordering.

These three results on the elimination of backtracking generalise the usual refutation completeness results by telling us what happens when *any* set of clauses—satisfiable or not—is closed under resolution. Refutation completeness, which states that the empty clause can be derived from any unsatisfiable set of clauses, follows as a special case of each of the three general results.

# 2 The Constraint Satisfaction Problem

An instance of the *constraint satisfaction problem* consists of an enumerable set of variables, $X_1, X_2, \ldots$, having domains $D_1, D_2, \ldots$, and a set of constraints. Roughly speaking, the problem is to decide if the variables can be instantiated with values from their domains without violating the constraints. Each constraint stipulates that a certain tuple of variables must take values from a certain set of tuples of values. In particular each constraint has an arity $1 \leq k \leq n$ and a $k$-ary constraint among variables $X_{i_1}, \ldots, X_{i_k}$ is a subset of $D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k}$.

A *total instantiation* maps each variable to an element in its domain. That is, it is a total function that maps each $X_i$ to an element of $D_i$. A

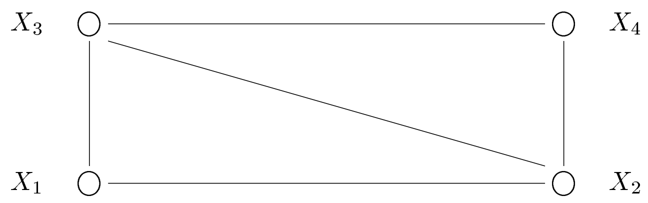


Figure 1: Graph Colouring: The problem is to colour each of the four nodes of this graph with the colours red, blue and green so that no two adjacent nodes have the same colour and $X_1$ is red.

total instantiation $A$ *satisfies* a constraint among variables $X_{i_1}, \ldots, X_{i_k}$ if and only if $\langle A(X_{i_1}), \ldots, A(X_{i_k})\rangle$ is a member of the constraint. A *solution* to an instance of CSP is a total instantiation that satisfies all the constraints. Given an instance of CSP the goal is usually either to determine whether the instance has any solutions, to find any solution, or to find all solutions. Determining whether any given instance has a solution is an NP-complete problem since CSP is a generalisation of both propositional satisfiability and graph colouring.

An instance of the CSP is said to be finite if it has a finite set of variables and a finite set of constraints; otherwise it is said to be infinite. Though the usual statement of the CSP admits only finite instances, this paper also admits infinite instances as these are covered by the main results of this paper. The possibility of infinite instances could arise if one had a constraint language that allowed schemas which could stand for an infinite set of constraints. For example, if one considers non-ground clauses as a constraint language, then the Herbrand Theorem tells us that each non-ground constraint can be treated as the possibly infinite set consisting of its ground instances. Thus, for example, the use of semantic trees in the study of resolution inference covers cases in which there is an infinite set of clauses. As this paper generalises this notion of semantic trees, infinite instances of the CSP are considered.

As an example, the graph colouring problem of Figure 1 can be formalised as a finite instance of CSP. As a CSP instance, this problem has four variables—$X_1$, $X_2$, $X_3$ and $X_4$—each having a domain consisting of the three colours:

$$D_1 = D_2 = D_3 = D_4 = \{r, g, b\}.$$

The problem instance has five binary constraints—one for each arc in the graph—and one unary constraint forcing $X_1$ to be red.

$$\begin{array}{l}
\langle X_1, X_2\rangle \in \{\langle r,b\rangle, \langle r,g\rangle, \langle b,r\rangle, \langle b,g\rangle, \langle g,r\rangle, \langle g,b\rangle\} \\
\langle X_1, X_3\rangle \in \{\langle r,b\rangle, \langle r,g\rangle, \langle b,r\rangle, \langle b,g\rangle, \langle g,r\rangle, \langle g,b\rangle\} \\
\langle X_2, X_3\rangle \in \{\langle r,b\rangle, \langle r,g\rangle, \langle b,r\rangle, \langle b,g\rangle, \langle g,r\rangle, \langle g,b\rangle\} \\
\langle X_2, X_4\rangle \in \{\langle r,b\rangle, \langle r,g\rangle, \langle b,r\rangle, \langle b,g\rangle, \langle g,r\rangle, \langle g,b\rangle\} \\
\langle X_3, X_4\rangle \in \{\langle r,b\rangle, \langle r,g\rangle, \langle b,r\rangle, \langle b,g\rangle, \langle g,r\rangle, \langle g,b\rangle\} \\
\langle X_1\rangle \in \{\langle r\rangle\}
\end{array}$$

This instance has exactly two solutions:

$$\begin{array}{l}
A_1 = \{X_1 \mapsto r, X_2 \mapsto b, X_3 \mapsto g, X_4 \mapsto r\} \\
A_2 = \{X_1 \mapsto r, X_2 \mapsto g, X_3 \mapsto b, X_4 \mapsto r\}
\end{array}$$

The simplest method for solving an instance of the CSP is known as *backtracking*. Starting with an empty instantiation this method repeatedly

Input: A CSP instance and
    a non-repeating enumeration $\mathbf{X} = X_1, X_2, \ldots$ of its variables.
Output: Failure or a solution to the input CSP.

(1) Let $A$ be the totally undefined partial instantiation.
(2) If $A$ is locally inconsistent then FAIL.
(3) For each $X_i$ of $\mathbf{X}$ in order do
(4)   Choose an element from the domain of $X_i$. Call it $d$.
(5)   Minimally extend $A$ so that $A(X_i)$ is $d$.
(6)   If $A$ is locally inconsistent then FAIL.
(7)   od
(8) SUCCEED and return $A$.

Figure 2: The Backtrack Algorithm.

takes a partial instantiation and further instantiates it in various ways. If it is obvious that a partial instantiation cannot be extended to a solution, then it is discarded and a different one is tried. This is repeated until a solution is found. To make this description more exact we must define what a partial instantiation is and when it is obvious that it cannot be extended to a solution.

A *partial instantiation* is the same as a total instantiation except that it may be a partial function over the variables. For our graph colouring example $A_3 = \{X_1 \mapsto r, X_4 \mapsto r\}$ and $\emptyset$ are partial instantiations but not total instantiations. Since total instantiations are also partial instantiations, we shall often use the word *instantiation* to refer to a partial instantiation. One partial instantiation $A'$ *extends* another $A$ if and only if for every variable $X$ such that $A(X)$ is defined, $A'(X)$ is also defined and is equal to $A(X)$. Notice that every partial instantiation extends both itself and the empty instantiation. In the colouring example, both $A_1$ and $A_2$ extend $A_3$. A *total extension* of $A$ is a total instantiation that extends $A$.

Partial instantiation $A$ is *locally consistent* if each constraint is satisfied by some total extension of $A$. $A$ is *globally consistent* if it has a total extension that satisfies all the constraints, that is, if it is extended by some solution. Notice that global consistency implies local consistency. In the colouring example $A_3$ is globally consistent and $\{X_4 \mapsto b\}$ is locally, but not globally, consistent. Neither $\{X_1 \mapsto b\}$ nor $\{X_2 \mapsto b, X_3 \mapsto b\}$ are locally consistent. Notice that a total instantiation is globally consistent if and only if it is locally consistent, if and only if it is a solution. Local consistency is easy to test, but deciding global consistency is NP-complete since the CSP itself is the problem of deciding whether the empty instantiation is globally consistent.

Using this terminology, the Backtrack Algorithm is shown in Figure 2. The choice made in line (4) is don't-know non-determinism; the algorithm succeeds if there is a choice that leads to success. This can be implemented deterministically by backtracking over the set of choices in a search for one that succeeds.

The backtrack algorithm can be seen as moving through a search space of choices. The search space for any non-repeating enumeration $\mathbf{X} = X_1, X_2, \ldots$ is a tree that consists of at least one node, the root, whose depth is 0. The nodes at depth $k$ in the tree are partial instantiations that assign values to variables $X_1, \ldots X_k$. To define the edges of the tree, let $A$ be a partial instantiation at depth $k$ in the search space and consider three

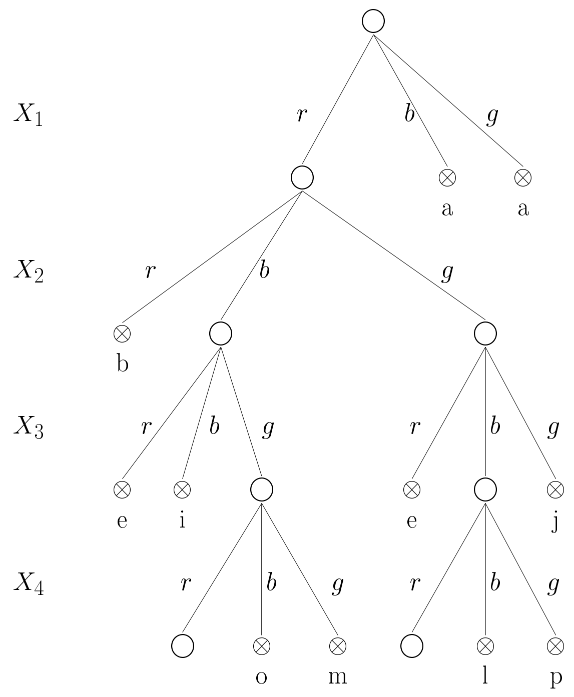


Figure 3: Search Sapce 1.

cases:

1. $A$ is locally inconsistent: Then $A$ is called a *failure node* and has no children.
2. $A$ is locally consistent and $\mathbf{X}$ contains only $k$ variables: Then $A$ is a solution to the problem and contains no children.
3. $A$ is locally consistent and $\mathbf{X}$ contains more than $k$ variables: Then, for each value $d$ in the domain of $X_{k+1}$ $A$ has a child that is formed by minimally extending $A$ so that it maps $X_{k+1}$ to $d$.

Figures 3 and 4 give two search spaces for the graph colouring problem of Figure 1. In these figures, $\otimes$ indicates a failure node. All other nodes are locally consistent and all locally-consistent leaf nodes are solutions. Search Space 1 uses enumeration $X_1, X_2, X_3, X_4$ and Search Space 2 uses enumeration $X_1, X_4, X_2, X_3$. This illustrates that the size of the search space can differ greatly according to which enumeration is used. Much research has addressed the question of how to chose enumerations that result in smaller search spaces. It is possible, and sometimes desirable, to use different enumerations down different paths of the search space; however this paper only considers search spaces in which all paths use the same enumeration.

Notice that every solution to a CSP appears in the search space as a complete path that starts at the root and is either infinite (in the case of an infinite set of variables) or terminates at a locally consistent instantiation (in the case of a finite set of variables). Since all infinite paths are solutions, the search space for an unsatisfiable CSP is finite. That is, as one would expect, the usual compactness result holds for CSPs.

A *backtrack node* in a search space is a locally-consistent node that is not globally consistent. Thus a backtrack node is not on the path to any

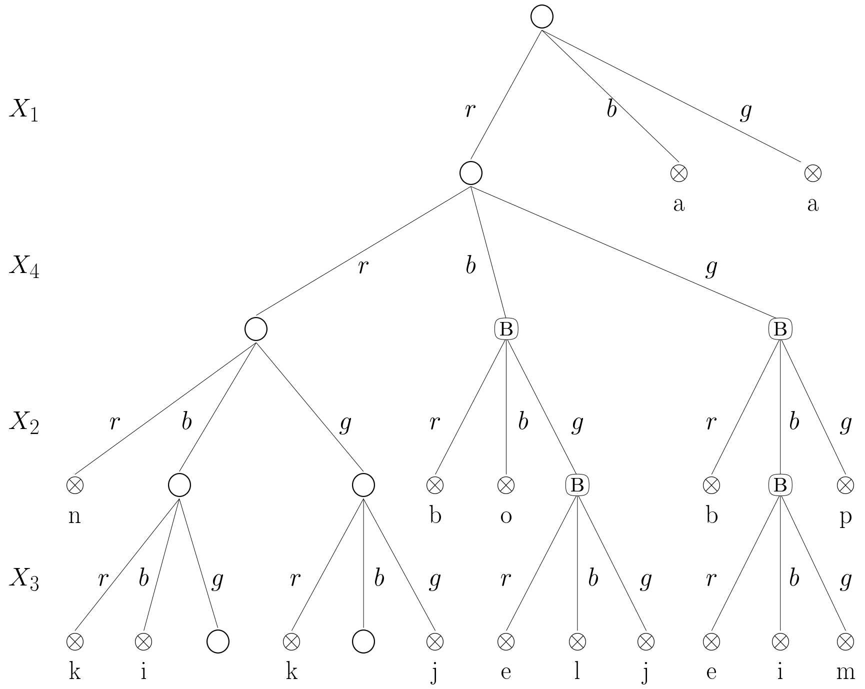


Figure 4: Search Sapce 2.

solution, but it is not pruned. A search space is said to be *backtrack-free* if it contains no backtrack nodes. Notice that Search Space 1 is backtrack free but that Search Space 2 contains four backtrack nodes, each labelled with a "B" inside the node. This illustrates the important fact that for a given instance of the CSP some enumerations may yield backtrack-free search spaces while other enumerations may not.

We conclude this section by noting a few general properties of these backtrack search spaces.

**Proposition 1** *A node with children is globally consistent if and only if at least one of its children is globally consistent. A node with children is locally consistent if at least one of its children is locally consistent. A node without children is locally consistent if and only if it is globally consistent.*

# 3 Resolution

The clausal satisfiability problem is to determine if any given set of clauses of propositional logic is satisfiable. A clause is a finite disjunction of zero or more literals, and a literal is a propositional variable or its negation. The clause that contains zero disjuncts is called the empty clause and is considered to be unsatisfiable. As an example, one instance of the clausal satisfiability problem is to determine if the following set of four clauses is satisfiable.

$$\neg X_1 \vee X_2 \tag{1}$$
$$X_4 \vee X_1 \tag{2}$$
$$\neg X_4 \vee X_1 \tag{3}$$

$$X_2 \vee \neg X_3 \tag{4}$$

The usual statement of the clausal satisfiability problem involves only a finite set of clauses but, as with the CSP, we shall allow instances that have an infinite set of clauses and an infinite, but enumerable, set of variables.

The resolution rule of inference is:

$$\frac{\begin{array}{c} X_i \vee \psi_1 \\ \neg X_i \vee \psi_2 \end{array}}{\psi_1 \vee \psi_2}$$

As an example, using the clauses above, resolution can perform the inference

$$\frac{\begin{array}{ll} (2) & X_4 \vee X_1 \\ (3) & \neg X_4 \vee X_1 \end{array}}{X_1} \tag{5}$$

In order to analyse resolution and related rules of inference, the idea of a *semantic tree* was introduced by Robinson [1968] and subsequently generalised and further exploited by Kowalski and Hayes [1969]. A semantic tree is a survey of all models of a set of clauses, $C$. Its construction is governed by a non-repeating enumeration of the propositional variables of $C$. Figure 5 shows a semantic tree for clauses (1) – (4) using enumeration $X_1, X_2, X_3, X_4$. The root node represents the set of all models of the clause set. This node has two descendants; one which represents all models that assign *TRUE* to $X_1$ and one which represents all models that assign *FALSE* to $X_1$. At each depth $k$ in the search tree this happens for the $k+1$ variable in the enumeration; the set of models at each node is divided into those that assign *TRUE* to the specified variable and those that assign it *FALSE*. Thus each node is associated with a set of models; the set is infinite if and only if the problem contains an infinite set of variables. If a clause in $C$ is falsified by every model associated with a certain node, then we say that the node *violates* the clause and we call the node a *failure node*. In constructing a semantic tree a failure node is not given any descendants and is labelled by a clause that it violates. The semantic tree of Figure 5 contains eight failure nodes (again indicated by $\otimes$) and each is labelled by the number of a clause that it violates.

A node in a semantic tree is an *inference node* if and only if it has children and each of them is a failure node. The semantic tree of Figure 5 contains three inference nodes, each indicated by an "I" inside the node.

Semantic trees are very useful because of an important relationship between them and resolution, which was first observed by Robinson [1968].

**Theorem 2 (Robinson, 1968)** *Let $T_C$ be a semantic tree for a set $C$ of propositional clauses. If $N$ is an inference node in $T_C$ and $c_1$ and $c_2$ label its two descendants then $c_1$ and $c_2$ are resolvable and their resolvent is a clause that is violated by $N$ (and perhaps some of its ancestors).*

To illustrate this, observe that the three inference nodes of Figure 5 each have two children labelled by clauses (2) and (3). The resolvent of (2) and (3), as shown in (5), is $X_1$. This resolvent is violated at node $\{X_1 \mapsto f\}$, which is an ancestor of all three inference nodes.

As a contrapositive to Theorem 2 we have the following corollary:

**Corollary 3** *If $C$ is closed under resolution then no semantic tree for $C$ contains an inference node.*

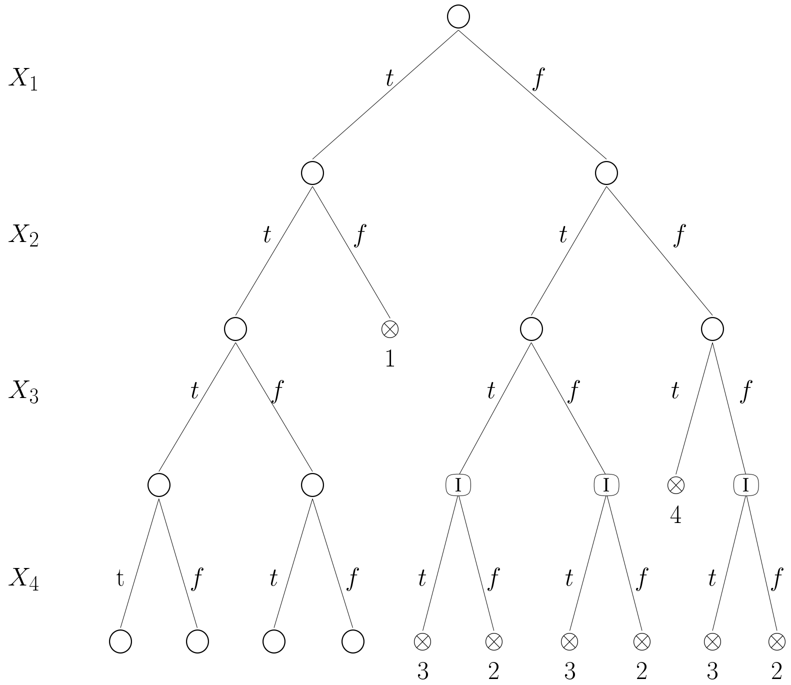


Figure 5: A semantic tree for clauses (1) – (4).

Notice that it may not be possible to compute the closure of a set of clauses in a finite number of resolution steps. It can, however, be done for any set of clauses that is finite or unsatisfiable.

A complete path in a semantic tree is either a path from the root of the tree to a leaf or an infinite path from the root. The models that satisfy the clausal satisfiability problem are precisely the complete paths that do not contain a failure node.

A common observation is that the clausal satisfiability problem is a special case of the CSP in which each variable has the Boolean domain $\{TRUE, FALSE\}$. Every clause in a clausal satisfiability problem can be encoded as a single CSP constraint.

It is now a simple observation, though I know of no one who has previously pointed it out, that the semantic tree for a clausal satisfiability problem is identical to the backtrack search space for the corresponding CSP. This observation forms the basis for this paper's results, which tie the CSP to resolution. To obtain these results, we begin by relating terminology from these two areas.

**Lemma 4**

1. *Every inference node is a backtrack node.*
2. *A node in a semantic tree is a backtrack node if and only if it has children and each of them is either a backtrack node or a failure node.*
3. *A semantic tree contains a backtrack node if and only if it contains an inference node.*

*Proof:*

1. If $N$ is an inference node then it has children and therefore it must be locally consistent. But since its children are locally inconsistent they are not globally consistent, and therefore neither is $N$. Since $N$ is locally consistent but not globally consistent, it is a backtrack node.

2. ($\Longleftarrow$) Let $N$ be a node with children, each of which is a backtrack node or a failure node. Node $N$ is locally consistent since it has children, but it is globally inconsistent since none of its children are globally consistent. Hence, $N$ is a backtrack node.

   ($\Longrightarrow$) If $N$ is a backtrack node then, by definition, it has children. Since $N$ is globally inconsistent, each of its children are also globally inconsistent and therefore each must be a backtrack node or a failure node.

3. ($\Longleftarrow$) This follows since every inference node is a backtrack node.

   ($\Longrightarrow$) First observe that if a semantic tree contains a backtrack node then it contains one that has no descendants that are backtrack nodes. If this were not the case, then the tree would contain an infinite path that included backtrack nodes—which is impossible because infinite paths are solutions.

   Let $B$ be a backtrack node that has no descendants that are backtrack nodes. Then, by 2 above, all children of $B$ are failure nodes. Hence, by definition, $B$ is an inference node. □

Putting this all together, the nodes in a semantic tree can be partitioned into three main categories:

**locally and globally consistent:** these are precisely the nodes on a solution path;

**locally consistent but globally inconsistent:** these are precisely the backtrack nodes; inference nodes are a subset of this category; and

**locally and globally inconsistent:** these are precisely the failure nodes.

Since, according to Lemma 4, a semantic tree contains a backtrack node if and only if it contains an inference node, it immediately follows from Corollary 3 that:

**Theorem 5** *If $C$ is closed under resolution then every semantic tree for $C$ is backtrack-free.*

The usual statement of the completeness of resolution tells us what happens if we close under resolution a set of clauses that is unsatisfiable, but tells us nothing about what happens if the set of clauses is satisfiable. Theorem 5 is a more general result covering both the satisfiable and unsatisfiable cases.

Now consider the special case in which $C$ is unsatisfiable and closed under resolution. Then Theorem 5 tells us that the root of any semantic tree for $C$ is not a backtrack node. Since the root is globally inconsistent and not a backtrack node it must be a failure node. The root node represents the set of all models and the only clause that is falsified by all models is the empty clause. Hence the root must be labelled by the empty clause. Thus, we have the following corollary to Theorem 5.

**Corollary 6** *If $C$ is unsatisfiable and closed under resolution then it contains the empty clause.*

Since, as first shown by Robinson [1968], any semantic tree for any unsatisfiable set of clauses is finite, the set $C'$ of clauses labelling the tree's failure nodes is finite. Hence, it is possible to close $C'$ under resolution in a finite number of steps and, by Corollary 6, this closure contains the empty clause. Thus, we have the usual statement of the completeness of resolution: from any set of unsatisfiable clauses the empty clause can be derived by a finite number of applications of the resolution inference rule. So we see that this well-known result follows as a special case of Theorem 5.

# 4 NB-Resolution

Now that we know how standard resolution inference relates to Boolean CSPs, we turn our attention to the more general setting of non-Boolean CSPs, those in which the variables can have domains of arbitrary size. We do this by introducing a clausal form involving non-Boolean variables and a resolution inference rule that operates on such clauses.

As with non-Boolean CSPs, our non-Boolean clauses—or NB-clauses—involve variables whose domains have arbitrary size. An NB-clause is of the form

$$X_{i_1}/S_{i_1} \ \vee \ \cdots \ \vee \ X_{i_n}/S_{i_n}$$

where $X_{i_1}, \ldots X_{i_n}$ are distinct variables and each $S_{i_j}$ is a non-empty subset of the domain of $X_{i_j}$. An NB-clause may have no atoms; such a clause is called the empty clause and is unsatisfiable.

There is no need for negative literals in NB-clauses since a literal of the form $\neg X/S$ is logically equivalent to $X/S'$, where $S'$ is the set of elements in the domain of $X$ that are not in $S$. Nor is there any need to repeat a variable in an NB- clause since $X/S_1 \vee X/S_2$ is logically equivalent to $X/S$, where $S$ is the union of $S_1$ and $S_2$.

All constraints in a CSP can be encoded as a set of NB-clauses. As an example, Figure 6 gives a set of NB-clauses that encode the constraints of the colouring problem of Figure 1.

The resolution rule of inference, which operates on Boolean clauses, can be generalised to operate on non-Boolean clauses. This generalised inference rule, which we call NB-resolution, is

$$\frac{\begin{array}{c} X_i/S_1 \ \vee \ \psi_1 \\ X_i/S_2 \ \vee \ \psi_2 \\ \vdots \\ X_i/S_k \ \vee \ \psi_k \end{array}}{\psi_1 \ \vee \ \psi_2 \ \vee \ \cdots \ \vee \ \psi_k}$$

where the premises are a minimal set of NB-clauses such that $S_1 \cap \cdots \cap S_k = \emptyset$. We say that such a resolution is a resolution upon $X_i$ and we say that the conclusion is the resolvent of the premises upon $X_i$.

Here are four examples of NB-resolution inferences using the clauses of Figure 6. Inference (6) is upon $X_2$, inference (7) is upon $X_1$ and inferences (8) and (9) are upon $X_3$.

$$\frac{\begin{array}{ll} \text{(b)} & X_1/\{b,g\} \vee X_2/\{b,g\} \\ \text{(c)} & X_1/\{r,g\} \vee X_2/\{r,g\} \\ \text{(d)} & X_1/\{r,b\} \vee X_2/\{r,b\} \end{array}}{X_1/\{r,g,b\}} \tag{6}$$

| | | |
|---|---|---|
| (a) | $X_1/\{r\}$ | $X_1$ is red |
| (b) | $X_1/\{b,g\} \lor X_2/\{b,g\}$ | $X_1$ and $X_2$ are different |
| (c) | $X_1/\{r,g\} \lor X_2/\{r,g\}$ | |
| (d) | $X_1/\{r,b\} \lor X_2/\{r,b\}$ | |
| (e) | $X_1/\{b,g\} \lor X_3/\{b,g\}$ | $X_1$ and $X_3$ are different |
| (f) | $X_1/\{r,g\} \lor X_3/\{r,g\}$ | |
| (g) | $X_1/\{r,b\} \lor X_3/\{r,b\}$ | |
| (h) | $X_2/\{b,g\} \lor X_3/\{b,g\}$ | $X_2$ and $X_3$ are different |
| (i) | $X_2/\{r,g\} \lor X_3/\{r,g\}$ | |
| (j) | $X_2/\{r,b\} \lor X_3/\{r,b\}$ | |
| (k) | $X_3/\{b,g\} \lor X_4/\{b,g\}$ | $X_3$ and $X_4$ are different |
| (l) | $X_3/\{r,g\} \lor X_4/\{r,g\}$ | |
| (m) | $X_3/\{r,b\} \lor X_4/\{r,b\}$ | |
| (n) | $X_2/\{b,g\} \lor X_4/\{b,g\}$ | $X_2$ and $X_4$ are different |
| (o) | $X_2/\{r,g\} \lor X_4/\{r,g\}$ | |
| (p) | $X_2/\{r,b\} \lor X_4/\{r,b\}$ | |

Figure 6: A set of NB-clauses representing the constraints of the colouring problem of Figure 1.

$$\begin{array}{ll} \text{(a)} & X_1/\{r\} \\ \text{(b)} & X_1/\{b,g\} \lor X_2/\{b,g\} \\ \hline \text{(b')} & X_2/\{b,g\} \end{array} \tag{7}$$

$$\begin{array}{ll} \text{(e)} & X_1/\{b,g\} \lor X_3/\{b,g\} \\ \text{(l)} & X_3/\{r,g\} \lor X_4/\{r,g\} \\ \text{(j)} & X_2/\{r,b\} \lor X_3/\{r,b\} \\ \hline \text{(r)} & X_1/\{b,g\} \lor X_4/\{r,g\} \lor X_2/\{r,b\} \end{array} \tag{8}$$

$$\begin{array}{ll} \text{(e)} & X_1/\{b,g\} \lor X_3/\{b,g\} \\ \text{(i)} & X_2/\{r,g\} \lor X_3/\{r,g\} \\ \text{(m)} & X_3/\{r,b\} \lor X_4/\{r,b\} \\ \hline \text{(q)} & X_1/\{b,g\} \lor X_2/\{r,g\} \lor X_4/\{r,b\} \end{array} \tag{9}$$

The first of these examples illustrates a small, but important, aspect of NB-resolution. Strict application of the inference rule would derive $X_2/\{b,g\} \lor X_2/\{r,g\} \lor X_2/\{r,b\}$, which is not a well-formed NB-clause since it contains multiple occurrences of the variable $X_2$. However, the three literals can be collapsed, resulting in the logically-equivalent, but well-formed NB-clause $X_2/\{r,g,b\}$, which happens to be a tautology.

Notice that Boolean resolution is a special case of NB-resolution. For example, the clauses and inference from (5) can be expressed as NB-clauses:

$$\begin{array}{ll} \text{(2)} & X_4/\{\mathit{TRUE}\} \lor X_1/\{\mathit{TRUE}\} \\ \text{(3)} & X_4/\{\mathit{FALSE}\} \lor X_1/\{\mathit{TRUE}\} \\ \hline & X_1/\{\mathit{TRUE}\} \end{array}$$

Here $X_1$ and $X_4$ are variables that each have domain $\{\mathit{TRUE}, \mathit{FALSE}\}$.

The Boolean notion of semantic tree presented in the previous section also can be generalised to the non-Boolean case. In these NB-semantic trees, when a node is split upon a variable whose domain size is $n$, the node has $n$ children—one in which the variable takes on each value of the domain. Thus Search Spaces 1 and 2 of Figures 3 and 4 are NB-semantic trees. Search Space 1 contains no inference nodes and Search Space 2 contains two inference nodes: $N_1 = \{X_1 \mapsto r, X_4 \mapsto b, X_2 \mapsto g\}$ and $N_2 = \{X_1 \mapsto r, X_4 \mapsto g, X_2 \mapsto b\}$.

The lemmas, theorems and corollaries of the previous section also generalise to the non-Boolean case. Both the statement and proof of Lemma 4 carry over verbatim to the non-Boolean case. Theorems 2 and 5 and their corollaries (Corollaries 3 and 6, respectively) are now restated and proved in the more-general non-Boolean setting.

**Theorem 7** *Let $T_C$ be an NB-semantic tree for a set $C$ of NB-clauses. If $N$ is an inference node in $T_C$, variable $X$ labels the edges from $N$ and $c_1, \ldots, c_k$ label the descendants of $N$ then $c_1, \ldots, c_k$ are NB-resolvable upon $X$ and their NB-resolvent upon $X$ is an NB-clause that is violated by $N$ (and perhaps some of its ancestors).*

*Proof:* Let $N_1, \ldots, N_k$ be the immediate descendants of $N$, and let $c_i$ be the NB-clause that labels $N_i$. Let $X = d_i$ be the label on the arc from $N$ to $N_i$, and observe that $\{d_1, \ldots, d_k\}$ is the domain of $X$. Since each $c_i$ is not violated at $N$, the assignment of $d_i$ to $X$ must contribute to the violation of $c_i$ at $N_i$. For this to be the case each $c_i$ must be of the form $X/S_i \vee \psi_i$, where $S_i$ does not contain $d_i$. Hence, $\cap_{i=i}^{n} S_i = \emptyset$ and therefore $\psi_1 \vee \cdots \vee \psi_n$ is a resolvent of $\{c_i \mid 1 \leq i \leq n\}$ upon $X$. Since each $\psi_i$ is violated at $N_i$ and does not contain any occurrences of $X$, it must be violated at $N$. Therefore $\psi_1 \vee \cdots \vee \psi_n$ is violated at $N$. $\square$

As an example, inference node $N_1 = \{X_1 \mapsto r, X_4 \mapsto b, X_2 \mapsto g\}$ of Figure 4 has three descendants labelled by (e), (l), (j). These are resolvable, as shown in (8), and their resolvent, (r), is violated at node $N_1$. Similarly, inference node $N_2 = \{X_1 \mapsto r, X_4 \mapsto g, X_2 \mapsto b\}$ has decendants labelled (e), (i) and (m) and their resolvent, as shown in (9), is violated at $N_2$.

As a contrapositive to Theorem 7 we have the following corollary:

**Corollary 8** *If $C$ is closed under NB-resolution then no NB-semantic tree for $C$ contains an inference node.*

Since, according to Lemma 4, a semantic tree contains a backtrack node if and only if it contains an inference node, it immediately follows from Corollary 8 that:

**Theorem 9** *If $C$ is closed under NB-resolution then every NB-semantic tree for $C$ is backtrack-free.*

And, as in the last section, if $C$ is unsatisfiable we have the following corollary:

**Corollary 10** *If $C$ is unsatisfiable and closed under NB-resolution then it contains the empty clause.*

And from this, it again follows that if $C$ is unsatisfiable then the empty clause can be derived by a finite number of applications of the NB-resolution inference rule.

# 5 Ordered NB-Resolution

The previous section demonstrates that NB-resolution can be used to eliminate backtrack nodes from every NB-semantic tree for a given set of NB-clauses. For certain purposes this could be more than is needed; it could be that we need to eliminate backtrack nodes only from the NB-semantic tree (or search space) that we are about to search. In this section we shall assume that we are given a non-repeating enumeration of variables of a set $C$ of NB-clauses and we shall introduce a severely restricted form of NB-resolution, called ordered NB-resolution, that can be used to eliminate backtrack nodes from the NB-semantic tree for $C$ that uses the given enumeration.

Let $O$ be a non-repeating enumeration of all variables in the language. We say that variable $X$ is greater than variable $Y$ if $X$ appears after $Y$ in the enumeration. We say that an NB-semantic tree is $O$-ordered if down every path of the tree the variables are enumerated according to $O$. Notice that for any fixed set of NB-clauses and any fixed ordering $O$ there is only one $O$-ordered NB-semantic tree.

A resolution or NB-resolution inference is said to be $O$-ordered if the variable resolved upon is greater than every other variable that occurs in the premises. For example, if $O$ is $X_1, X_2, X_3, X_4$, then inference (6) is $O$-ordered, but inferences (7), (8) and (9) are not. On the other hand, if is $O$ is $X_1, X_4, X_2, X_3$ then inferences (6), (8) and (9) are $O$-ordered but (7) is not. Notice that for any set of NB-clauses and any ordering $O$ there is at most one $O$-ordered NB-resolution that uses all the NB-clauses as premises.

**Theorem 11** *Let $C$ be an enumerable set of NB-clauses and let $O$ be an a non-repeating enumeration of the variables in $C$. Let $T_C$ be* the *$O$-ordered NB-semantic tree for $C$. If $N$ is an inference node in $T_C$ and $c_1, \ldots, c_k$ label its descendants then $c_1, \ldots, c_k$ are $O$-ordered NB-resolvable and their $O$-ordered NB-resolvent is an NB-clause that is violated by $N$ (and perhaps some of its ancestors).*

*Proof:* Let variable $X$ label the edges from $N$. Then, by Theorem 7, $c_1, \ldots, c_k$ are NB-resolvable upon $X$ and their NB-resolvent upon $X$ is an NB-clause that is violated by $N$ (and perhaps some of its ancestors). It remains to show that this resolution is $O$-ordered. Since $c_1, \ldots, c_k$ label failure nodes every variable that occurs in these NB-clauses is assigned a value by each of the nodes. Thus, each of these variables label an edge on the path from the root to $N$. Since $X$ labels the edges from $N$ and the NB-semantic tree is $O$-ordered, $X$ must be greater than all the other variables that appear in $c_1, \ldots, c_k$. Hence, the NB-resolution of $c_1, \ldots, c_k$ upon $X$ is $O$-ordered. □

To illustrate this, consider inferences (8) and (9) and ordering $O = X_1, X_4, X_2, X_3$. We have previously observed that these two inferences arise from considering the two inference nodes of Search Space 2, which is an $O$-ordered NB-semantic tree. Consequently, Theorem 11 tells us that these inferences are $O$-ordered and that their resolvents prune Search Space 2. As we have previously observed, this is indeed the case.

On the other hand, inference (7) is not $O$-ordered for $O = X_1, X_4, X_2, X_3$. Looking at Search Space 2 we can see that its resolvent, $X_2/\{b, g\}$, is violated only at failure nodes. Hence, it eliminates no backtrack nodes from the search space. We shall not address whether this is true in general: that NB-resolvents that are not $O$-ordered prune no backtrack nodes from the $O$-ordered NB-semantic tree.

As a contrapositive to Theorem 11 we have the following corollary:

**Corollary 12** *If $C$ is closed under $O$-ordered NB-resolution then the $O$-ordered NB-semantic tree for $C$ contains no inference node.*

Since, according to Lemma 4, a semantic tree contains a backtrack node if and only if it contains an inference node, it immediately follows from Corollary 12 that:

**Theorem 13** *If $C$ is closed under $O$-ordered NB-resolution then the $O$-ordered NB-semantic tree for $C$ is backtrack-free.*

Let us continue with the example colouring problem of Figure 6 and use ordered NB-resolution to generate a backtree-free search space for the order $O = X_1, X_4, X_2, X_3$. We have already seen that inferences (8) and (9), which derive clauses (q) and (r), are $O$-ordered. We can then continue with the following two inferences:

$$\begin{array}{ll}
\text{(b)} & X_1/\{b,g\} \vee X_2/\{b,g\} \\
\text{(o)} & X_2/\{r,g\} \ \vee \ X_4/\{r,g\} \\
\text{(r)} & X_1/\{b,g\} \ \vee \ X_4/\{r,g\} \ \vee \ X_2/\{r,b\} \\
\hline
\text{(s)} & X_1/\{b,g\} \ \vee \ X_4/\{r,g\}
\end{array}$$

$$\begin{array}{ll}
\text{(b)} & X_1/\{b,g\} \vee X_2/\{b,g\} \\
\text{(q)} & X_1/\{b,g\} \ \vee \ X_2/\{r,g\} \vee \ X_4/\{r,b\} \\
\text{(p)} & X_2/\{r,b\} \ \vee \ X_4/\{r,b\} \\
\hline
\text{(t)} & X_1/\{b,g\} \ \vee \ X_4/\{r,b\}
\end{array}$$

The set containing clauses (a) through (t) produces a backtrack-free NB-semantic tree for ordering $X_1, X_4, X_2, X_3$, as shown in Figure 7.

And, as in the previous two sections, if $C$ is unsatisfiable we have the following corollary:

**Corollary 14** *If $C$ is unsatisfiable and closed under $O$-ordered NB-resolution then it contains the empty clause.*

# 6 Future Directions

Certain problems have natural representations that are non-clausal, and translating them to clausal form can blow up the size of the formulas. It would be advantageous to leave them in non-clausal form and use non-clausal resolution to remove backtracking from the search space. Can non-clausal resolution be generalised to handle non-Boolean formulas? Can orderings be built into the resulting inference rule? Does closure under these non-clausal inference rules result in a backtrack-free search space?

Certain problems could be represented much more succinctly if our clauses allowed variables that range over domain values. Graph colouring is one such problem. To express that two nodes have different colours, the representation used in this paper requires one clause for each colour. However, with variables, we could express the constraint that $X_1$ and $X_2$ have different colours as a single clause:

$$\forall x \, \neg X1(x) \ \vee \ \neg X2(x).$$

The question, then, is whether the results of this paper could be generalised to clauses with variables.

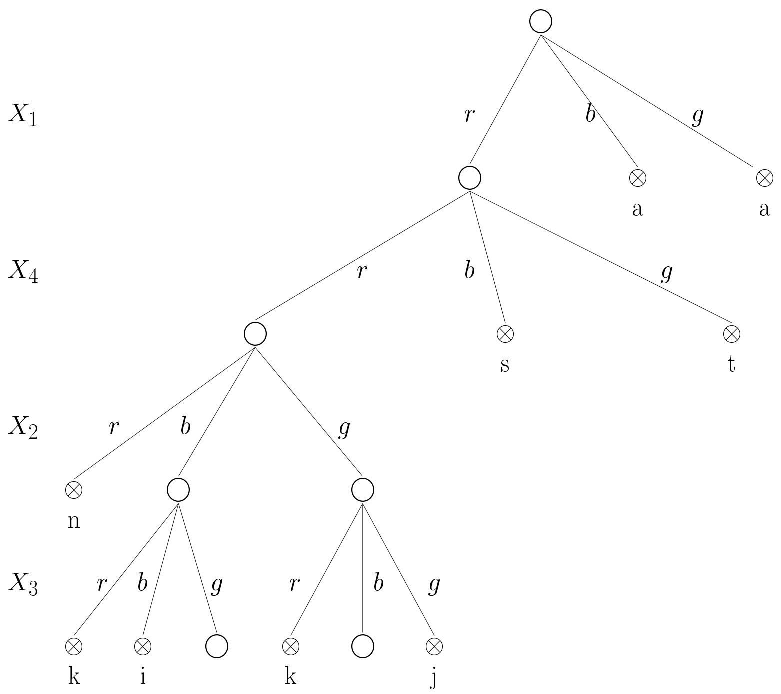


Figure 7: Search Sapce 3.

# References

[Baaz and Fermüller, 1995] Matthias Baaz and Christian G Fermüller. Resolution-based theorem proving for many-valued logics. *Journal of Symbolic Computation*, 19:353–391, 1995.

[de Kleer, 1989] Johan de Kleer. A comparison of ATMS and CSP techniques. In N. S. Sridharan, editor, *Proc. Eleventh International Joint Conference on Artificial Intelligence*, pages 290–296, Detroit, Michigan, August 1989.

[Kowalski and Hayes, 1969] R. Kowalski and P. J. Hayes. Semantic trees in automatic theorem proving. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 87–101. Edinburgh University Press, 1969.

[Robinson, 1968] J.A. Robinson. The generalized resolution principle. In D. Michie, editor, *Machine Intelligence 3*, pages 77–93. Edinburgh University Press, 1968.

[Stickel, 1985] Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(4):333–355, 1985.