Global Constraints

Toby Walsh
NICTA and University of New South Wales
www.cse.unsw.edu.au/~tw

Quick advert

- UNSW is in Sydney
  - Regularly voted in top 10 cities in World
- UNSW is one of top universities in Australia
  - In top 100 universities in world
- Talk to me about our PhD programme!
  - Also happy to have PhDs/PostDocs visit for weeks/months/years …
  - Attend CP/KR/ICAPS in Sept

Value precedence

- Global constraint used to deal with value symmetry
- Good example of "global" constraint where we can use an efficient encoding
  - Encoding gives us GAC
  - Asymptotically optimal, achieve GAC in O(nd) time
  - Good incremental/decremental complexity

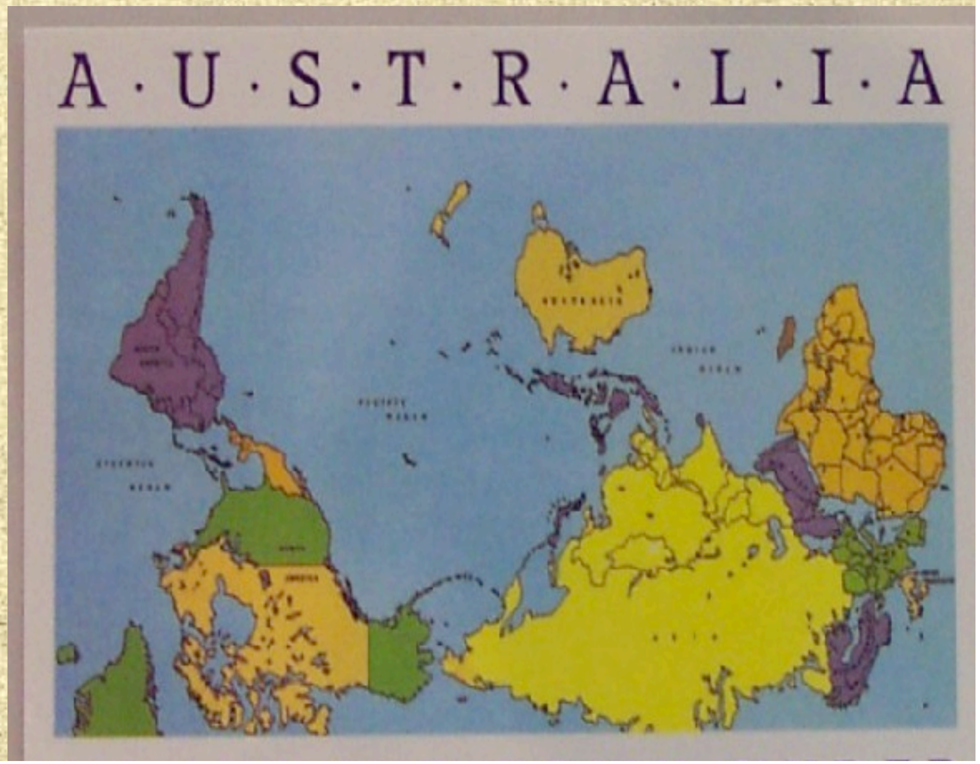
Value symmetry

- Decision variables:
  - Col[Italy], Col[France], Col[Austria] ...
- Domain of values:
  - red, yellow, green, ...
- Constraints

  Col[Italy]=/=Col[France]

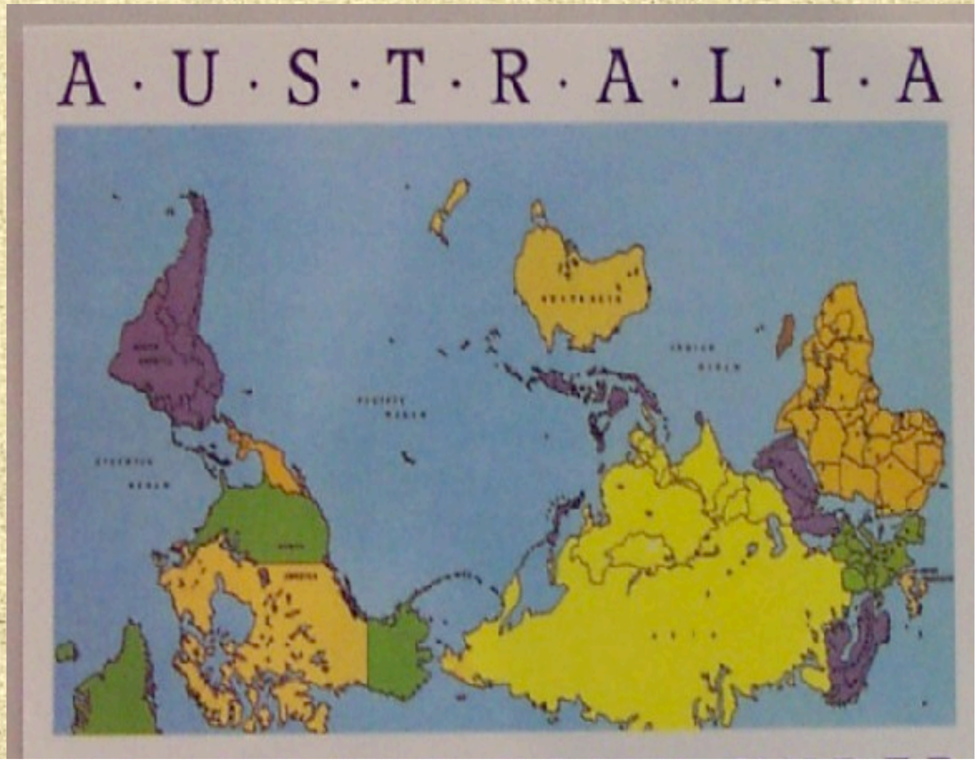
  Col[Italy]=/=Col[Austria]

  ...



Value symmetry

- Solution:
  - Col[Italy]=green
    Col[France]=red
    Col[Spain]=green
    ...
- Values (colours) are interchangeable:
  - Swap red with green everywhere will still give us a solution



Value precedence

- Old idea
  - Used in bin-packing and graph colouring algorithms
  - Only open the next *new* bin
  - Only use one *new* colour
- Applied now to constraint satisfaction

Value precedence

- Suppose all values from 1 to *m* are interchangeable
  - Might as well let X1=1

Value precedence

- Suppose all values from 1 to *m* are interchangeable
  - Might as well let X1=1
  - For X2, we need only consider two choices
    - X2=1 or X2=2

Value precedence

- Suppose all values from 1 to *m* are interchangeable
  - Might as well let X1=1
  - For X2, we need only consider two choices
  - Suppose we try X2=2

Value precedence

- Suppose all values from 1 to *m* are interchangeable
  - Might as well let X1=1
  - For X2, we need only consider two choices
  - Suppose we try X2=2
  - For X3, we need only consider three choices
    - X3=1, X3=2, X3=3

Value precedence

- Suppose all values from 1 to *m* are interchangeable
  - Might as well let X1=1
  - For X2, we need only consider two choices
  - Suppose we try X2=2
  - For X3, we need only consider three choices
  - Suppose we try X3=2

Value precedence

- Suppose all values from 1 to *m* are interchangeable
  - Might as well let X1=1
  - For X2, we need only consider two choices
  - Suppose we try X2=2
  - For X3, we need only consider three choices
  - Suppose we try X3=2
  - For X4, we need only consider three choices
    - X4=1, X4=2, X4=3

Value precedence

- Global constraint
  - Precedence([X1,..Xn]) iff

    $\min(\{i \mid Xi=j \text{ or } i=n+1\}) < \min(\{i \mid Xi=k \text{ or } i=n+2\})$ for all $j<k$
  - In other words
    - The first time we use j is before the first time we use k

Value precedence

- Global constraint
  - Precedence([X1,..Xn]) iff

    $\min(\{i \mid X_i = j \text{ or } i = n+1\}) <$

    $\min(\{i \mid X_i = k \text{ or } i = n+2\})$
  - E.g
    - Precedence([1,1,2,1,3,2,4,2,3])
    - But not Precedence([1,1,2,1,4])

Value precedence

- Global constraint
  - Precedence([X1,..Xn]) iff

    $$\min(\{i \mid Xi=j \text{ or } i=n+1\}) < \min(\{i \mid Xi=k \text{ or } i=n+2\})$$

  - Propagator proposed by [Law and Lee 2004]
    - Pointer based propagator (alpha, beta, gamma) but only for two interchangeable values at a time

Value precedence

- Precedence([j,k],[X1,..Xn]) iff

  $$\min(\{i \mid X_i=j \text{ or } i=n+1\}) < \min(\{i \mid X_i=k \text{ or } i=n+2\})$$

- Of course
  - Precedence([X1,..Xn]) iff Precedence([i,j],[X1,..Xn]) for all $i<j$
  - Precedence([X1,..Xn]) iff Precedence([i,i+1],[X1,..Xn]) for all i

Value precedence

- Precedence([j,k],[X1,..Xn]) iff

  min({i | Xi=j or i=n+1}) <

  min({i | Xi=k or i=n+2})

- Of course
  - Precedence([X1,..Xn]) iff Precedence([i,j],[X1,..Xn]) for all i<j
- But this hinders propagation
  - GAC(Precedence([X1,..Xn])) does strictly more pruning than GAC(Precedence([i,j],[X1,..Xn])) for all i<j
  - Consider

    X1=1, X2 in {1,2}, X3 in {1,3} and X4 in {3,4}

Puget's method

- Introduce Z_j to record first time we use j
- Add constraints
  - $X_i=j$ implies $Z_j <= i$
  - $Z_j=i$ implies $X_i=j$
  - $Z_i < Z_{i+1}$

Puget's method

- Introduce Z_j to record first time we use j
- Add constraints
  - $X_i=j$ implies $Z_j < i$
  - $Z_j=i$ implies $X_i=j$
  - $Z_i < Z_{i+1}$
- Binary constraints
  - easy to implement

Puget's method

- Introduce Zj to record first time we use j
- Add constraints
  - Xi=j implies Zj < I
  - Zj=i implies Xi=j
  - Zi < Zi+1
- Unfortunately hinders propagation
  - AC on encoding may not give GAC on Precedence([X1,..Xn])
  - Consider X1=1, X2 in {1,2}, X3 in {1,3}, X4 in {3,4}, X5=2, X6=3, X7=4

Propagating Precedence

- Simple ternary encoding
- Introduce sequence of variables, Yi
  - Record largest value used so far
  - Y1=0

Propagating Precedence

- Simple ternary encoding
- Post sequence of constraints

$C(X_i, Y_i, Y_{i+1})$ for each $1 \le i \le n$

These hold iff

$X_i \le 1 + Y_i$ and $Y_{i+1} = \max(Y_i, X_i)$

Propagating Precedence

- Simple ternary encoding
- Post sequence of constraints
- Easily implemented within most solvers
    - Implication and other logical primitives
    - GAC-Schema (alias “table” constraint)
    - …

Propagating Precedence

- Simple ternary encoding
- Post sequence of constraints
  - C(Xi,Yi,Yi+1) for each 1<=i<=n
  - This decomposition is Berge-acyclic
  - Constraints overlap on one variable and form a tree

Propagating Precedence

- Simple ternary encoding
- Post sequence of constraints
  - C(Xi,Yi,Yi+1) for each 1<=i<=n
  - This decomposition is Berge-acyclic
  - Constraints overlap on one variable and form a tree
  - Hence enforcing GAC on the decomposition achieves GAC on Precedence([X1,..Xn])
  - Takes O(n) time
  - Also gives excellent incremental behaviour

Propagating Precedence

- Simple ternary encoding
- Post sequence of constraints
  - C(Xi,Yi,Yi+1) for each 1<=i<=n
  - These hold iff Xi<=1+Yi and Yi+1=max(Yi,Xi)

  - Consider Y1=0, X1 in {1,2,3}, X2 in {1,2,3} and X3=3

Precedence and matrix symmetry

- Alternatively, could map into 2d matrix
  - Xij=1 iff Xi=j
- Value precedence now becomes column symmetry
  - Can lex order columns to break all such symmetry
  - Alternatively view value precedence as ordering the columns of a matrix model

Precedence and matrix symmetry

- Alternatively, could map into 2d matrix
  - Xij=1 iff Xi=j
- Value precedence now becomes column symmetry
- However, we get less pruning this way
  - Additional constraint that rows have sum of 1
  - Consider, X1=1, X2 in {1,2,3} and X3=1

Partial value precedence

- Values may partition into equivalence classes
  - Values within each equivalence class are interchangeable
- E.g.

  Shift1=nursePaul, Shift2=nursePeter, Shift3=nurseJane, Shift4=nursePaul ..

Partial value precedence

- Shift1=nursePaul, Shift2=nursePeter, Shift3=nurseJane, Shift4=nursePaul ..
- If Paul and Jane have the same skills, we can swap them (but not with Peter who is less qualified)
  - Shift1=nurseJane, Shift2=nursePeter, Shift3=nursePaul, Shift4=nurseJane ...

Partial value precedence

- Values may partition into equivalence classes
- Value precedence easily generalized to cover this case
  - Within each equivalence class, vi occurs before vj for all i<j (ignore values from other equivalence classes)

Partial value precedence

- Values may partition into equivalence classes
- Value precedence easily generalized to cover this case
  - Within each equivalence class, vi occurs before vj for all i<j (ignore values from other equivalence classes)
  - For example
    - Suppose vi are one equivalence class, and ui another

Partial value precedence

- Values may partition into equivalence classes
- Value precedence easily generalized to cover this case
  - Within each equivalence class, vi occurs before vj for all i<j (ignore values from other equivalence classes)
  - For example
    - Suppose vi are one equivalence class, and ui another
    - X1=v1, X2=u1, X3=v2, X4=v1, X5=u2

Partial value precedence

- Values may partition into equivalence classes
- Value precedence easily generalized to cover this case
  - Within each equivalence class, vi occurs before vj for all i<j (ignore values from other equivalence classes)
  - For example
    - Suppose vi are one equivalence class, and ui another
    - X1=v1, X2=u1, X3=v2, X4=v1, X5=u2
    - Since v1, v2, v1 … and u1, u2, …

Variable and value precedence

- Value precedence compatible with other symmetry breaking methods
  - Interchangeable values and lex ordering of rows and columns in a matrix model

Conclusions

- Symmetry of interchangeable values can be broken with value precedence constraints
- Value precedence can be decomposed into ternary constraints
  - Efficient and effective method to propagate
- Can be generalized in many directions
  - Partial interchangeability, …

Global constraints

- Hardcore algorithms
  - Data structures
  - Graph theory
  - Flow theory
  - Combinatorics
  - …
- Computational complexity
  - Global constraints are often balanced on the limits of tractability!

Computational complexity 101

- Some problems are essentially easy
  - Multiplication, $O(n^{1.58})$
  - Sorting, $O(n \log n)$
  - Regular language membership, $O(n)$
  - Context free language membership, $O(n^3)$ ..
- P (for "polynomial")
  - Class of decision problems recognized by deterministic Turing Machine in polynomial number of steps
- Decision problem
  - Question with yes/no answer? E.g. is this string in the regular language? Is this list

NP

- NP
  - Class of decision problems recognized by non-deterministic Turing Machine in polynomial number of steps
  - Guess solution, check in polynomial time
  - E.g. is propositional formula Ψ satisfiable? (SAT)
    - Guess model (truth assignment)
    - Check if it satisfies formulae in polynomial time

NP

- Problems in NP
  - Multiplication
  - Sorting
  - ..
  - SAT
  - 3-SAT
  - Number partitioning
  - K-Colouring
  - Constraint satisfaction
  - …

NP-completeness

- Some problems are computationally as hard as any problem in NP
  - If we had a fast (polynomial) method to solve one of these, we could solve any problem in NP in polynomial time
  - These are the NP-complete problems
    - SAT (Cook's theorem: non-deterministic TM => SAT)
    - 3-SAT
    - ….

NP-completeness

- To demonstrate a problem is NP-complete, there are two proof obligations:
  - in NP
  - NP-hard (it’s as hard as anything else in NP)

NP-completeness

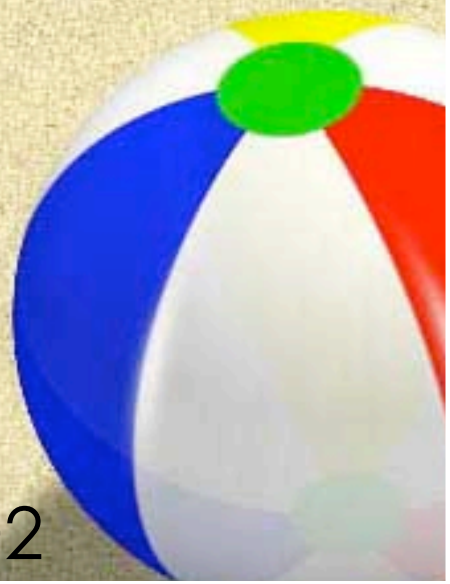
- To demonstrate a problem is NP-complete, there are two proof obligations:
  - in NP
    - Polynomial witness for a solution
    - E.g. SAT, 3-SAT, number partitioning, k-Colouring, …
  - NP-hard (it’s as hard as anything else in NP)

NP-completeness

- To demonstrate a problem is NP-complete, there are two proof obligations:
  - NP-hard (it's as hard as anything else in NP)
    - Reduce some other NP-complete to it
    - That is, show how we can use our problem to solve some other NP-complete problem
    - At most, a polynomial change in size of problem

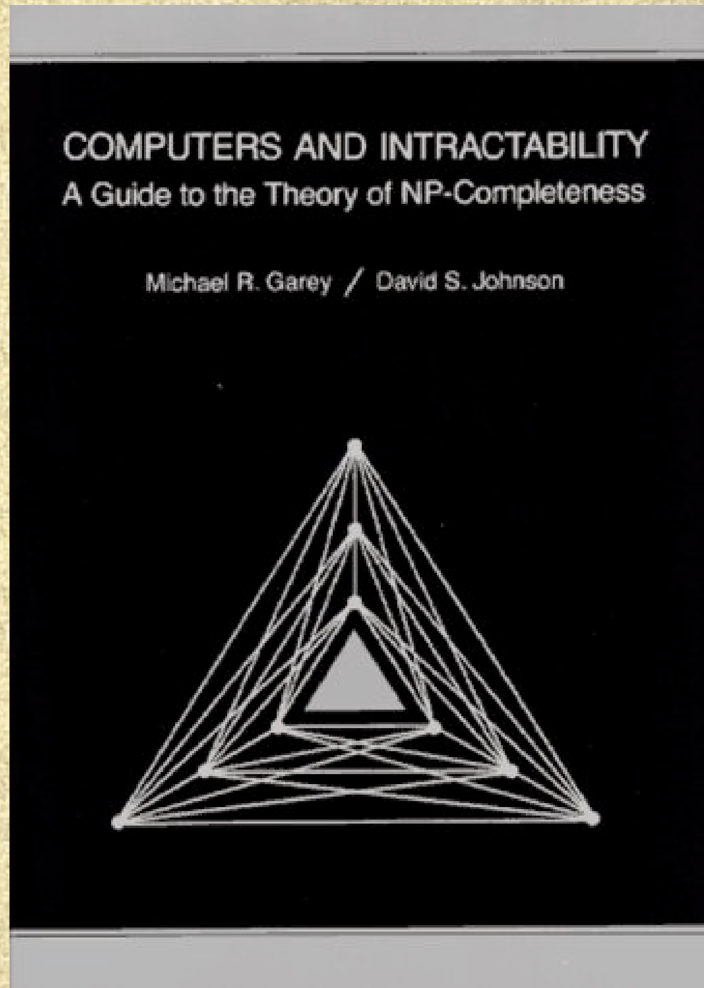
Global constraints are NP-hard

- Can solve 3SAT using a single global constraint!
  - Given 3SAT problem in N vars and M clauses
  - 3SAT([X1,…Xn]) where n=N+3M+2
  - Constraint holds iff X1=N, X2=M,
  - X_2+i is 0/1 representing value assigned to xi
  - X_2+N+3j, X_2+N+3j+1 and X_2+N+3j+2



Our hammer

- Use tools of computational complexity to study global constraints



Questions to ask?

- GACSupport? is NP-complete
  - Does this value have support?
  - Basic question asked within many propagators
- MaxGAC? is DP-complete
  - Are these domains the maximal generalized arc-consistent domains?
  - Termination test for a propagator
  - DP = NP u coNP
  - Propagation "harder" than solving

Questions to ask?

- IsItGAC? is NP-complete
  - Are the domains GAC?
  - Wakeup test for a propagator
- NoGACWipeOut? is NP-complete
  - If we enforce GAC, do we not get a wipeout?
  - Used in many reductions
- GACDomain? is NP-hard
  - Return the maximal GAC domains

Relationships between questions

- NoGACWipeOut = GACSupport = GACDomain
  - NoGACWipeOut in P <-> GACDomain in P
  - NoGACWipeOut in NP <-> GACDomain in NP

- GACDomain in P => MaxGAC in P => IsItGAC in P
- IsItGAC in NP => MaxGAC in NP => GACDomain in NP

Constraints in practice

- Some constraints proposed in the past are intractable
- NValues(N,X1,..Xn)
- CardPath(N,[X1,..,Xn],C)
- …

NValues

- NValues(N,X1,..,Xm)
- N values used in X1,...,Xm
- Useful for resource allocation

NValues

- NValues(Y,X1,..,Xn)

- Reduction of 3SAT to NValues
  - 3SAT problem in N vars, M clauses
  - Xi in {i,-i} for 1 ≤ i ≤ N
  - XN+s in {i,-j,k} if s-th clause is: (i or -j or k)
  - Y= N
  - Hence 3SAT has a solution => NoGACWipeOut answers “yes”

NValues

- NValues(N,X1,..,Xm)

- Reduction of 3SAT to NValues
- 3SAT problem in n vars, l clauses
- Xi in {i,-i} for $1 \leq i \leq n$
- Xn+s in {i,-j,k} if s-th clause is: (i or -j or k)
- N = n
- Hence 3SAT has a solution <=> NoGACWipeOut answers “yes”
- Enforce lesser level of local consistency (e.g. BC)

Generalizing constraints

- Take a tractable constraint
  - GCC([X1,...,Xn],[l1,...,lm],[u1,...,um])
  - Value j occurs between lj and uj times in X1,...,Xn
- Generalize some constants to variables
  - E.g. GCC([X1,...,Xn],[O1,...,Om])
  - NP-hard to enforce GAC!

Generalizing constraints

- GCC([X1,..,Xn],[O1,..,Om])
- Reduction from 1in3SAT on positive clauses
- If jth clause is (x or y or z) then Xj in {x,y,z}
- If x occurs k times in all clauses then Ox in {0,k}
- Hence 1in3SAT has a solution iff NoGACWipeOut answers “yes”
- Thus enforcing GAC is NP-hard

Meta-constraints

- Global constraint used in sequencing problems
- CardPath(C,[X1,..Xn],N) iff C(Xi,..Xi+k) holds N times
  - E.g. number of changes is CardPath(=/=,[X1,..Xn],N)
- Fixed parameter tractable
  - k fixed, GAC takes O(nd^k) time
  - k = O(n), GAC is NP-hard even when C is polynomial to test

Meta-constraints

- CardPath(C,[X1,..Xn],N) iff C(Xi,..Xi+k) holds N times
  - Reduce 3SAT in N variables and M clauses to CardPath where k=N+2
  - NM vars Xi to represent repeated truth assignment
  - M vars Yj to represent jth clause
  - C(X1,..,XN,Yj,X1') iff Yj=k and Xk=1 and X1=X1'

    or Yj=-k and Xk=0 and X1=X1'
  - C(X2,..,XM.Yj,X1',X2') iff X2=X2'

Conclusions

- Computational complexity is a useful hammer to study global constraints
- Uncovers fundamental limits of reasoning with global constraints
  - Lesser consistency needs to be enforced
  - Generalization intractable
  - ..

Global grammar constraints

- Often easy to specify a global constraint
  - ALLDIFFERENT([X1,..Xn]) iff
    Xi=/=Xj for i<j
- Difficult to build an efficient and effective propagator
  - Especially if we want global reasoning

Global grammar constraints

- Promising direction initiated is to specify constraints via automata/grammar

  - Sequence of variables = string in some formal language
  - Satisfying assignment = string accepted by the grammar/automata

REGULAR constraint

- REGULAR(A,[X1,..Xn]) holds iff
  - X1 .. Xn is a string accepted by the deterministic finite automaton A
  - Proposed by Pesant at CP 2004
  - GAC algorithm using dynamic programming
  - However, DP is not needed since simple ternary encoding is just as efficient and effective

REGULAR constraint

- Deterministic finite automaton (DFA)
  - <Q,Sigma,T,q0,F>
  - Q is finite set of states
  - Sigma is alphabet (from which strings formed)
  - T is transition function: Q x Sigma -> Q
  - q0 is starting state
  - F subseteq Q are accepting states

- DFAs accept precisely regular languages
  - Regular language can be specified by rules of the form:

REGULAR constraint

- DFAs accept precisely regular languages
  - Regular language can be specified by rules of the form:

```
NonTerminal -> Terminal
NonTerminal -> Terminal NonTerminal |
                  NonTerminal Terminal
```

- Alternatively given by regular expressions
- More limited than BNF which can express context-free grammars

REGULAR constraint

- Regular language
  - S -> 0 | 0A | AB | AC | 1B | 1
  - A -> 0 | 0A
  - B -> 1 | 1B
  - C -> 1 | 1C | 0 | 0A
- DFA
  - Q={q0,q1,q2}
  - Sigma={0.1}
  - T(q0,0)=q0. T(q0,1)=q1
  - T(q1,0)=q2, T(q1,1)=q1
  - T(q2,0)=q2
  - F={q0,q1,q2}

REGULAR constraint

- Regular language
  - S -> 0 | 0A| AB | AC | 1B | 1
  - A -> 0 | 0A
  - B -> 1 | 1B
  - C -> 1 | 1C | 0 | 0A
- DFA
  - Q={q0,q1,q2}
  - Sigma={0.1}
  - T(q0,0)=q0. T(q0,1)=q1
  - T(q1,0)=q2, T(q1,1)=q1
  - T(q2,0)=q2
  - F={q0,q1,q2}

REGULAR constraint

- Many global constraints are instances of REGULAR
  - AMONG, CONTIGUITY, LEX, PRECEDENCE, STRETCH, ..
- Domain consistency can be enforced in O(ndQ) time using dynamic programming
  - Contiguity example: {0,1}, {0}, {1}, {0,1}, {1}

REGULAR constraint

- REGULAR constraint can be encoded into ternary constraints
- Introduce Qi+1
  - state of the DFA after the ith transition
- Then post sequence of constraints
  - C(Xi,Qi,Qi+1) iff

    DFA goes from state Qi to Qi+1 on symbol Xi

REGULAR constraint

- REGULAR constraint can be encoded into ternary constraints
- Constraint graph is Berge-acyclic
  - Constraints only overlap on one variable
  - Enforcing GAC on ternary constraints achieves GAC on REGULAR in O(ndQ) time

REGULAR constraint

- PRECEDENCE([X1,..Xn]) iff
  - $\min(\{i \mid Xi=j \text{ or } i=n+1\}) < \min(\{i \mid Xi=k \text{ or } i=n+2\})$ for all $j<k$
- States of DFA represents largest value so far used
  - $T(Si,vj)=Si$ if $j<=i$
  - $T(Si,vj)=Sj$ if $j=i+1$
  - $T(Si,vj)=fail$ if $j>i+1$
  - $T(fail,v)=fail$

REGULAR constraint

- PRECEDENCE([X1,..Xn]) iff
  - min({i | Xi=j or i=n+1}) < min({i | Xi=k or i=n+2}) for all j<k
- States of DFA represents largest value so far used
  - T(Si,vj)=Si if j<=i
  - T(Si,vj)=Sj if j=i+1
  - T(Si,vj)=fail if j>i+1
  - T(fail,v)=fail
  - REGULAR encoding of this is just these transition constraints (can ignore fail)

REGULAR constraint

- STRETCH([X1,..Xn]) holds iff
  - Any stretch of consecutive values is between shortest(v) and longest(v) length
  - Any change (v1,v2) is in some permitted set, P
  - For example, you can only have 3 consecutive night shifts and a night shift must be followed by a day off

REGULAR constraint

- STRETCH([X1,..Xn]) holds iff
  - Any stretch of consecutive values is between shortest(v) and longest(v) length
  - Any change (v1,v2) is in some permitted set, P
- DFA
  - Qi is <last value, length of current stretch>
  - Q0= <dummy,0>
  - T(<a,q>,a)=<a,q+1> if q+1<=longest(a)
  - T(<a,q>,b)=<b,1> if (a,b) in P and q>=shortest(a)
  - All states are accepting

Other generalizations of REGULAR

- REGULAR FIX(A,[X1,..Xn],[B1,..Bm]) iff
  - REGULAR(A,[X1,..Xn]) and Bi=1 iff exists j. Xj=I
  - Certain values must occur within the sequence
  - For example, there must be a maintenance shift
  - Unfortunately NP-hard to enforce GAC on this

Other generalizations of REGULAR

- REGULAR FIX(A,[X1,..Xn],[B1,..Bm])
  - Simple reduction from Hamiltonian path
  - Automaton A accepts any walk on a graph
  - n=m and Bi=1 for all i

Chomsky hierarchy

- Regular languages
- Context-free languages
- Context-sensitive languages
- ..

Chomsky hierarchy

- Regular languages
  - GAC propagator in O(ndQ) time
- Conext-free languages
  - GAC propagator in O(n^3) time and O(n^2) space
  - Asymptotically the same as parsing!
- Conext-sensitive languages
  - Checking if a string is in the language PSPACE-complete
  - Undecidable to know if empty string in grammar and thus to detect domain

Context-free grammars

- Applications
  - Hierarchy configuration
  - Bioinformatics
  - Natual language parsing
  - Rostering
  - …
- CFG(G,[X1,…Xn]) holds iff
  - X1 .. Xn is a string accepted by the context free grammar G

CFG propagator

- Adapt CYK parser
- Works on Chomsky normal form
  - Non-terminal -> Terminal
  - Non-terminal -> Non-terminal Non-terminal
- Using dynamic programming
  - Computes V[i,j], set of possible parsings for the ith to the jth symbols

Conclusions

- Global grammar constraints
  - Specify wide range of global constraints
  - Provide efficient and effective propagators automatically
  - Nice marriage of formal language theory and constraint programming!