AUTOMATED REASONING

Agostino Dovier

Università di Udine
CLPLAB

Udine, October 2016

Constraint Solving

As seen in DPLL for SAT, solution search alternates two stages: non-deterministic choices and constraint propagation (aiming at reaching some form of local consistency).

- ♡ A variable is chosen (using a suitable heuristics) and its domain is reduced (typically, by chhosing it a value, but other possibilities are considered e.g., splitting the value in two parts, etc.)
- Alternatively (less frequent in implementations) a constraint is chosen and split
- Then propagation is applied
- If a domain become empty (fail) backtracking!
- If there are no longer uninstantiated variables, the solution is returned.
- Otherwise, go to ♡

Size and shape of the search tree depend on variable selection, on the kind of propagation used, on the kind of assignment/domain reduction rules employed.

1. (domain) labeling:

$$\frac{X \in \{a_1, \dots, a_k\}}{X \in \{a_1\} | \cdots | X \in \{a_k\}}$$

2. (domain) enumeration:

$$\frac{X \in \mathcal{D}}{X \in \{a\} | X \in \mathcal{D} \setminus \{a\}}$$

where $a \in \mathcal{D}$

3. (domain) bisection:

$$\frac{X \in \mathcal{D}}{X \in \min(\mathcal{D})..a | X \in b..\max(\mathcal{D})}$$

where $a, b \in \mathcal{D}$, and b is the element following a in $\mathcal{D}$. If $\mathcal{D}$ is an interval $x..y$ choose $a = \lfloor (x + y)/2 \rfloor$ and $b = a + 1$.

1. implicazione:

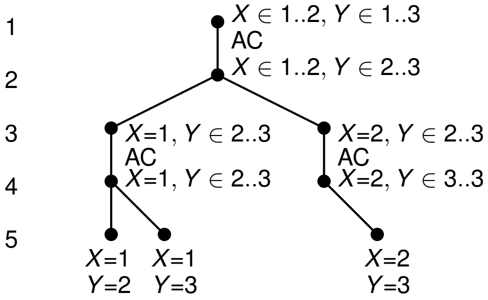
$$\frac{(C_1 \rightarrow C_2)}{\neg C_1 | C_2}$$

2. Absolute value:

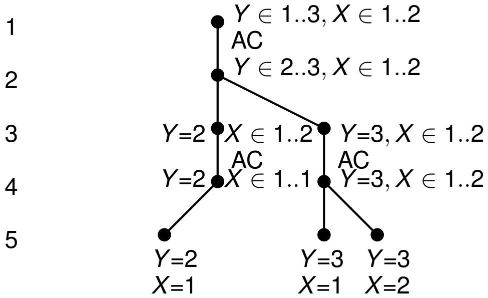
$$\frac{|e| = X}{X = e | X = -e}$$

3. Inequality:

$$\frac{e_1 \neq e_2}{e_1 < e_2 | e_2 < e_1}$$



prop-labeling-tree for $\mathcal{P} = \langle X < Y; X \in \{1, 2\}, Y \in \{1, 2, 3\}\rangle$.



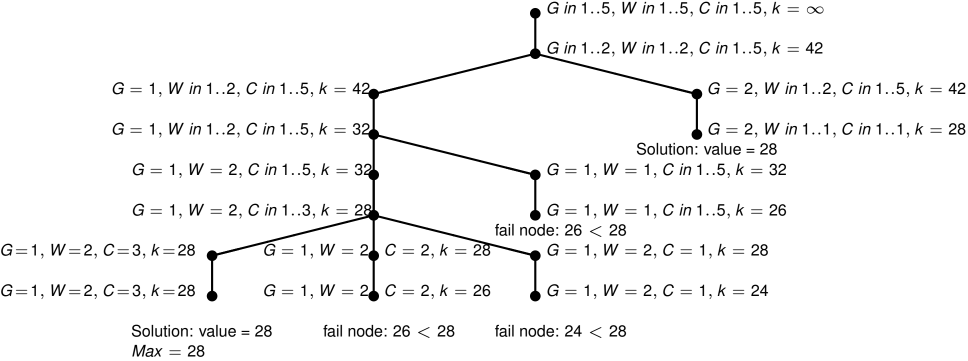
prop-labeling-tree for $\mathcal{P} = \langle X < Y; X \in \{1, 2\}, Y \in \{1, 2, 3\}\rangle$.

- The construction and the visit of the prop-labeling-tree is called by calling a built-in (`labeling` in CLPFD, `solve` in Minizinc).
- Every constraint solver has a set of parameters
- How choosing a variable (leftmost, ff, etc)
- How choosing the value in the domain (min, max, med, etc)
- Other parameters (approximated search, LNS, timeout etc)

Constraint Solving

Branch and Bound for COP

$\mathcal{C} = 17G + 10W + 4C < 50,\ f(W, G, C) = 10G + 6W + 2C$



MINIZINC

- Minizinc is defined, implemented and mantained by NICTA
- You can download it from `http://www.minizinc.org/`
- Youll find a tutorial by Marriott and Stuckey
- Typically, a Minizinc model is first translated to Flatzinc using `mzn2fzn`
- A Flatzinc model is an unfolded version of the Minizinc one; basically it is a sequence of simple (flat) constraints
- Any modern constraint solver reads Flatzinc models as input (Minizinc challenge is organized yearly since 2008)

- Variables (and parameters/constants) need to be typed. E.g.
  ```
  par int:  a = 3;
  var int:  b;
  ```
  Parameters should be assigned asap and are assigned once. `par` is the default value. `var` should be made explicit.
- Possible types for var/par are (plus string):
  - INT: integer variables (e.g. FD)
  - BOOL: Boolean variables (particular cases of FD)
  - FLOAT: floating point variables (for hybrid modeling)
- A variable should be assigned to a domain. E.g.,
  `var 0..100:v;` for intervals domain (typical case)
  `var {0,2,4,6}:w;` for explicitly listed domains

You can define single/multi-dimensional arrays of variables:

- `array [ indexset1, indexset2, . . . ] of var type: varname;`
- For instance:
  `array [0..2] of var 1..5 : v;`
  `array [1..5,1..5] of var 0..2 : M;`
  arrays are accessed as `V[i]`, `M[i,j]`.
- Set of integers as domains are allowed.
  `set of 1..8 : s;`
  `s` is any subset of $\{1, \ldots, 8\}$. You can use membership (in), set inclusion (subset, superset), union (union), intersection (inter), set difference (diff), symmetric difference (symdiff) and cardinality (card) to build expressions with set variables.

Constraints are added explicitly either in a flat or compact way. E.g.,

- `constraint a + b < 100;`
- `constraint a\/b;` (this means $a \vee b$ for Boolean variables)
- `constraint alldifferent(V);` (where V is an array of variables: the global constraint should be imported using import ... more details in next lessons)
- constraint `forall(EXPRESSION);` (where EXPRESSION is a complex statement, such as a list comprehension). E.g. `forall( [ v[ i ] != v[ j ] | i , j in 1..3 where i < j ] );` (You should read the manual for the syntax of EXPRESSIONs, of course)
- There is a simplified, user-friendly version:

```
forall(i,j in 1..3 where i<j)
        (v[ i ] != v[ j ]);
```

You can choose the built-in search directives:

- solve satisfy;
- solve maximize(⟨Arithmetic EXPRESSION⟩);
- solve minimize(⟨Arithmetic EXPRESSION⟩);

Example of expressions can be a single variable or a function.

```
int_search(variables, varchoice,
               constrainchoice, strategy)
```

- `variables` is an one dimensional array of var int,
- `varchoice` is a variable choice annotation
- `constrainchoice` is a choice of how to constrain a variable
- `strategy` is a search strategy (for now use `complete`).

- `input_order` Search variables in the given order
- `occurrence` Choose the variable with the largest number of attached constraints
- `first_fail/anti_first_fail` Choose the variable with the smallest/largest domain
- `most_constrained` Choose the variable with the smallest domain, breaking ties using the number of attached constraints
- `dom_w_deg` Choose the variable with largest domain, divided by the number of attached constraints weighted by how often they have caused failure
- `impact` Choose the variable with the highest impact so far during the search
- `max_regret` Choose the variable with largest difference between the two smallest values in its domain
- `smallest/largest` Choose the variable with the smallest/larger value in its domain

- `indomain` Assign values in ascending order
- `indomain_interval` If the domain consists of several contiguous intervals, reduce the domain to the first interval. Otherwise bisect the domain.
- `indomain_max` Assign the largest value in the domain
- `indomain_median` Assign the middle value in the domain
- `indomain_middle` Assign the value in the domain closest to the mean of its current bounds
- `indomain_min` Assign the smallest value in the domain
- `indomain_random` Assign a random value from the domain

- `indomain_reverse_split` Bisect the domain, excluding the lower half first
- `indomain_split` Bisect the domain, excluding the upper half first
- `indomain_split_random` Bisect the domain, randomly selecting which half to exclude first
- `outdomain_max` Exclude the largest value from the domain
- `outdomain_median` Exclude the middle value from the domain
- `outdomain_min` Exclude the smallest value from the domain
- `outdomain_random` Exclude a random value from the domain