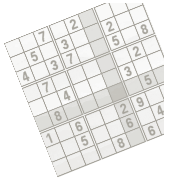

Constraint Programming

Practical Exercises

Roman Barták

Department of Theoretical Computer Science and Mathematical Logic

Design of filtering algorithms

Today program

We will look inside constraint solvers.

- **Design of filtering algorithms**
  - **reification**: design of meta-constraints
  - **indexicals**: design of primitive constraints
  - **global constraints**: design of complex constraints

Reification

- We can set satisfaction/violation of certain constraints.
- Implemented via equivalence and a Boolean variable

  `Constraint #<=> B`

  Example:

  `X#>5 #<=> B` //no change of domains
  - after adding **`X#<3`** we get X in inf..2 and B=0
  - after adding **`X#>8`** we get X in 9..sup and B=1
  - after adding **`B=1`** we get X in 6..sup

  `Constraint` must be **reifiable**, i.e., it can be used in logical constraints (arithmetical constraints are reifiable while global constraints are usually not reifiable).

Reification for "new" constraints

`exactly(N,List,X)`

`N` is a FD variable, **`List`** is a list of FD variables, and **`X`** is a FD variable

Semantics:

exactly **`N`** variables from the list **`List`** equals to **`X`**

Implementation using reification:

```
exactly(0, [], _X).
exactly(N, [Y|L], X) :-
  X #= Y #<=> B,
  N #= M+B,
  exactly(M, L, X).
```

Recall: arc consistency loop

We can propagate information to a set of variables, not just to one variable Y.

```
procedure GAC(G)
    Q ← {Xs →Y | Xs →Y is a method for some constraint in G}
    while Q non empty do
        select and delete (As→B) from Q
        if REVISE(As→B) then
            if $D_B=\emptyset$ then stop with fail
            Q ← Q ∪ {Xs →Y | Xs →Y is a method s.t. B∈Xs}
        end if
    end while
end GAC-3
```

We can decide which change of the domain of B will invoke constraint filtering.

Filtering algorithms

News constraints are defined via the REVISE procedures.

How to do it?

1) We need to decide the event for **constraint invocation**.
- when the domain of some variable is changed (suspensions)
  - whenever the domain changes
  - when the domain bounds are changed
  - when the domain becomes singleton
- it is possible to use different suspensions for different variables

  Example:
    - A<B is invoked when min(A) and max(B) change
  - This way we can even define directional consistency or forward checking!

2) We need to write the **filtering procedure**.
- the output is the suggestion of new domains
- there could be more filtering procedures for a single constraint

Example: A<B
  - min(A): B in min(A)+1..sup
  - max(B): A in inf..max(B)-1

Indexicals: primitive constraints

- We can define new primitive constraints in a style similar to Prolog using "reactive" rules called **indexicals**
- There are rules for positive and negative version of each constraint and for verification of satisfaction/violation of the constraint:
  - Head +: Indexicals.
  - Head -: Indexicals.
  - Head +? Indexical.
  - Head -? Indexical.
- Such constraints are reifiable!

Primitive constraints: filtering example

Bounds consistency

```
plus(X,Y,T) +:
    X in min(T) - max(Y) .. max(T) - min(Y),
    Y in min(T) - max(X) .. max(T) - min(X),
    T in min(X) + min(Y) .. max(X) + max(Y).
```

Arc consistency

```
plusd(X,Y,T) +:
    X in dom(T) - dom(Y),
    Y in dom(T) - dom(X),
    T in dom(X) + dom(Y).
```

X+Y #= T

Indexicals

- Description of how the domain of the variable is changed using the form **X in R**.
  - *processing domains*
    - dom(X), {T1,…,Tn}, T1..T2
    - R1 /\ R2, R1 \/ R2, \R1, R1+R2, R1-R2
    - …
  - *using terms*
    - min(X), max(X), card(X)
    - X (wait until X is bound), I (integer), inf, sup
    - T1+T1, T1+T2, T1*T2, T1 mod T2, T1 rem T2
    - …

Primitive constraints: complex example

```
’x\\=y’(X,Y) +:
        X in \{Y},
        Y in \{X}.
’x\\=y’(X,Y) -:
        X in dom(Y),
        Y in dom(X).
’x\\=y’(X,Y) +?
        X in \dom(Y).
’x\\=y’(X,Y) -?
        X in {Y}.
```

propagation for the satisfied constraint

propagation for the violated constraint

verification of the satisfied constraint

verification of the violated constraint

Domain access

How to access the values in variables‘ domains?

`fd_min(?X, ?Min)`
- Min is unified with the smallest value in the domain of X (it could be inf)

`fd_max(?X, ?Max)`
- Max is unified with the largest values in the domain of X (it could be sup)

`fd_size(?X, ?Size)`
- Size is unified with the number of values in the domain (it could be sup)

`fd_set(?X, ?Set)`
- Set is unified with the representation of the domain of X

`fd_degree(?X, ?Degree)`
- Degree is unified with the number of constraints over X

Domain properties

- `empty_fdset(?Set)`
- `fdset_min(+Set, -Min)`
- `fdset_max(+Set, -Min)`
- `fdset_subset(+Set1, +Set2)`
- `fdset_disjoint(+Set1, +Set2)`
- `fdset_intersect(+Set1, +Set2)`
- `fdset_eq(+Set1, +Set2)`
- `fdset_member(?Elt, +Set)`

Domain modification/transformation

- `fdset_add_element(`**`+Set1, +Elt, -Set2`**`)`
- `fdset_del_element(`**`+Set1, +Elt, -Set2`**`)`
- `fdset_intersection(`**`+Set1, +Set2, -Intersection`**`)`
- `fdset_subtract(`**`+Set1, +Set2,-Difference`**`)`
- `fdset_union(`**`+Set1, +Set2, -Union`**`)`
- `fdset_complement(`**`+Set, -Complement`**`)`
- `fdset_parts(`**`?Set, ?Min, ?Max, ?Rest`**`)`

- `list_to_fdset(`**`+List, -Set`**`)`
- `fdset_to_list(`**`+Set, -List`**`)`
- `range_to_fdset(`**`+Range, -Set`**`)`
- `fdset_to_range(`**`+Set, -Range`**`)`

Global constraints

- **Constraint initialization**
  - fd_global(**:Constraint**, +**State**, +**Susp**)
    - Constraint – term describing the constraint
    - State – an initial state for the filtering algorithm
    - Susp – a list of suspensions
      - dom(**X**), min(**X**), max(**X**), minmax(**X**), val(**X**)
- **Constraint definition – filtering algorithm**
  - clpfd:dispatch_global(**+Constraint**, **+State0**, **-State**, **-Actions**)
    - filtering algorithm describing how to modify the domains
      - exit, fail, X = V, X in R, X in_set S, call(Goal)

Global constraints: "less than"

How to describe a filtering procedure for A<B?

Note: bounds consistency is equivalent to AC for A<B!

```
less_then(A,B):-
   fd_global(a2b(A,B),no_state,[min(A)]),
   fd_global(b2a(A,B),no_state,[max(B)]).

:-multifile clpfd:dispatch_global/4.
clpfd:dispatch_global(a2b(A,B),S,S,Actions):-
   fd_min(A,MinA), fd_max(A,MaxA), fd_min(B,MinB),
   (MaxA<MinB ->
    Actions = [exit]
   ;   LowerBoundB is MinA+1,
    Actions = [B in LowerBoundB..sup]).
clpfd:dispatch_global(b2a(A,B),S,S,Actions):-
   fd_max(A,MaxA), fd_min(B,MinB), fd_max(B,MaxB),
    (MaxA<MinB ->
    Actions = [exit]
   ;   UpperBoundA is MaxB-1,
    Actions = [A in inf..UpperBoundA]).
```

A#<B

Global constraints: "diff"

How to describe a filtering procedure for A≠B?

Idea: Constraint is **consistent** if domains of both variables contain two or more values! Hence any filtering is useful only if any **domain becomes singleton**.

```
diff(A,B):-
  fd_global(diff(A,B),no_state,[val(A)]),
  fd_global(diff(B,A),no_state,[val(B)]).

:-multifile clpfd:dispatch_global/4.
clpfd:dispatch_global(diff(X,Y),S,S,Actions):-
  (ground(X) ->
   fd_set(Y,SetY),
   fdset_del_element(SetY,X,NewSetY),
   Actions = [exit, Y in_set NewSetY]
  ;
   Actions = []
  ).
```

A#\=B

Global constraints: “all-diff”

How to find out that each variable in a list has a value different from all other variables?

Idea: If we assign a value to some variable (its domain becomes singleton), then this value is deleted from the domains of other variables.

```
all_diff(List):-
   start_all_diff(List,List).
start_all_diff([],_).
start_all_diff([H|T],List):-
   fd_global(all_diff(H,T,List),no_state,[val(H)]),
   start_all_diff(T,List).

:-multifile clpfd:dispatch_global/4.
clpfd:dispatch_global(all_diff(X,Pointer,List),S,S,Actions):-
   (ground(X) -> % a value has been assigned to X
       filter_diff(List,X,Pointer, Actions)
   ;
       Actions = []
   ).

filter_diff([],_X,_Pointer, [exit]).
filter_diff([Y|T],X,Pointer, Actions):-
   (T==Pointer -> % identical objects
       Actions = RestActions
   ;
       fd_set(Y,SetY),
       fdset_del_element(SetY,X,NewSetY),
       Actions = [Y in_set NewSetY | RestActions]
   ),!,
   filter_diff(T,X,Pointer, RestActions).
```

all_different(List)

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz

Diff vs. all-diff

All-diff for N variables can also be described using N.(N-1)/2 constraints diff.

Which approach is better?

- **Filtering power**
  - both our models remove exactly the same inconsistent values
  - **all-distinct** removes more inconsistencies by global reasoning
- **Time efficiency**
  - all-diff is faster than a set of diff constraints

  Example:

  filling partial Latin square of order 20 with 8 prefilled cells

  - all-diff 0.68s, diff 1.43s

Latin square of order N is a matrix of size N×N filled by values {1,...,N} such that values in each row (and column) are different. **Partial Latin square** has some cells pre-filled.

4	1	3	2
1	4	2	3
2	3	4	1
3	2	1	4