Tractable Constraint Languages

Zion Schell

Based on Chapter 11 of

Rina Dechter's Constraint Processing

by David Cohen and Peter Jeavons

Disclaimer

- This chapter is a bit weird
  - It lacks a central thread of ideas
  - It lacks a unifying thesis
  - It doesn't present clear derivations of many of its theorems or techniques
- I'm not going to *teach* this chapter
- I will *present* this chapter
- I hope to acquaint you with this content, not impart true understanding of it

Outline

- Introduction
- Basic Definitions
- Constraint Languages
  - Expressiveness of Constraint Languages
  - Complexity of Constraint Languages
- Hybrid Tractability
- Review

Introduction

Introduction

- **Constraint solvers** allow you to define and solve constraint networks.
- They do this by defining some set of basic constraints to be applied to variables.
- This set of constraint primitives can be called the **constraint language** of the solver.

Introduction

- As a solver's constraint language increases in complexity, its **expressiveness** (the complexity of constraint satisfaction problems that it can describe) increases.
- On the other hand, a more complex constraint language requires more complex algorithms, and the solver's **performance** decreases accordingly.

Introduction

- It is therefore necessary to choose a balance between *performance* and *expressiveness* when designing a constraint language.
- This chapter focuses on the design of constraint languages that choose to be less expressive, but that have tractable performance.

Constraint Languages

- A **constraint language** is a set of relations.
  - *e.g*: {$x=y$, $x \neq y$, $x>y$} or {$x+y=z$, $x>y$, $x=3$}
- The **relational subclass** of a constraint language is the set of all CSP instances that only use relations from the language.

Constraint Languages

Tractability:

- Tractable Constraint Language
  - A polynomial algorithm exists to solve all problems in its relational subclass
- Tractable Relation
  - The constraint language consisting of only the relation is tractable

Constraint Languages

Tractability seems to be heavily determined by *domain size* and *constraint arity*

- 2SAT (tractable)
  - domain size 2 and constraint arity 2
- Graph 3-coloring (intractable)
  - domain size 3 and constraint arity 2
- 3SAT (intractable)
  - domain size 2 and constraint arity 3

However...

Constraint Languages

An Example Constraint Language: CHiP

Constraint Languages

An Example Constraint Language: CHiP

- Constraint Handling in Prolog
- Domain
  - $\mathbb{N}$ (natural numbers)
- Constraint Language
  - Domain constraints
  - Arithmetic constraints
  - Compound arithmetic constraints

Constraint Languages

An Example Constraint Language: CHiP

1.) Domain constraints (unary)

- $x \geq a$; $x \leq a$

2.) Arithmetic constraints (unary or binary)

- $ax \neq b$; $ax = by + c$; $ax \leq by + c$; $ax \geq by + c$

3.) Compound arithmetic constraints (*n*-ary)

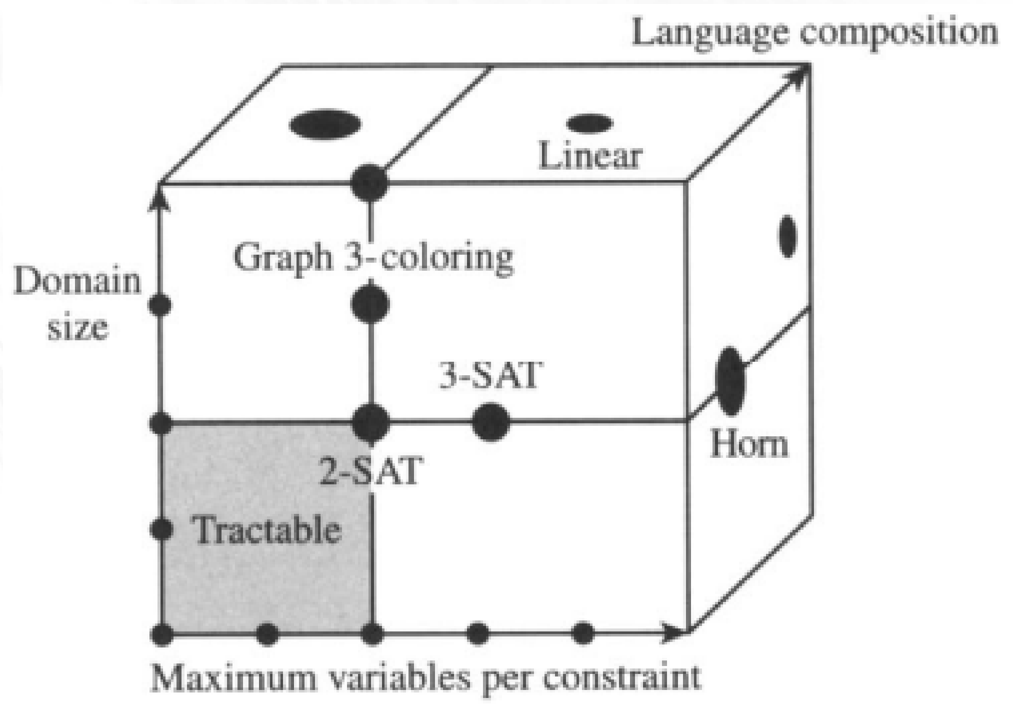
- $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq by + c$
- $ax_1x_2\ldots x_n \geq by + c$
- $(a_1x_1 \geq b_1) \vee (a_2x_2 \geq b_2) \vee \ldots \vee (a_nx_n \geq b_n) \vee (ay \leq b)$

Constraint Languages

- CHiP is actually tractable (!)
  - Enforcing arc-consistency allows backtrack-free solution generation
- CHiP breaks both previous heuristics:
  - Domain $\mathbb{N}$ is infinite
  - Compound arithmetic constraints can have arbitrary arity
- More to tractability than just those two factors

Constraint Languages

The composition of constraint languages somehow determines their tractability



Constraint Languages: Examples

More tractable languages:

- The Constant Language
- Max-closed Languages
- Horn-SAT

Constraint Languages: Examples

The Constant Language:

Constraint Languages: Examples

The Constant Language:

- Domain:
  - {0}
- Constraint language:
  - Relations of the form {(x=0),(x=y=0),(x=y=z=0),…}
  - As well as the relation {(x≠0)}
- Solving is trivial:
  - Set all variables to 0
  - Test constraints
    - If any fail, there is no solution

Constraint Languages: Examples

Max-closed Languages:

Constraint Languages: Examples

Max-closed Languages:

- Domain:
  - A linearly-ordered set
    - Given *x* and *y* in the set, either *x*>*y* or *y*>*x*
- Constraint language:
  - Any max-closed relations on the domain

Constraint Languages: Examples

Max-closed Languages:

- Max-closed relations are based on the function max(a,b)
  - Expanded to tuples elementwise:

    $\max((a_1,a_2),(b_1,b_2)) = (\max(a_1,b_1),\max(a_2,b_2))$

    e.g: max((3,7,2),(2,9,1)) = (3,9,2) = (max(3,2),max(7,9),max(2,1))
  - With the function's domain *closed*:

    The function can always operate on its own output

    e.g: (1,2) and (3,4) in the domain implies (2,4) = max((1,2),(3,4)) in the domain

Constraint Languages: Examples

Horn-SAT:

Constraint Languages: Examples

Horn-SAT:

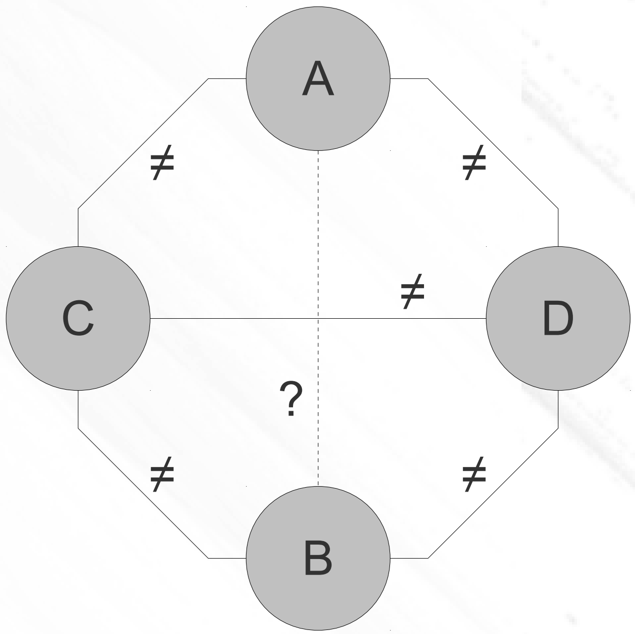
- Domain:
  - {0,1} (Boolean)
- Constraint language:
  - Disjunctive constraints over variables; exactly one element per clause unnegated, the rest negated
    - *e.g*: $(x \vee \bar{y} \vee \bar{z} \vee \bar{w})$
- Solvable in *P* by unit resolution

Constraint Languages: Expressiveness

- In order to define the expressiveness of a constraint language, we need to begin from the bottom and build our way up.
- Just because something is not strictly in the language doesn't mean it can't be expressed with the given constraints.
- We therefore will define **gadgets**, to be used in the construction of any expressible relation.

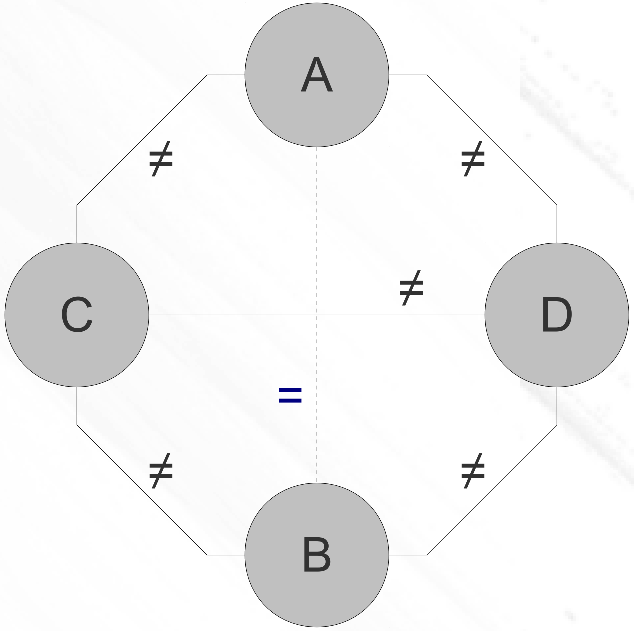
Constraint Languages: Expressiveness

- Gadget Example
  - Consider the problem (with domain {*r*,*g*,*b*})
  - What is the relation between A and B?



Constraint Languages: Expressiveness

- Gadget Example
  - Consider the problem (with domain {*r*,*g*,*b*})
  - What is the relation between A and B?
- The relation is A=B
  - The problem is a **gadget** for =
  - (A,B) is its **construction site**



Constraint Languages: Expressiveness

- Gadgets are CSPs that extend a language outside of what it strictly contains
- From the previous gadget, if a language contains the constraint A≠B, it can also express the constraint A=B
- But imagine trying to make new gadgets
  - Try to make A+B=0 out of A≠B
  - or prove that you can't
- Trial and error isn't going to work, so...

Constraint Languages: Expressiveness

- Let us define the **k^{th}-order universal gadget** of a constraint language *Q*
  - We'll call it **$U_k(Q)$**
- A gadget is only a CSP, so we can define it the same way:
  - Domain
  - Variables
  - Constraints

Constraint Languages: Expressiveness

The *k*th-order universal gadget $U_k(Q)$

- Domain of $U_k(Q)$
  - The same domain as all problems in the relational subclass of Q
  - *e.g*:
    - $Q = \{(A=0),(A=1),(A=2)\}$
      - $\text{Domain}(U_k(Q)) = \{0,1,2\}$
    - $Q = \{(A \vee B)\}$
      - $\text{Domain}(U_k(Q)) = \{0,1\}$

Constraint Languages: Expressiveness

The *k*th-order universal gadget $U_k(Q)$

- Variables of $U_k(Q)$
  - One variable for each *k*-tuple composed of elements in the domain of $U_k(Q)$
  - *e.g*:
    - k=2 and Domain($U_k(Q)$) = {0,1,2}
      - Variables of $U_k(Q)$ = {v00,v01,v02,v10,v11,v12,v20,v21,v22}

Constraint Languages: Expressiveness

The *k*th-order universal gadget $U_k(Q)$

- Variables of $U_k(Q)$
  - **name** of a variable
    - the tuple to which it corresponds
    - *e.g*: name of v01 is (0,1)

Constraint Languages: Expressiveness

The *k*th-order universal gadget $U_k(Q)$

- Variables of $U_k(Q)$
  - **name relation** of a list of variables
    - defined elementwise by variable **names**
    - *e.g*: (v02,v01,v10,v22) → {(0,0,1,2),(2,1,0,2)}

$$(v02,v01,v10,v22) \longrightarrow \begin{array}{c} v02 \\ v01 \\ v10 \\ v22 \end{array} \xrightarrow{\textit{transpose}} \begin{array}{cccc} v & v & v & v \\ \hline 0 & 0 & 1 & 2 \\ \hline 2 & 1 & 0 & 2 \end{array} \longrightarrow \{(0,0,1,2),(2,1,0,2)\}$$

Constraint Languages: Expressiveness

The *k*th-order universal gadget $U_k(Q)$

- Relations of $U_k(Q)$
  - For each relation R in Q
    - Apply R to a tuple of variables in $U_k(Q)$ if and only if the **name relation** of the tuple is a subset of R
  - In other words, find all tuples of variables so if you write them vertically, the rows spell out some of the tuples of R

Constraint Languages: Expressiveness

- This is a horribly strange definition.
- We will therefore derive and show $U_1(Q)$, $U_2(Q)$ and $U_3(Q)$ for $Q = \{A \oplus B, \neg A\}$
  - $A \oplus B$ being the “xor” relation
    - (A,B) in {(0,1),(1,0)} satisfies the constraint
  - $\neg A$ being the “not” relation
    - (A) in {(0)} satisfies the constraint
  - Domain($U_k(Q)$) is Boolean

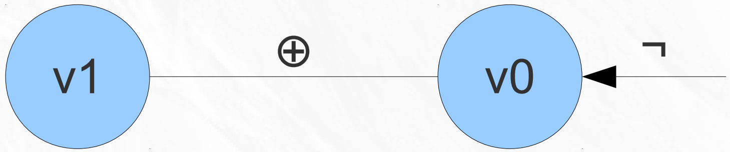
Constraint Languages: Expressiveness

$U_1(Q)$:

- Variables: 1-tuples of domain elements
  - {v0,v1}
- Constraints:
  - ⊕ matches (v0,v1) and (v1,v0)
    - *i.e*: name relation of (v0,v1) is {(0,1)}
  - ¬ matches (v0)
    - *i.e:* name relation of (v0) is {(0)}

Constraint Languages: Expressiveness

$U_1(Q)$:



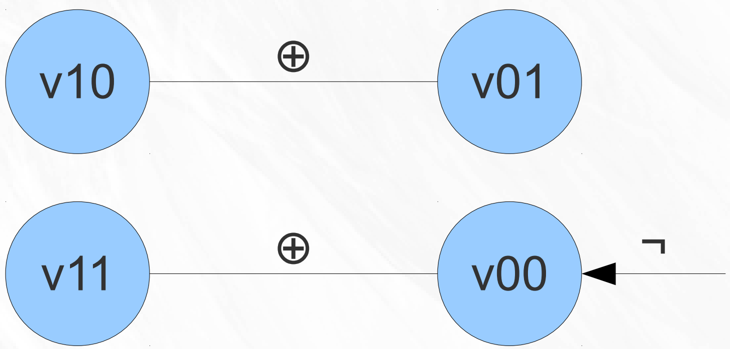
Constraint Languages: Expressiveness

$U_2(Q)$:

- Variables: 2-tuples of domain elements
  - {v00, v01, v10, v11}
- Constraints:
  - ⊕ matches (v00,v11),(v01,v10),(v10,v01), (v11,v00)
    - name relation of (v00,v11) is {(0,1),(0,1)}, etc.
  - ¬ matches (v00)
    - name relation of (v00) is {(0),(0)}

Constraint Languages: Expressiveness

$U_2(Q)$:



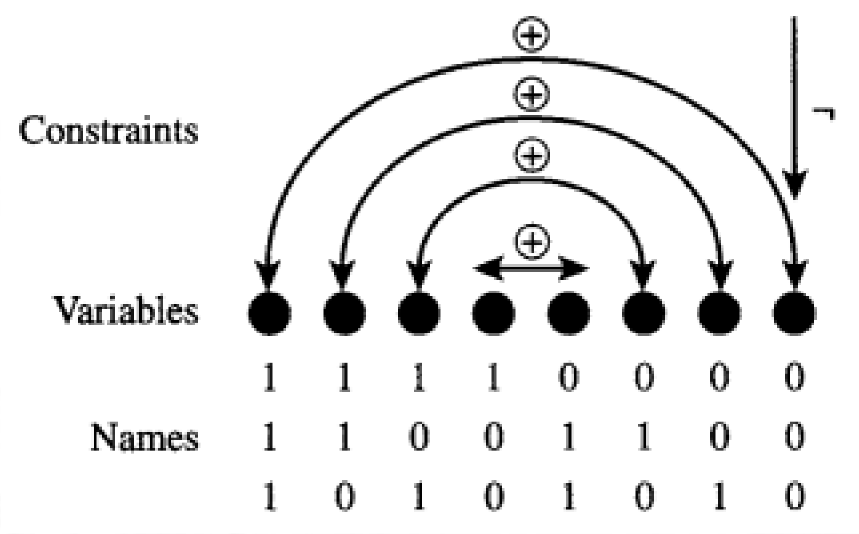
Constraint Languages: Expressiveness

$U_3(Q)$:

- Variables: 3-tuples of domain elements
  - {v000, v001, v010, v011, v100, v101, v110, v111}
- Constraints:
  - ⊕ matches (v000,v111),(v001,v110),(v010,v101),(v011,v100),(v100,v011),(v101,v010),(v110,v001),(v111,v000)
    - name relation of (v001,v110) is {(0,1),(0,1),(1,0)}, etc.
  - ¬ matches (v000)
    - name relation of (v000) is {(0),(0),(0)}

Constraint Languages: Expressiveness

$U_3(Q)$:



Constraint Languages: Expressiveness

And here's where the magic happens:

- Theorem 11.1 (Cohen, Gyssens, Jeavons, 1996)
  - Let Q be a constraint language over a domain D
  - Let R be a relation over D
  - Let k be the number of supports in R
  - Let L_R be **any** list of variables in $U_k(Q)$ whose name relation is R
  - Then,
    - either $U_k(Q)$ expresses R as a gadget with construction site L_R,
    - or R is not expressible in Q

Constraint Languages: Expressiveness

Example application:

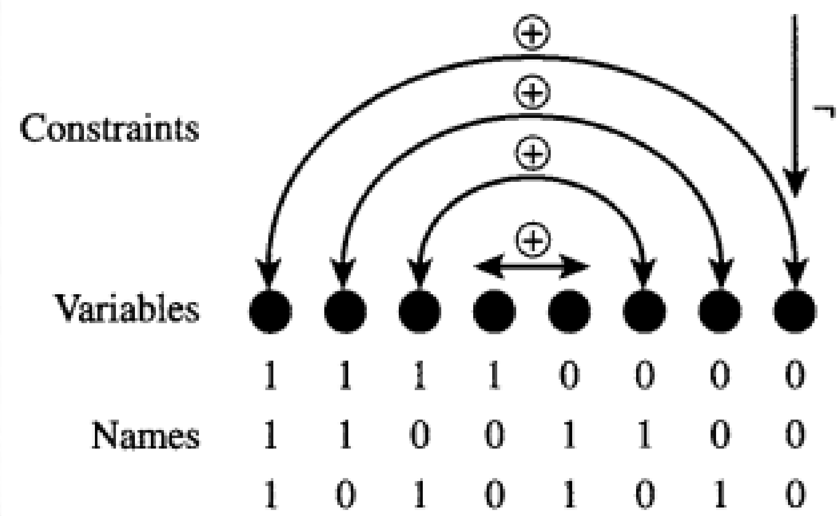
- Is it possible to express $(A \Rightarrow B)$ with the constraint language $Q = \{A \oplus B, \neg A\}$?
- $(A \Rightarrow B)$ formally:

  (A,B) in $\{(0,0),(0,1),(1,1)\}$ satisfies the constraint
- $(A \Rightarrow B)$ has three supports, so we use $U_3(Q)$

Constraint Languages: Expressiveness

$U_3(Q)$ revisited:

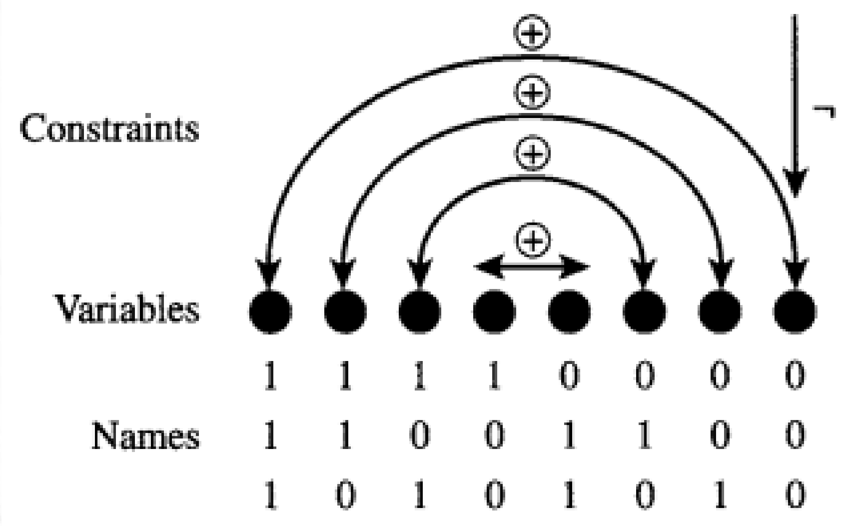
– Looking for a pair of variables with name relation a subset of {(0,0),(0,1),(1,1)}



Constraint Languages: Expressiveness

$U_3(Q)$ revisited:

- Looking for a pair of variables with name relation a subset of {(0,0),(0,1),(1,1)}
- Sample names would be ((0,0,1),(0,1,1)), ((1,0,0),(1,0,1))



Constraint Languages: Expressiveness

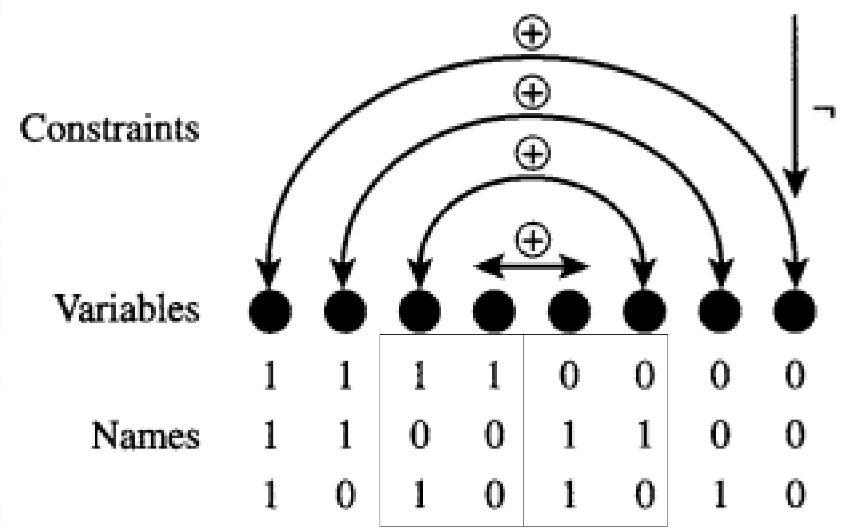
$U_3(Q)$ revisited:

- Looking for a pair of variables with name relation a subset of {(0,0),(0,1),(1,1)}
- Sample names would be ((0,0,1),(0,1,1)), ((1,0,0),(1,0,1))

(v100,v101) or

(v010,v011) work

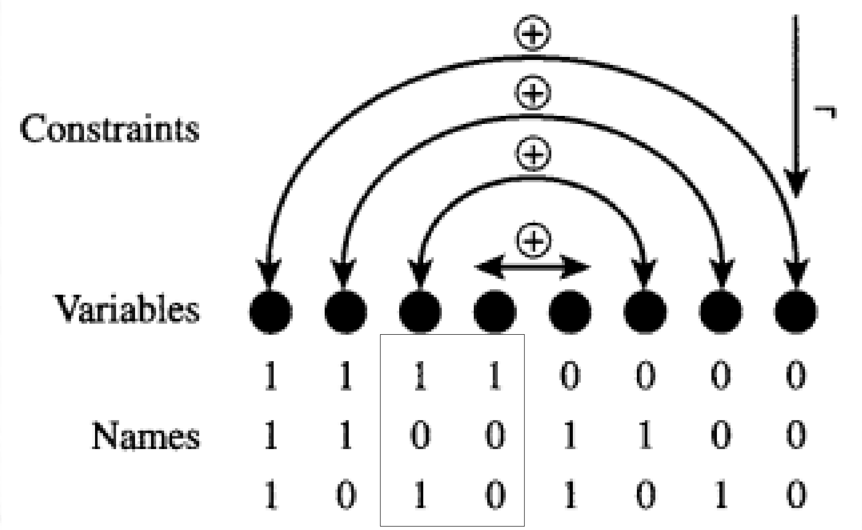
(among others)



Constraint Languages: Expressiveness

$U_3(Q)$ revisited:

- If we treat $U_3(Q)$ as a gadget with construction site (v100,v101), what relation is formed?

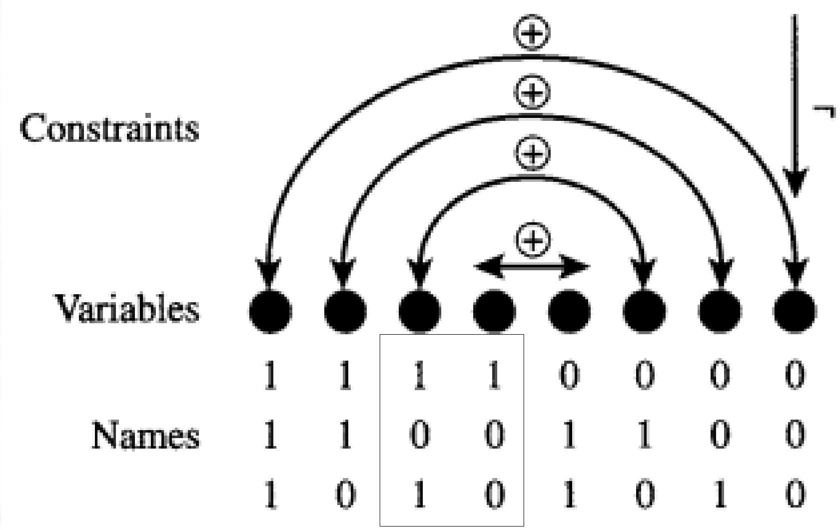


Constraint Languages: Expressiveness

$U_3(Q)$ revisited:

– If we treat $U_3(Q)$ as a gadget with construction site (v100,v101), what relation is formed?

  • {(0,0),(0,1),(1,0),(1,1)} are all viable pairs



Constraint Languages: Expressiveness

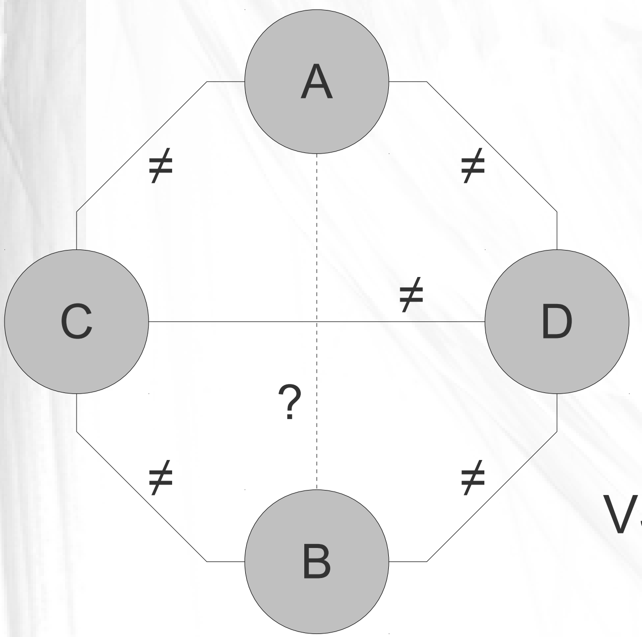
$U_3(Q)$ revisited:

- The gadget does not form the relation $(A \Rightarrow B)$
- This is proof that it is not possible to form $(A \Rightarrow B)$ from $\{A \oplus B, \neg A\}$

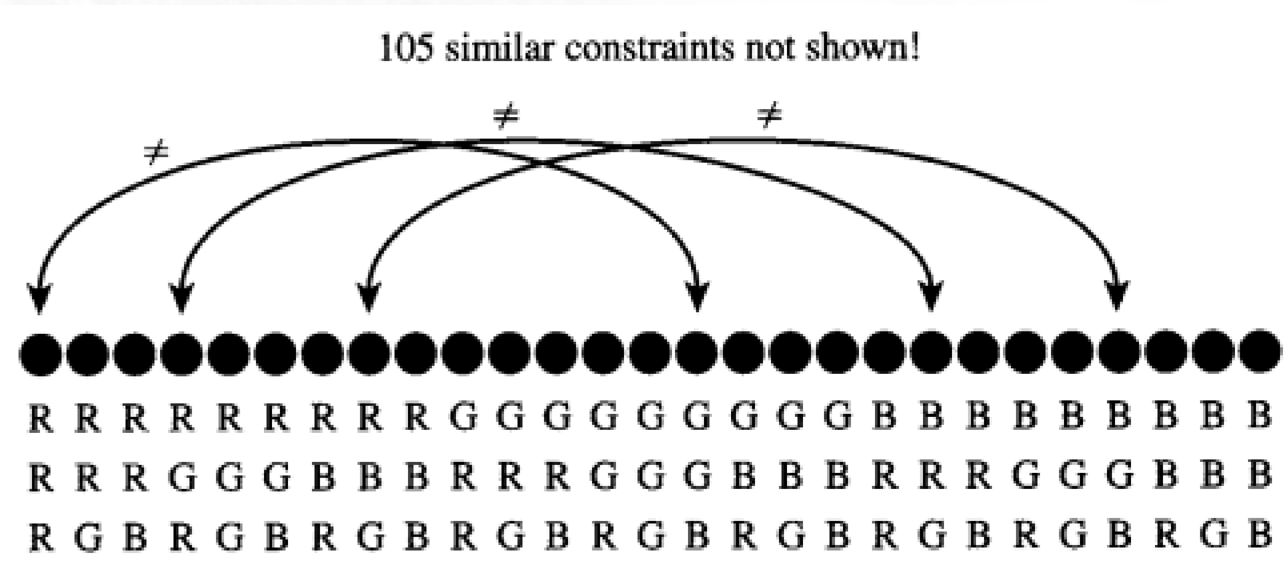
- The proof of Theorem 11.1 is esoteric and whimsical
- Read it if you want to simulate a hangover (without the night out beforehand)

Constraint Languages: Expressiveness

- The universal gadget is not the smallest gadget for all applications; for instance:



VS



Constraint Languages: Expressiveness

- The size of the universal gadget $U_k(Q)$ with domain size d is d^k
- Finding better gadgets in given constraint languages is a open avenue of research
- $U_k(Q)$ also can be used to determine the tractability of Q
- The specific method even suggests algorithms for problems in Q

Constraint Languages: Complexity

- $U_k(Q)$ also can be used to determine the tractability of Q
- The specific method even suggests algorithms for problems in Q
- Let's start with some an example/theorem.

Constraint Languages: Complexity

Theorem 11.2 (Schaefer 1978)

Let *Q* be a boolean constraint language

- *Q* is tractable **if and only if** for each *R* in *Q*:
  - *R* allows (0,0,...,0) or (1,1,...,1)
  - *R* allows disjunctive clauses with at most one negated variable (anti-Horn-clauses)
  - *R* allows disjunctive clauses with at most one nonnegated variable (Horn-clauses)
  - *R* allows disjunctive clauses with at most two variables per clause (2-SAT)
  - *R* is a set of solutions to linear equations on {0,1}

Constraint Languages: Complexity

- By Theorem 11.2, we know *exactly* when a constraint language on a domain of size 2 is tractable
- Not true with other domain sizes
- We do have some conditions for tractability

Constraint Languages: Complexity

- A necessary condition for a tractable constraint language depends on the following definitions:
  - ***k*-ary operation**
    - **idempotent** *k*-ary operation
    - **essentially-unary** *k*-ary operation
      - **projection**
    - **semi-projection**
    - **majority** operation
    - **affine** operation

Constraint Languages: Complexity

- A ***k*-ary operation** from D^k to D maps all k-tuples of elements of D to members of D
  - *e.g:* addition function (k=2, D=$\mathbb{N}$)
    - $+(a,b)$ maps (a,b) to $a+b$
  - *e.g:* maximum function (k=2, D=$\mathbb{R}$)
    - $max(a,b)$ maps (a,b) to a or to b
  - *e.g:* 3-disjunction function (k=3, D={0,1})
    - $3OR(a,b,c)$ maps (a,b,c) to $a \vee b \vee c$

Constraint Languages: Complexity

- **Idempotent** k-ary operation
  - Maps $(x,x,...,x)$ to x for all x
    - *e.g*: $max(a,a) = a$
- **Essentially-unary** k-ary operation
  - Maps $(x_1,x_2,...,x_k)$ to $f(x_i)$ for some $f(x)$ and i
  - A **projection** is a essentially-unary k-ary operation with $f(x) = x$
    - *e.g*: $\pi_b(a,b) = b$

Constraint Languages: Complexity

- **Semi-projection**
  - Maps $(x_1,x_2,\ldots,x_k)$ to x_i in only some cases
    - requires $k \geq 3$
    - examples are very contrived
- **Majority operation**
  - Maps (a,b,c) to its most common element
- **Affine operation**
  - Maps (a,b,c) to $a+b^{-1}+c$
    - where $\langle D,+\rangle$ is an Abelian group

Constraint Languages: Complexity

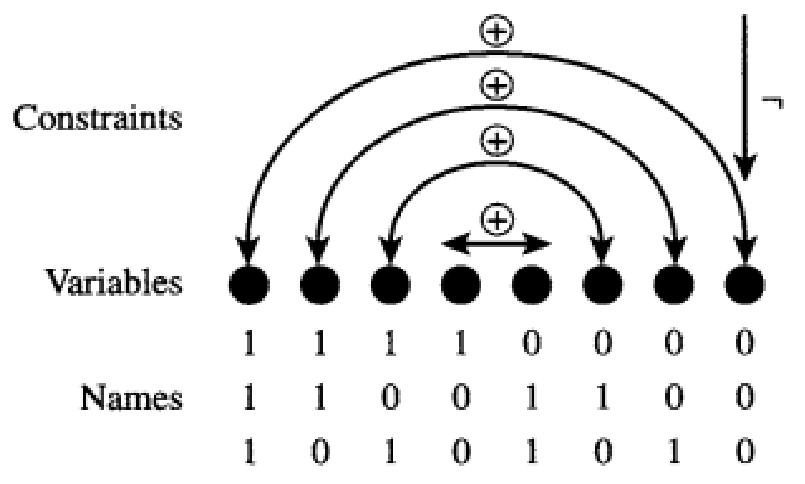
- **Abelian group** is a pair ⟨D,+⟩
  - D is a domain
    - Contains an *identity* element i
      - $a + i = a$
    - Every element a has an *inverse* a^{-1} under +
      - $a + a^{-1} = i$
  - + is an operation on D^2 such that
    - D is *closed* under +
    - + is *associative*
    - + is *commutative*

Constraint Languages: Complexity

- One final definition:
  - Let *s* be a solution to $U_k(Q)$
  - Let the *k*-ary operation $\hat{s}$ **associated with** *s* be defined as follows:
    - The value of $\hat{s}(x)$ is the value assigned to the variable with name *x* in the solution *s.*

Constraint Languages: Complexity

- Example: Q = {A⊕B,¬A}
  - There exists a solution *s* to $U_3(Q)$ as follows:



v111 = 1; v110 = 1; v101 = 1; v100 = 0;

v011 = 1; v010 = 0; v001 = 0; v000 = 0

  - $\hat{s}(1,1,1) = 1$, $\hat{s}(1,0,1) = 1$, $\hat{s}(0,1,0) = 0$, etc.

Constraint Languages: Complexity

- Why would anyone care about the specific types of *k*-ary operations associated with solutions of k^{th}-order universal gadgets to constraint languages?
- Theorem 11.4

  Assuming P≠NP, any tractable constraint language over a finite domain must have a solution to its universal gadget associated with either a *constant* operation, a *majority* operation, an *idempotent* binary operation, an *affine* operation, or a *semi-projection*.

Constraint Languages: Complexity

- Example: $Q = \{A \oplus B, \neg A\}$
  - From:

    v111 = 1; v110 = 1; v101 = 1; v100 = 0;

    v011 = 1; v010 = 0; v001 = 0; v000 = 0
  - We get:

    $\hat{s}(1,1,1) = \hat{s}(1,1,0) = \hat{s}(1,0,1) = \hat{s}(0,1,1) = 1$

    $\hat{s}(1,0,0) = \hat{s}(0,1,0) = \hat{s}(0,0,1) = \hat{s}(0,0,0) = 0$
  - $\hat{s}$ is a majority operation!
  - Q is therefore not intractable

Constraint Languages: Complexity

Corollary to Theorem 11.4

- If all solutions to $U_{|D|}(Q)$ are essentially unary, then Q is NP-complete
- This gets disgusting...
  - $U_{|D|}(Q)$ has $^{2}|D| = |D|^{|D|}$ variables, and combinatorially manyconstraints
    - This is the only time I have ever seen tetration used in an actual formula (and I studied mathematics as an undergrad)
  - Attempting an exhaustive solution is imbecilic

Constraint Languages: Complexity

- Some sufficient conditions for a tractable constraint language depend on the following definitions:
  - A relation *R* **allows** an operation *w* if *w* is associated with a solution to $U_k(\{R\})$
  - ***inv(w)*** is the set of *R* such that *R* allows *w*

Constraint Languages: Complexity

Example: Constant Operations

- Let *w* be any constant operation on *D*
  - *w(x)=C for all x*
- Let *Q* = *inv(w)*
  - *all relations having only one support*
- *Q* is always tractable:
  - Each constraint determines all variables in its scope
  - If two constraints disagree, no solution exists
- The constant language is an example

Constraint Languages: Complexity

Example: Semilattice Operations

- Let + be any operation which is idempotent, commutative, and associative
  - $x+x = x$; $x+y = y+x$; $(x+y)+z = x+(y+z)$
- Let $Q = inv(+)$
- Q is always tractable; the following procedure finds a solution:

Constraint Languages: Complexity

Example: Semilattice Operations

- Establish GAC on the problem
- If any domain is empty, no solution exists
- Otherwise, return the solution where for each variable *v* with domain $d=\{d_1,d_2,\ldots,d_k\}$, the value of *v* is $d_1+d_2+\ldots+d_k$
- An example of this is Horn-SAT
  - The operation “and” is a semilattice operator found as a solution to U_2(Horn-clause)

$$x \wedge x = x; \quad x \wedge y = y \wedge x; \quad (x \wedge y) \wedge z = x \wedge (y \wedge z)$$

Constraint Languages: Complexity

Example: Near-unanimity Operations

- Let w be any k-ary operation which requires near-unanimity
  - all arguments but one must agree, returns the most common
  - *e.g*: the 3-majority operation: $w(x,x,y) = w(x,y,x) = w(y,x,x) = x$ for all x,y
- 2-SAT falls into this category:
  - $w(a,b,c) = (b{=}c)?b{:}a$ can be found in U_3(2-SAT)

Constraint Languages: Complexity

- There are other examples, but the basic idea is this:
  - Even though the specifics of constraint languages vary significantly, the operations associated with solutions to their universal gadgets determine quite effectively whether or not a given language is tractable.

Constraint Languages: Complexity

- Necessary *and* sufficient conditions for tractability are unknown for most cases
- With domain size 2, we have solved it completely, as Theorem 11.2
- In the general case for higher domain size, it isn't known
- If necessary and sufficient conditions could be determined, this would prove or disprove P = NP

Constraint Languages: Hybridization

- Relational Subclasses
  - specific sets of CSPs determined by their constraint language
- Structural Subclasses
  - specific sets of CSPs determined by properties of their hypergraphs
    - *e.g*: requiring a tree structure, requiring clique subgraphs, or any of the other elements discussed in previous classes

But this is not all...

Constraint Languages: Hybridization

- Hybrid Subclasses
  - specific sets of CSPs determined by their constraint languages AND properties of their hypergraphs
- There seems to be no particular heuristics for the tractability of hybrid subclasses
- An example follows:

Constraint Languages: Hybridization

- Given any constraint problem C with domain size d and maximum constraint arity r, then if C is strong $d(r+1)$-consistent, it is globally consistent.
- This subclass is tractable and dependent both on the language (domain size and constraint arity) and structure (consistency requirements).

Constraint Languages: Summary

- Constraint languages are sets of relations.
- Gadgets can be used to extend constraint languages beyond the strict relations present in their definition.
- The expressiveness of constraint languages can be readily defined.
- The tractability of constraint languages can be determined to some extent, but the general case (domain size greater than 2) remains unknown.