Constraint Programming

- Symmetries in CP

  - Types of Symmetries
  - Symmetry Detection and Breaking
  - Symmetry Breaking: Reformulation
  - Symmetry Breaking: Before and During Search

Types of Symmetries

Types of Symmetries

- Although, in an abstract way, symmetries are well defined as permutations, different types of symmetries have been defined in Constraint Programming, namely because the methods used to break them are not completely general.
- In the beginning we gave the informal definition below:
  - a symmetry is some particular transformation of the solution of a problem into another solution of the problem, or from a non-solution into a another non-solution.
- This definition, is not only quite informal but also not very useful for direct application, since the purpose of defining symmetries is exactly to avoid computing all the symmetric solutions.
- To formalise symmetries, let us consider them not only defined on the set of solutions, but also as a property of the problem. These notions are captured in the following definitions.

Types of Symmetries

Solution symmetries

- A solution symmetry is a permutation of the set of <variable,value> pairs which preserves the set of solutions.

Problem symmetries

- A problem symmetry is a permutation of the set of <variable,value> pairs which preserves the set of constraints.

- A number of other definitions are stricter than the general definitions above.
- In particular, two kinds of symmetries are very important as they are often present and detectable in many problems.

Types of Symmetries

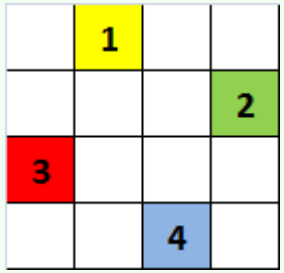
- **Value Symmetries**

  A value symmetry is a symmetry **s** that is a permutation on the values of the variables of the problem, i.e.

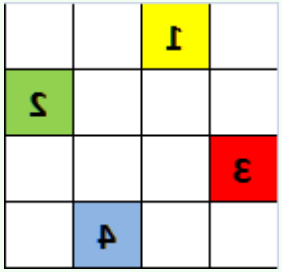
  s(x = a) ≡ x = s(a)

- These symmetries define permutations of values that are applicable to all variables. For example, the symmetry **v** in n*n board problems are value symmetries (the case for n = 4 is shown below)

vi → vj
1 → 4
2 → 3
3 → 2
4 → 1



id



v

Types of Symmetries

- The next definition is dual of the value symmetry.

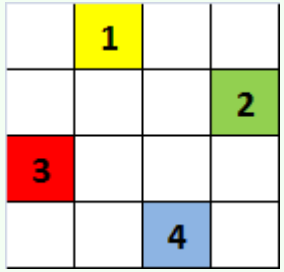
Variable Symmetries

A variable symmetry is a symmetry **s** that is a permutation on the variables of the problem, i.e.

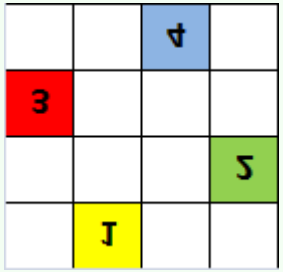
$$s(x = a) \equiv s(x) = a$$

- These symmetries define permutations of variables that are applicable to all values. For example, the symmetry **h** in n*n board problems are value symmetries (the case for n = 4 is shown below)

xi → xj
x1 → x4
x2 → x3
x3 → x2
x4 → x1



id



h:

Types of Symmetries

- A number of other definitions are more strict than the general definitions above. The next definitions are value symmetries that are a limited form of solution symmetry,

Interchangeable Symmetries

Two values a and b for a variable v are fully interchangeable iff every solution to the CSP containing the assignment v=a remain a solution when b is substituted for a, and vice-versa.

- A similar restriction for problem symmetry can be defined

Neighbourhood Interchangeable Symmetries

Two values a and b for a variable v are neighbourhood interchangeable iff for every constraint C of the problem on variable v, the set of variable-value pairs that satisfy the constraint with v = a, is the same of set of variable-value pairs that satisfy the constraint with v = b.

Types of Symmetries

- The next definitions exploit further the previous ideas of Interchangeablility.

Semantic Symmetries (satisfiability)

- Two values a and b are semantically symmetric for satisfiability iff whenever there is a solution with v=a there is a solution with v = b.
- The above definition is useful in the case we are interested in a single solution, in which case we can remove one of the values. However, the solutions obtained with v=a can be very different from those with v=b, with no obvious mapping. In optimisation problems the following symmetry is more important

Semantic Symmetries (all solutions)

- Two values a and b are semantically symmetric for all solutions iff there is a mapping from whatever solution with v=a into a solution with v = b, and vice-versa.
- In some cases, especially for breaking symmetries during search it is important to define symmetries over partial assignments.

Symmetry Detection and Symmetry Breaking

- There are two main issues in dealing with symmetries in Constraint Programming, namely

Symmetry Detection

- Symmetries have to be detected, before they can be efficiently addressed. In some cases this is an obvious task, in others, specially when their number is very large, this is not so easy. We will later address the problem of automated detection of symmetries, but for the moment we will assume the symmetries are known.

Symmetry Breaking

- Once detected, methods have been studied to take advantadge of the symetries, namely to avoid the exploitation of useless parts of the search tree, whose positive or negative findings can be obtained by applying symmatrical reasoning on the exploited parts.

Symmetry Breaking

- Breaking the symmetries can be performed by a number of techniques. They can be classified as follows:

Problem Reformulation

- A new model of the problem may be studied, that eliminates some (or all) of the symmetries that were found. Of course it is assumed that the new and the old model are equivalent, i.e. they lead to the same solutions (modulo symmetry). Reformulation is highly problem-dependent, so there are not many general purpose techniques available. We will nevertheless discuss several problems dealing with sets.

Addition of Symmetry Breaking Constraints

- When symmetries are detected in a model, additional constraint may be added to the model in order to eliminate these symmmetries. Such addition can be static (before execution) or dynamic (during execution).

Symmetry Breaking by Reformulation

Sets

- By nature, problem reformulation does not lead to many general purpose techniques for symmetry breaking through problem reformulation. An important exception is the dealing with **sets**.
- If one wants to find a set of **k** objects in some finite domain that satisfy some constraints, a simple way to model this requisite is to create k finite-domain variables, and assign them values in their domain that satisfy the constraints.
- In fact, this is not enough, since sets do not accept repetitions. But this can be easily imposed by an all-different constraint over the k variables.
- However, this is not sufficient in that the resulting model has a huge number of undesirable symmetries. In fact, if there are k variables, any permutation of the variables is a valid symmetry, and hence there are **k!** such undesirable variable-symmetries.
- We can see how to address this reformulation problem with an example.

Symmetry Breaking by Reformulation

Example: Find a set of 3 elements in the range 1..7.

- In this case, solutions $< x_1 = 2, x_2 = 4, x_3 = 7 >$ and $< x_1 = 7, x_2 = 4, x_3 = 2 >$ are exactly the same, in that the intended set is composed by values {2,4,7}.
- Although these symmetries can be solved by adding symmetry breaking constraints, another way to break them is by means of reformulating the problem with sets.
- Rather than finding the value for k finite domain variables, the problem is reformulated as finding a set with k elements.
- In addition to specialised modelling of sets available in languages such as Conjunto and Cardinal, a simple general technique to represent a set is by means of Boolean variables, one for each possible value in the domain of the set.
- Finding the set is to find an assignment of k 1's to these variables, that represent the elements in the set.

Symmetry Breaking by Reformulation

Example (cont) : Find a set of 3 elements in the range 1..7.

i. Boolean variables to represent the set: b1, .., b7.

ii. Cardinality of the set: **`b1+ b2 + ... + b7 #= 3`**.

iii. Hence, the solution {2,4,7} is univocally represented as

`b2 = b4 = b7 = 1` and **`b1 = b3 = b5 = b6 = 0`**

- In addition to these variables we should add some channeling constraints to map the set representation into information about set membership.

- In this case, such mapping can be done with k reified constraints:

`i ∈ S #<=> bi #= 1.`

Symmetry Breaking by Reformulation

- In other cases, set variables are not so easy to implement. See the following example from Bioinformatics

Example: A number of yeasts y_1, ..., y_m, is to be identified through digestion of a number of enzymes e_1, ..., e_n. An enzyme may or may not differentiate a pair of yeasts y_i-y_j. The problem consists of finding a minimal set of enzymes that, for every pair of yeasts, contains an enzyme that differentiate them.

- A straightforward model of the problem is to assign for every enzyme e_k, a set E_k of booleans s_{kij}, 1 <= i < m, i < j <= where each s_{kij} indicates whether enzyme e_k differentiates yeasts y_i and y_j. Now the problem is cast into a set covering problem – find a set X of enzymes that covers all pairs of yeasts.
- With a Boolean set formulation, the set X is represented by Booleans x_1, ..., x_k, (denoting whether enzymes x_1, ..., x_k are selected) such that the following k constraints are satisfied

$$\forall_{1 \le i < m}, \forall_{i < j \le m} \sum_k x_k * s_{kij} = 1$$

- Now, the minimization of the cardinality of such sets is obtained with

$$\min \sum_k x_k$$

Symmetry Breaking by Reformulation

- The solution proposed already assumes the set of enzymes to be represented by the Boolean variables x_i.
- There are however some drawbacks with this Boolean sets representation. If a set of k elements with domain 1..d is represented by d Boolean variables, the search space is 2^d. With a representation with k finite domain variables, the search space is d^k, which is usually much lower.
  - For d = 30 and k = 4, we have $2^d \approx 10^9$ and $30^k \approx 8.1*10^5$.
  - In our example it is d = 300 and k = 3, i.e. $2^{300} \approx 2*10^{90}$ and $300^3 \approx 2.7*10^7$
- Moreover, there is a global constraint, **`nvalue(List, N)`**, that associates a **List** of variables with the number **N** of different values in that list, that can be exploited in a finite domain representation.
- All that is needed is to create, for each of the pairs y_i - y_j, a variable $z_{i,j}$ whose domain is the id of the enzymes e_k that differentiate them. Now the problem is simply modelled by

`min N` such that **`nvalue([`**$z_{1,2}$**`,`** $z_{1,3}$**`, ...., `**$z_{m-1,m}$**`], N)`**

Symmetry Breaking by Reformulation

- Although this model is much more efficient than the Boolean alternative, still there is a huge number of symmetries that are not considered in the nvalue global constraint.
- For example, and for a simplified problem with only 4 yeasts (6 yeast pairs) and 5 enzymes, a number of solutions are symmetric. For example, the lists

  `[1,4,1,4,4,1]` and `[4,4,1,4,1,1]`

  represent the same set of enzymes {e1, e4}.
- In the worst case for a set with N enzymes chosen from a total of k available enzymes there are C^k_N potential representations of the same set.
- In our case, the minimal sets have 3 enzymes, chosen from 300 possibilities, so the number of symmetries can be as high as $C^{300}_3 \approx 4.5 * 10^6$.
- This large number of symmetries in fact prevents this model to find alternative sets of enzymes, since the same set is obtained a huge number of times (although less than the worst case of 4.5 million).

Symmetry Breaking before Search

- Another approach to break symmetries consists of adding symmetry breaking constraints to the problem, before starting the search.
- This is a technique that has been used ad hoc in many applications, the best example of it is possibly in the management of sets represented by finite domain variables.
- Given a set S of k elements, these can be modelled by k variables s_1, ... , s_k that will be assigned to the (different) elements of the set.
- Of course any permutation of variables is acceptable, and hence there are k! variable symmetric solutions.
- For example, for the set {1,2,3} the following solutions are possible (given as tuples for variables $<s_1,s_2,s_3>$

  <1,2,3>, <1,3,2> , <2,1,3>, <2,3,1>, <3,1,2> and <3,2,1>
- Since any of these solutions are acceptable, one may consider one of the solutions as “canonical”, whereas the others can be obtained by symmetry transformation.

Symmetry Breaking before Search

- When there is no priviliged solution, one may simply impose an arbitrary solution as canonic. Usually, the canonical solution imposes an increased ordering on the lexicographic order of the variables, i.e.

$$x_1 < x_2 < x_3$$

  i.e. the canonical solution is <1,2,3>.

- This technique may be used in applications envolving several sets, such as the well known example of the

Social Golphers problem (g,s,w):

- The goal is to schedule **g*s** golphers, in **g** groups of **s** players each, such that they can play for **w** weeks and two players never play in the same group more than once.

- This problem presents a huge number of symmetries that prevents an efficeint execution, even for small numbers of g, s and w, unless these symmetries are not broken. To simplify, we will use the values g = 3, s= 2, and w = 4.

Symmetry Breaking before Search

- **Value symmetries** [(g*s)!]: If a solution assigns to the variables x_1, x_2 , ... , x_n (with n = g*s) different values in the range 1, 2, ..., n, any permutation of these values is also a solution.

  This is a typical situation in sport tournaments: a schedule is prepared in terms of virtual teams t_1, t_2, t_n, which are assigned an arbitrary assignmnet (by a random selection – e.g drawing the names from a box)

- **Group Symmetries** [w*g* (s !)]: : Within each group, a set, the elements can be permuted

- **Group Symmetries within each weak** [w*(g!)]: the order of the groups in each week is arbitrary.

- **Week Symmetries** [w!] : The weeks are arbitrarily permuted.

- All together, there are (w!) * w*(g!) * w*g*(s!) * (s*g)! Symmetries. This may be a huge number. Even in the small instances that is considered here, there are 4! * 4*3! * 4*3* 2! * (3*2!) = 24*24*24*720 ≈ 10^7 symmetries.

Symmetry Breaking before Search

- A number of ad hoc constraints can be used to eliminate these constraints. Let us represent a solution by variables x_{wgp} where w is the week, g the group and p the position in the group of the player

$$\{ x_{111} , x_{112} \} , \{ x_{121} , x_{122} \} , \{ x_{131} , x_{132} \}$$

$$\{ x_{211} , x_{212} \} , \{ x_{221} , x_{222} \} , \{ x_{231} , x_{232} \}$$

$$\{ x_{311} , x_{312} \} , \{ x_{321} , x_{322} \} , \{ x_{331} , x_{332} \}$$

$$\{ x_{411} , x_{412} \} , \{ x_{421} , x_{422} \} , \{ x_{431} , x_{432} \}$$

- Taking into account an implicit all-different constraint on the variables in each week:

Value symmetries:

Some of them can be broken by fixing the values for the first week, which are completely arbitrary as, for example, { 1 , 2 } , { 3 , 4 } , { 5, 6 }

Symmetry Breaking before Search

Group symmetries:

- The standard technique can be applied to sort the sets in increasing order, by adding constraints $\mathbf{x_{wgp} < x_{wgq}}$ for all ellements (**p < q** in 1..2) of all groups (**g** in 1..3) of all weeks (**w** in 1..4).

$\{x_{111} < x_{112}\}, \{x_{121} < x_{122}\}, \{x_{131} < x_{132}\}$

Group in Weeks symmetries:

Taking into account that the first elements of the groups in the same week must be different and smaller than the other elements of the corresponding groups, a standard order of the groups may be imposed by adding constraints $\mathbf{x_{wg1} < x_{wh1}}$ for all all weeks (**w** in 1..4). 1..4 and for all groups in each week (**g< h** in 1..3).

Week symmetries:

$\{x_{111}, x_{112}\}, \{x_{121}, x_{122}\}, \{x_{131}, x_{132}\}$

- The previous constraints force, for all weeks (w in 1..4), the first element of the first group to be 1 ($x_{w11} = 1$). Then the second elements must be different. Week symmetries may be broken by imposing $x_{w12} < x_{v12}$ for each pair of weeks w < k in 1..4.

$\{x_{111}, x_{112}\}, \ldots$

$\{x_{211}, x_{212}\}, \ldots$

Symmetry Breaking before Search

- In this case it was possible to identify a number of symmetry breaking constraints to be posted before the search that can break most if not all the existing symmetries in the initial finite domain model.
- These constraints are in general easier to identify when the variables are all different, in which case, a total ordering of the variables may be imposed.
- Even when no such all-different constraints exists, non strict total ordering can nevertheless be imposed - the **lexicographic** ordering – to define a canonical solution, thus breaking variable symmetries.
- For example, let us assume that we have 3 variables, A, B and C that form a symmetry group, i.e. any permutation of them is still a solution of the problem.
- Intuitively, we may impose an ordering $A \leq B \leq C$ to break the variable symmetries. Hence. If a solution is found with values
- **3, 4, 5**: then the canonical solution is **A = 3, B = 4, C = 5**
- **3, 4, 4**: then the canonical solution is **A = 3, B = 4, C = 4**.

Symmetry Breaking before Search

- In other cases, the symmetry breaking constraints are not so easily obtained. However, there is a technique that is always safe (if not always practical) to break variable symmetries of a symmetry group. This technique, lex-leader

  1. assumes that the list of variables can be sorted in some (alphabetic) order, V.
  2. For every symmetry s in the variable symmetry group S one lex-leader constraint is added:

$$\forall s \text{ in } S : V \leq s(V)$$

- In our example, we have V = ABC. The symmetries correspond to permutations of the variables and the lex-leader constraints are thus

$$ABC \leq ABC \qquad ABC \leq BAC \qquad ABC \leq CAB$$
$$ABC \leq ACB \qquad ABC \leq BCA \qquad ABC \leq CBA$$

- In the constraints above UV ≤ XY has the usual lexicographic meaning, i.e.

$$\alpha(U) < \alpha(X) \quad \vee \quad (\alpha(U) = \alpha(X) \wedge \alpha(V) \leq \alpha(Y))$$

- where $\alpha(X)$ denotes the value assigned to variable X.

Symmetry Breaking before Search

- From a practical view point, the lex-leader method may impose an exponential number of constraints. However, some simplifications are possible, as in the following examples.

Example 1:

The variables A, B, C are all different and may be permuted freely.

$ABC \leq ABC$ $\quad$ $ABC \leq BAC$ $\quad$ $ABC \leq CAB$

$ABC \leq ACB$ $\quad$ $ABC \leq BCA$ $\quad$ $ABC \leq CBA$

i. Now, $ABC \leq BAC$ can be simplified to $A < B$ (since $A \neq B$); and

ii. $ABC \leq ACB$ can be simplified to $BC \leq CB$ and then to $B < C$.

- By now we have the total order of the variables, $A < B < C$, that is usually introduced in an ad hoc manner. Moreover, the other constraints are redundant. For example

$ABC \leq BCA$ follows from $A < B$.

Symmetry Breaking before Search

Example 2:

The variables A, B, C are **not** all different and may be permuted freely.

$ABC \leq ABC$ $\quad ABC \leq BAC$ $\quad ABC \leq CAB$

$ABC \leq ACB$ $\quad ABC \leq BCA$ $\quad ABC \leq CBA$

i. $ABC \leq ACB$ can be simplified to

$$B < C \ \vee \ (B = C \ \wedge \ C \leq B) \ \Leftrightarrow \ B < C \ \vee \ B = C \ \Leftrightarrow \ B \leq C$$

ii. $ABC \leq BAC$ can be simplified to

$$A < B \ \vee \ (A = B \ \wedge \ B \leq A) \ \Leftrightarrow \ A < B \ \vee \ A = B \ \Leftrightarrow \ A \leq B$$

- Again, we were led to a non-strict total order of the variables, $A \leq B \leq C$, that would be introduced ad hoc. Also, the other constraints are redundant. For example, $ABC \leq CAB$ can be simplified to (given the above constraints)

$$A < C \ \vee \ (A = C \ \wedge \ \cancel{B < A}) \ \vee \ (A = C \ \wedge \ B = A \ \wedge \ C \leq B)$$

$$\Leftrightarrow \ A < C \ \vee (A = C \ \wedge \ B = A \ \wedge \ B = C)$$

$$\Leftrightarrow \ A < C \ \vee \ (A = C \ \wedge \ B = A)$$

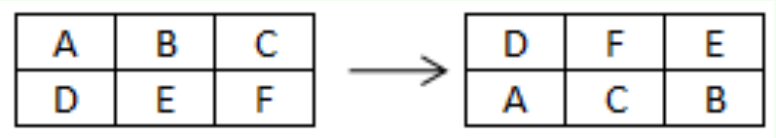
which is redundant given $A \leq B$ and $B \leq C$

Symmetry Breaking before Search

Example:

This is an example of a matrix model, where variables are organised as a 2*3 matrix, for which the rows and the columns can be permuted freely (for example the latin squares have this kind of symmetries)

- In the figure, there were permutations of rows 1 and 2 and columns 2 and 3.



In this case there are 2! 3! = 12 symmetries (for the permutations of 2 rows and 3columns) leading to 12 lex-leader constraints

1. ABCDEF ≤ ABCDEF
2. ABCDEF ≤ ACBDFE
3. ABCDEF ≤ BACEDF
4. ABCDEF ≤ CBAFED
5. ABCDEF ≤ BCAEFD
6. ABCDEF ≤ CABFDE
7. ABCDEF ≤ DEFABC
8. ABCDEF ≤ DFEACB
9. ABCDEF ≤ EDFBAC
10. ABCDEF ≤ FEDCBA
11. ABCDEF ≤ EFDBCA
12. ABCDEF ≤ FDECAB

Symmetry Breaking before Search

Example 3 (cont):

Some of these constraints can be simplified. For example, for constraint 2

$$
\begin{aligned}
& ABCDEF \leq ACBDFE \\
\Leftrightarrow\ & BCEF \leq CBEF \\
\Leftrightarrow\ & B < C \ \vee\ (B=C \ \wedge\ CEF \leq BFE) \\
\Leftrightarrow\ & B < C \ \vee\ (B=C \ \wedge\ EF \leq FE) \\
\Leftrightarrow\ & B < C \ \vee\ (B=C \ \wedge\ (E < F \ \vee\ (E = F \ \wedge\ F \leq E)) \\
\Leftrightarrow\ & B < C \ \vee\ (B=C \ \wedge\ (E < F \ \vee\ E = F) \\
\Leftrightarrow\ & B < C \ \vee\ (B=C \ \wedge\ E \leq F) \\
\Leftrightarrow\ & BE \leq CF
\end{aligned}
$$

Symmetry Breaking before Search

Example 3 (cont):

Similar reasoning can be applied to all the other constraints below,

1. ABCDEF ≤ ABCDEF
2. ABCDEF ≤ ACBDFE
3. ABCDEF ≤ BACEDF
4. ABCDEF ≤ CBAFED
5. ABCDEF ≤ BCAEFD
6. ABCDEF ≤ CABFDE
7. ABCDEF ≤ DEFABC
8. ABCDEF ≤ DFEACB
9. ABCDEF ≤ EDFBAC
10. ABCDEF ≤ FEDCBA
11. ABCDEF ≤ EFDBCA
12. ABCDEF ≤ FDECAB

resulting in the simplified lex-leader constraints.

1. true
2. BE ≤ CF
3. AD ≤ BE
4. AD ≤ CF
5. ABDE ≤ BCEF
6. ABDE ≤ CAFD
7. ABC ≤ DEF
8. ABC ≤ DFE
9. ABC ≤ EDF
10. ABC ≤ FED
11. ABCDE ≤ EFDBC
12. ABCDE ≤ FDECA

Symmetry Breaking before Search

Example 3 (cont):

Seen as a set of constraints, they can be further simplified to,

1. true
2. BE ≤ CF
3. AD ≤ BE
4. AD ≤ CF
5. ABDE ≤ BCEF
6. ABDE ≤ CAFD
7. ABC ≤ DEF
8. ABC ≤ DFE
9. ABC ≤ EDF
10. ABC ≤ FED
11. ABCDE ≤ EFDBC
12. ABCDE ≤ FDECA

resulting in the simplified lex-leader constraints.

1. true
2. BE ≤ CF
3. AD ≤ BE
4. true
5. true
6. true
7. ABC ≤ DEF
8. ABC ≤ DFE
9. ABC ≤ EDF
10. ABC ≤ FED
11. ABCD ≤ EFDB
12. ABC ≤ FDE

Symmetry Breaking before Search

Lex-Leader and All-different

In case the variable symmetries occur in a set of variables that are all-different, a number of more interesting results have been proven. For example, given A = A and B ≠ C. the lex-leader constraints can be simplified , such as

$$ABCDEF \leq ACBDFE \quad \rightarrow \quad B < C$$

- This is a general result that can be generalised as follows

Lemma:

Given a CSP with n variables V that form a variable-symmetry group S, subject to an all-different constraint, than all variable symmetries can be broken by adding the following constraints

$$\forall s \text{ in } S : v(i) < \; v(s(i))$$

where i is the first position where the lex-leader constraint has different variables, and s(i) the corresponding variable in the symmetry.

Symmetry Breaking before Search

- The previous resullt can be used only in a subset of the lex-leader constraints, which are found with a n algorithm (**Scheier Sims**) that is available in some Computational Group Theory tools.

- Moreover, it has been used to prove the following

Theorem:

Given a CSP with n variables V, subject to an all-different constraint than all variable symmetries can be broken by adding at most n-1 constraints.

- This theorem thus guarantees that the exponential number of lex-leader constraints that might be needed in the general case, can be reduced to n-1.

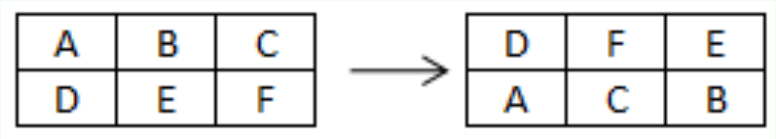
- This was the case with the simple permutation of n variables, xi, (cf example 1) whose symmetries could be broken by the n-1 constraints

$$x_1 < x_2 \ , \quad x_2 < x_3 \quad , \ldots, \quad x_{n-1} < x_n$$

Symmetry Breaking before Search

Lex-Leader and all-different

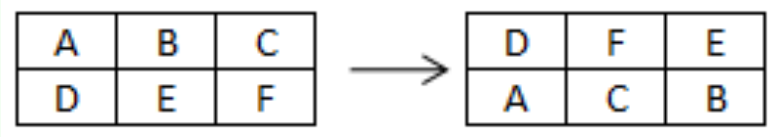
The 12 lex-leader constraints of example 3 can also be reduced as follows



2. ABCDEF ≤ ACBDFE → **B < C**
3. ABCDEF ≤ BACEDF → **A < B**
4. ABCDEF ≤ CBAFED → A < C from 2 and 3
5. ABCDEF ≤ BCAEFD → A < B
6. ABCDEF ≤ CABFDE → A < C
7. ABCDEF ≤ DEFABC → **A < D**
8. ABCDEF ≤ DFEACB → A < D
9. ABCDEF ≤ EDFBAC → **A < E**
10. ABCDEF ≤ FEDCBA → **A < F**
11. ABCDEF ≤ EFDBCA → A < E
12. ABCDEF ≤ FDECAB → A < F

Symmetry Breaking before Search

Lex-Leader and all-different



In this case, the 12 symmetries are broken by only 5 constraints

A < B , B < C, A < D , A < E , A < F

with an obvious interpretation:

- Variable A must be to the left of B (they are in the same row)
- Variable B must be “to the left” of C (they are in th same row)
- Variables D, E and F must be “above” variable A (they are in a different row of A)

- Notice that it is not imposed that D < E (nor E < F) because they must be in the same columns that A and B (B and C) , and so must remain.

Symmetry Breaking before Search

Lex-Leader when NOT all-different

- Since the set of lex-leader constraints might have an exponential number of elements, sub-sets of the lex-leader constraints have been tried by several authors.
- Of course, if not all lex-leader constraints are posted, it is not guaranteed that all variable symmetries are removed.
- However, the goal is to achieve a good trade-of between a reasonable number of broken symmetries and a polinomial number of lex-leader constraints (or their simplification).
- An obvious subset is composed, not by all variable symmetries in the variable symmetry group, but only a subset of its generators. (in a SAT problem, very good results were obtained by using only the 21 generators of a symetry group with 1016 elements).

Symmetry Breaking before Search

Matrix Models

- Another approach to symmetry breaking before search, similar but not the same as lex-leader, has been studied in special cases of variable symmetries, occurring in matrix models.
- These are problems where variables can be regarded as disposed in a matrix and all the constraints are preserved when columns and rows are swapped, as in the example seen before
- Many applications fit these matrix models, namely latin-squares problems in 2 or a higher number of dimensions). Another example is the social golfers, that can be modelled in 3-dimensions by Boolean variables x_{wgp}, which are true if in week w, player p plays in group g.
- In these models, symmetry breaking constraints can be obviously posted to impose a lexicographic order in the rows, or a lexicographic order in the columns.

Dynamic Symmetry Breaking

Dynamic Symmetry Breaking (DSB)

- Rather than posting constraints before the search starts, as done in lex-leader and ad hoc methods that have been used in specific problems, DSB methods, analyse the search at every choice point and perform some extra operations to avoid the exploration of paths leading to symmetric solutions. Two main methods have been proposed

SBDS – Symmetry Breaking During Search

Assuming that an assignment has been tested, all the symmetric assignments are excluded from the search by addition of symmetry breaking constraints.

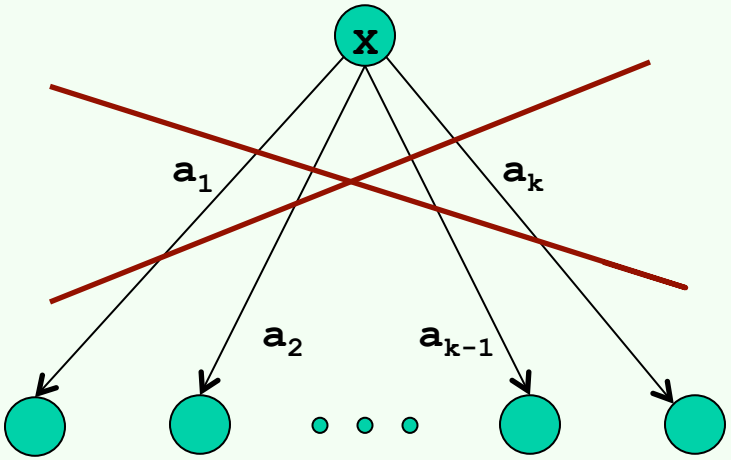
SBDD – Symmetry Breaking via Dominance Detection

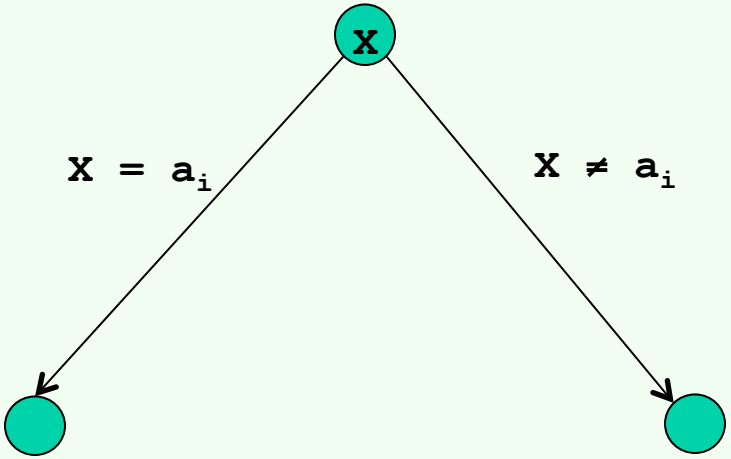
At every node of the search tree, SBDD checks whether the node is symmetric of some other node already exploited, in which case it does not expand the node.

Dynamic Symmetry Breaking

- Before addressing these two DSB methods, we note that both methods assume that, during enumeration, search is made according to a binary search tree.

- In practice, if a variable with k values in its domain is selected, its enumeration is not done according to a k-ary tree (by expanding its k children), but rather by k binary expansions of its nodes, either on an assignment or its negation.

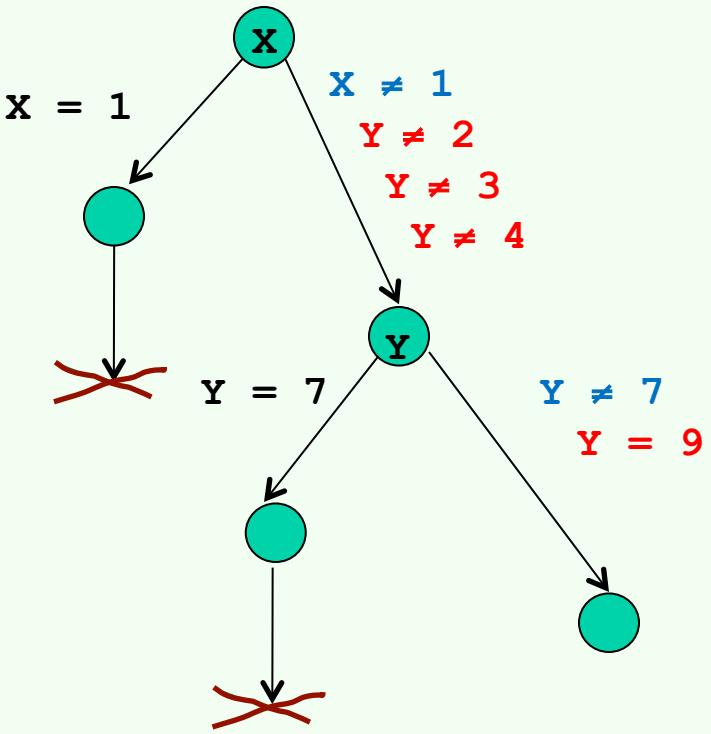




Dynamic Symmetry Breaking

- Binary expansions allow for a more flexible application of heuristics. Different variables may be picked in alternance.

- **Example**: X in {1, 6, 8, 9} Y in {2, 3, 4, 7, 9} , X =< Y, p(X,Y)



X in {1,6,8,9}
Y in {2,3,4,7,9}

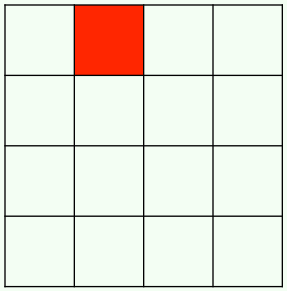
X in {6,8,9}
Y in {7,9}

X in {6,8,9}
Y in {9}

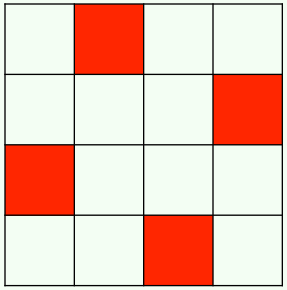
SBDS by example

- In SBDS, the negations done in the right branches take into account not only the actual assignments made but also their symmetrical.

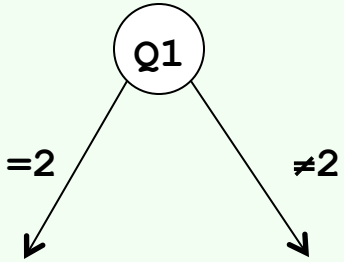
Example: 4 queens



- Let us assume that the first assignment made is Q1 = 2 (left branch).



- This assignment is fully explored (it leads to the single solution shown) in the left hand branch.



- Then the right hand branch introduces constraint Q1 ≠ 2, aiming at some optimisation (by propagation)

SBDS by example

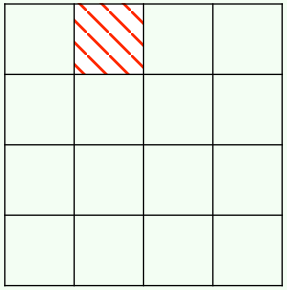
Example: 4 queens

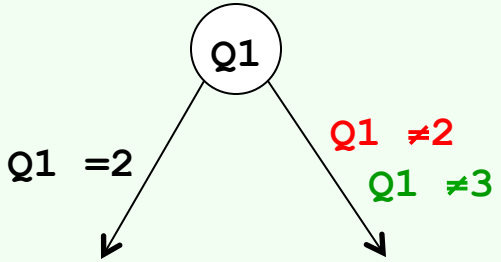
- The added constraint Q1 ≠ 2 guarantees that already found solution(s) (with Q1 = 2) are not found again).

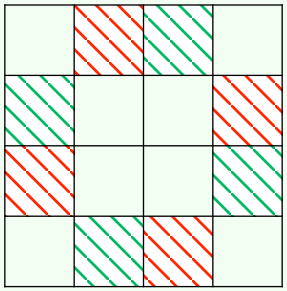
- But finding symmetrical solutions may also be avoided if symmetrical constraints are posted.

- For example, the vertical reflection of assignment Q1 = 2 is Q1 = 3, so this constraint can be added as well.

- In fact this eliminates finding the other solution, which can be obtained later by (vertical reflection) symmetry.

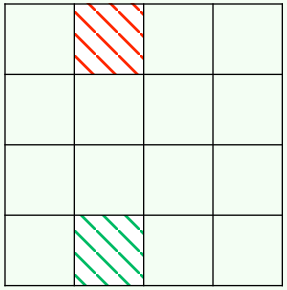






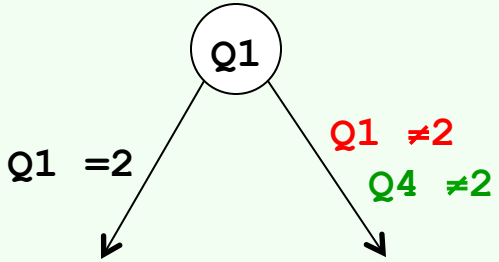
SBDS by example

Example: 4 queens



- The technique works with any symmetry. We could use vertical symmetry to exclude Q4 = 2.

- Again the other solution (with Q1 = 3 and Q4 = 2) would not be rediscovered.



Inneficiency 1

- This would possibly be not as efficient as the elimination of the vertical reflection, as it would lead to 2 variables with 3 values in the domain

  Q1 in {1,3,4}
  Q4 in {1,3,4}

  rather than 1 variable with only two values in the domain

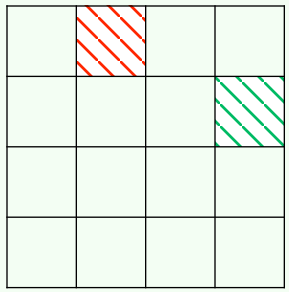
  Q1 in {1,4}

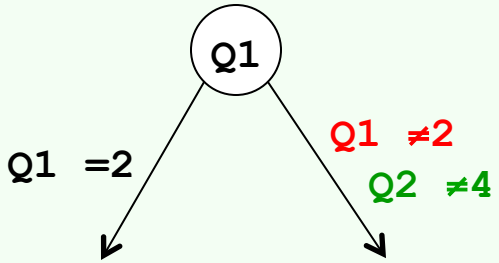
SBDS by example

Example: 4 queens

- If the technique works with any symmetry, some will lead to useless prunning.

- For example, 90º rotation will exclude Q2 = 4.

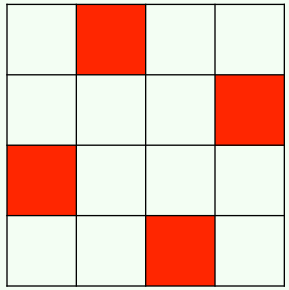




Inneficiency 2

- In this case, we would simply eliminate the rediscovery of the same solution already found with Q1 = 2, i.e.

{Q1 = 2, Q2 = 4, Q3 = 2, Q4 = 3 }



SBDS by example

- The situation is a bit more complex when the node where the symmetry breaking constraint is added is not the root node.

- Take the example shown. In this case, we cannot simply state Y ≠ 4 on the right branch of node Y, because it could lead to loss of solutions

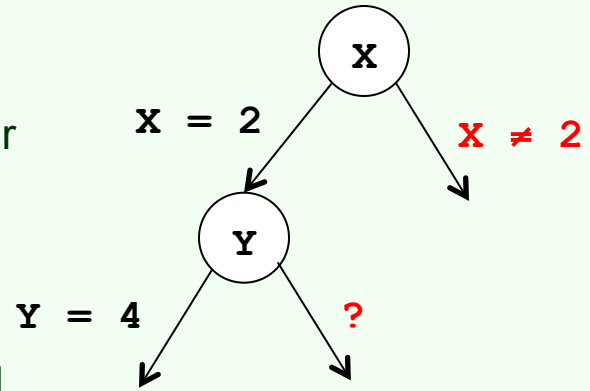
- For example, a solution with X = 3 and Y = 4.

- All that is safe to is impose that Y = 4 is no longer acceptable in the context of X = 2, i.e.

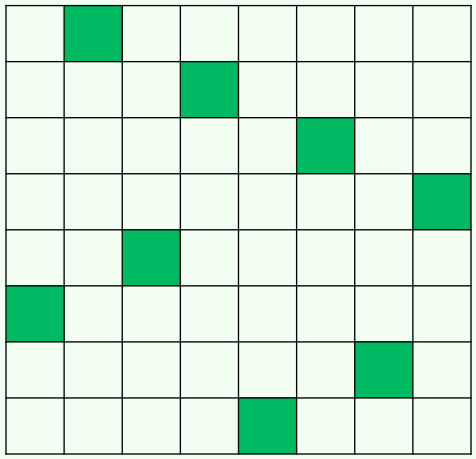
$$X = 2 \rightarrow Y \neq 4.$$

- Again, symmetrical conclusions should be inferred. Given a symmetry s, we could add on the right branch the constraint

$$s(X = 2 \rightarrow Y \neq 4)$$



Symmetry Breaking During Search



8-queens

Symmetry Breaking During Search

Some final comments on the SBDS method:

1. Contrary to adding constraints before search (e.g. lex-leader) SBDS does not require any change in the heuristics that are used.

2. Some CLP systems (e.g. Ecl^{i}ps^{e}) allow some provisions to break symmetries – user must supply some specification of them, in some adequate language.

3. Once a symmetry has been broken in some node, one should not be worried with it in any successor of this node. Hence implementations may add a bit map to each node to store the broken symmetries.

4. Although problems with thousands of symmetries have been addressed with SBDS, nevertheless SBDS has major problems to deal with very large numbers of symmetries. Some experiences have been made with a selection of a subset that removes most symmetries, but only with relatively small number of symmetries.

Symmetry Breaking with Stabilisers - STAB

- Symmetry Breaking Using Stabilizers (STAB), like SBDS, adds symmetry breaking constraints during search.
- Unlike SBDS, which places constraints to break **all** the symmetry of the problem, STAB places symmetry breaking constraints only for symmetries that leave the partial assignment A at the current node unchanged. That is, instead of breaking symmetry in the whole group, STAB breaks symmetry in the stabiliser $\mathbf{G_A}$.
- This means that much less symmetries are required to break, since stabilisers are much less than the whole set of symmetries in a group.
- STAB constraints take the form of lexicographic ordering constraints at each node A.

$$V \leq_{lex} g(V) \text{ for all } g \in G_A$$

- These constraints remove all the solutions that are not lexicographically minimum with respect to the stabiliser G_A in the sub tree rooted at A.

STAB by example

Example: Consider a 4×5 matrix model as shown below, where each symmetry for A is defined by a row permutation and a column permutation, and a partial assignment also shown.

x1	x2	x3	x4	x5
x6	x7	x8	x9	x10
x11	x12	x13	x14	x15
x16	x17	x18	x19	x20

0	0	0	1	1
0	1	1	0	0
x11	x12	x13	x14	x15
x16	x17	x18	x19	x20

- The symmetry obtained by composing row symmetry (1,2), swapping rows 1 and 2, and columns permutation (2,4,3,5), leaves the assigned variables unchanged.
- .

x6	x9	x10	x8	x7
x1	x4	x5	x3	x2
x11	x14	x15	x13	x12
x16	x19	x20	x18	x17

0	0	0	1	1
0	1	1	0	0
x11	x14	x15	x13	x12
x16	x19	x20	x18	x17

Symmetry Breaking with Stabilisers - STAB

- The lexical symmetry breaking constraint, allowing for the assignment A, is then

[0, 0, 0, 1, 1, 0, 1, 1, 0, 0, x11, x12, x13, x14, x15, x16, x17, x18, x19, x20]

$\leq_{lex}$

[0, 0, 0, 1, 1, 0, 1, 1, 0, 0, x11, x14, x15, x13, x12, x16, x19, x20, x18, x17]

0	0	0	1	1
0	1	1	0	0
x11	x12	x13	x14	x15
x16	x17	x18	x19	x20

0	0	0	1	1
0	1	1	0	0
x11	x14	x15	x13	x12
x16	x19	x20	x18	x17

- As the two vectors have the same first 10 elements, the constraint can be simplified into:

[x11, x12, x13, x14, x15, x16, x17, x18, x19, x20]

$\leq_{lex}$

[x11, x14, x15, x13, x12, x16, x19, x20, x18, x17].

- Notice that unlike SBDS, this is an incomplete method, since it only breaks the symmetries described by the stabilisers.

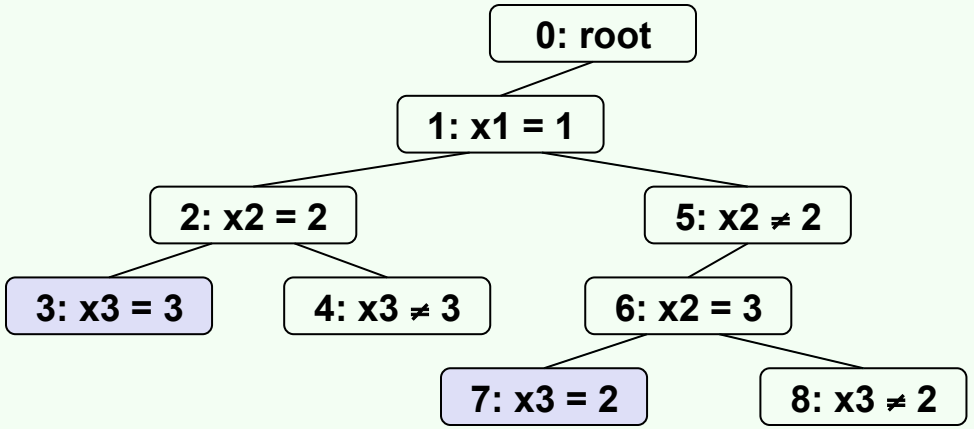
SBDD by example

- We now focus our attention to another method for dynamic symmetry breaking: Symmetry Breaking via Dominance Detection (SBDD).

- Contrary to SBDS, SBDD does not add any constraints during search. Instead it checks whether a node ready to be expanded is **dominated** by a previously expanded node, in which case, the expansion is aborted. We can illustrate the method with a very simple example

Example: Find different values for variables x1, x2, x3, with domains {1,2,3,4}.

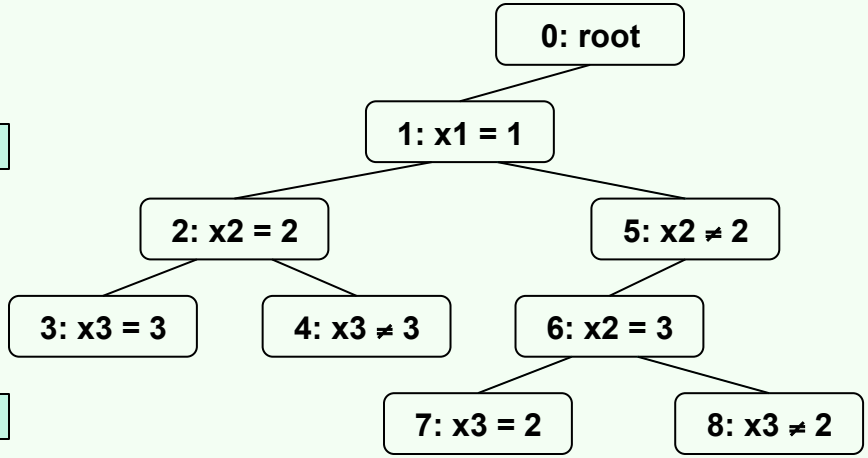
For this example, a search tree can be represented as shown, each node n:δ where n is the ordering of visit and δ the decision made.



SBDD by example

- In the problem, there are several variable symmetries (6 in all). As such, the table shows that solutions in nodes 3 and 7 are symmetrical, corresponding to permute variables x2 and x3.

node	x1	x2	x3	sol
0	1 2 3 4	1 2 3 4	1 2 3 4	
1	1	2 3 4	2 3 4	
2	1	2	3 4	
3	1	2	3	y
4	1	2	4	y
5	1	3 4	2 3 4	
6	1	3	2 4	
7	1	3	2	y
8	1	3	4	y



- These solutions would not be generated if we could detect that node 7 is **dominated** by node 2. In fact, assuming that node 2 is fully explored, all solutions with values 1 and 2 in the variables are found, and the solution in node 7, that restricts x3 to value 3, is just a repetition (modulo symmetry) of some node previously found by expansion of node 2 (i.e. node 3).

Symmetry Breaking via Dominance Detection

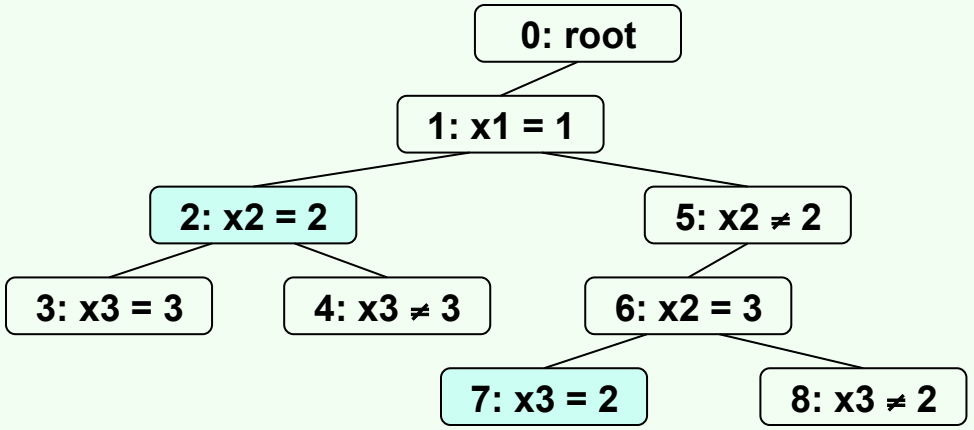
- To formalise SBDD the dominance property is tested between certain subsets of the tree nodes. These are captured by the notion of

- **Definition: Nogood**

  In a search tree as described, a node **n** is a no-good wrt. node **m** iff n is the left hand child of some node m_a, an ancestor of m, and n is not an ancestor of m.

In the figure:

1. Node 3 is a nogood wrt node 4.
2. Node 7 is a nogood wrt node 8
3. Node 2 is a nogood wrt nodes 5,6,7 and 8.



- **Note:** In the context of SBDD a nogood does not mean a failed node. Rather, it aims at defining dominant nodes, avoiding the expansion of dominated nodes.

Symmetry Breaking via Dominance Detection

- Being based in the notion of dominance, SBDD requires a clear definition of this concept. The most obvious is based on state inclusion.

- **Definition: Dominance (by state inclusion)**

  A node m is dominated if there is some node n, that is a nogood wrt m, and a symmetry s such that the domains of the variables in s(n) contain the domains of the variables in m.

- It is clear from this definition that for any solution obtained by expansion of node m, there will be a symmetrical solution obtained by expansion of node n, whose variable domains include (modulo symmetry) the domains of the variables in m.

- Nevertheless, an alternative definition allows a more efficient detection of dominance, based on sets of decisions.