

Satisfiability

Generate-and-test / NP / NP-complete
SAT, weighted MAX-SAT, CNF-SAT, DNF-SAT, 3-CNF-SAT, TAUT, QSAT
Some applications: LSAT ©, register allocation

From a previous lecture ...

- So, Prof. Eisner, what are **declarative** methods??
 - A declarative program states only *what* is to be achieved
 - A procedural program describes explicitly *how* to achieve it
- **Sorting in a declarative language**
 - "Given array A, find an array B such that
 - (1) B is a permutation of A
 - (2) B's elements are in increasing order"
 - Compiler is free to invent any sorting algorithm! (Hard?!)
 - You should be aware of when compiler will be efficient/inefficient
- **Sorting in a procedural language**
 - "Given array A, run through it from start to finish, swapping adjacent elements that are out of order ..."
 - Longer and probably buggier
 - Never mentions conditions (1) and (2), except in comments

Generate-and-test problems

(a common form of declarative programming)

- Problem specified by a fast "checking" function $f(A,B)$
- **Input:** string a
- **Output:** Some string b such that $f(a,b) = \text{true}$
 a, b may encode any data you like
- **Examples:**
 - $f(a,b)$ is true iff b is sorted permutation of a
 - $f(a,b)$ is true iff b is timetable that satisfies everyone's preferences, as encoded by a
- **NP** = {all generate-and-test problems with "easy" f }
 - $f(a,b)$ can be computed in polynomial time $O(|a|^k)$, for any b we should consider (e.g., legal timetables)
 - Could do this in for all b in parallel (NP = "nondeterministic polynomial time")

Generate-and-compare problems

(a common form of declarative programming)

- Problem is specified by a fast "scoring" function $f(A,B)$
- **Input:** string a
- **Output:** Some string b such that $f(a,b)$ is **maximized**
 a, b may encode any data you like
- **Examples:**
 - $f(a,b)$ evaluates how **well** the timetable b satisfies everyone's preferences, as encoded by a
- **OptP** = {all generate-&-compare problems with easy f }
 - $f(a,b)$ can be computed in polynomial time $O(|a|^k)$, for each b we need to consider (e.g., legal timetables)

An LSAT Practice Problem

(can we encode this in logic?)

- When the animated "Creature Buddies" go on tour, they are played by puppets.
- **Creatures:** Dragon, Gorilla, Kangaroo, Tiger
- **Names:** Audrey, Hamish, Melville, Rex
- **Chief Puppeteers:** Ben, Jill, Paul, Sue
- **Asst. Puppeteers:** Chris, Emily, Faye, Zeke

An LSAT Practice Problem

(can we encode this in logic?)

Creatures: Dragon, Gorilla, Kangaroo, Tiger
Names: Audrey, Hamish, Melville, Rex
Chief Puppeteers: Ben, Jill, Paul, Sue
Asst. Puppeteers: Chris, Emily, Faye, Zeke

- Melville isn't the puppet operated by Sue and Faye.
- Hamish's chief puppeteer (not Jill) is assisted by Zeke.
- Ben does the dragon, but Jill doesn't do the kangaroo.
- Chris assists with the tiger.
- Rex (operated by Paul) isn't the gorilla (not named Melville).

1. What is the Dragon's name?
2. Who assists with puppet Melville?
3. Which chief puppeteer does Zeke assist?
4. What kind of animal does Emily assist with?

Is there a technique that is guaranteed to solve these?
• Generate & test?
• Is the "grid method" faster? Always works?

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
	H											
	M						x					x
	R											
chief	B											
	J											
	P											
	S						✓					
assistant	C											
	E											
	F											
	Z											

Melville isn't the puppet operated by Sue and Faye.

600.325/425 Declarative Methods - J. Eisner 7

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
	H											
	M							x				x
	R											
chief	B											
	J											
	P											
	S											
assistant	C											
	E											
	F											
	Z											

Melville isn't the puppet operated by Sue and Faye.

600.325/425 Declarative Methods - J. Eisner 8

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
	H											
	M						x					x
	R											
chief	B											
	J											
	P											
	S											
assistant	C											
	E											
	F											
	Z											

This English gave a few facts: e.g., Jill is not assisted by Zeke.

Hamish's chief puppeteer (not Jill) is assisted by Zeke.

600.325/425 Declarative Methods - J. Eisner 9

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
	H											
	M											
	R											
chief	B											
	J											
	P											
	S											
assistant	C											
	E											
	F											
	Z											

Hamish's chief puppeteer (not Jill) is assisted by Zeke.

600.325/425 Declarative Methods - J. Eisner 10

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
	H											
	M											
	R											
chief	B											
	J											
	P											
	S											
assistant	C											
	E											
	F											
	Z											

Ben does the dragon, but Jill doesn't do the kangaroo.

600.325/425 Declarative Methods - J. Eisner 11

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
	H											
	M											
	R											
chief	B											
	J											
	P											
	S											
assistant	C											
	E											
	F											
	Z											

Chris assists with the tiger.

600.325/425 Declarative Methods - J. Eisner 12

		creature				assistant				chief			
		D	G	K	T	C	E	F	Z	B	J	P	S
name	A												
	H					x	x	x	✓		x	x	
	M		x					x	x			x	x
	R		x							x	x	✓	x
chief	B	✓	x	x	x				x				
	J	x		x						x	x	x	x
	P	x							x				
	S	x				x	x	✓	x				
assistant	C	x	x	x	✓								
	E								x				
	F								x				
	Z								x				

Rex (operated by Paul) isn't the gorilla (not named Melville).

600.325/425 Declarative Methods - J. Eisner 15

		creature				assistant				chief			
		D	G	K	T	C	E	F	Z	B	J	P	S
name	A												
	H					x	x	x	✓		x	x	
	M		x					x	x			x	x
	R		x							x	x	✓	x
chief	B	✓	x	x	x				x				
	J	x		x						x	x	x	x
	P	x							x				
	S	x				x	x	✓	x				
assistant	C	x	x	x	✓								
	E								x				
	F								x				
	Z								x				

Paul doesn't operate the gorilla.

Rex (operated by Paul) isn't the gorilla (not named Melville).

600.325/425 Declarative Methods - J. Eisner 14

Grid method

(for puzzles that match people with roles, etc.)

- Each statement simply tells you to fill a particular cell with or
- If you fill a cell with , then fill the rest of its row and column with
- If a row or column has one blank and the rest are , then fill the blank with

600.325/425 Declarative Methods - J. Eisner 15

Is that enough?

No! It didn't even solve the puzzle before.

We need more powerful reasoning patterns ("propagators"):

- We were told that Ben operates the dragon.
- We already deduced that Ben doesn't work with Faye.
- What should we conclude?

600.325/425 Declarative Methods - J. Eisner 16

		creature				assistant				chief			
		D	G	K	T	C	E	F	Z	B	J	P	S
name	A												
	H					x	x	x	✓		x	x	
	M		x					x	x			x	x
	R		x							x	x	✓	x
chief	B	✓	x	x	x				x				
	J	x		x						x	x	x	x
	P	x							x				
	S	x				x	x	✓	x				
assistant	C	x	x	x	✓								
	E								x				
	F								x				
	Z								x				

Combine facts from different 4x4 grids: If XY and YZ then conclude XZ

600.325/425 Declarative Methods - J. Eisner 17

		creature				assistant				chief			
		D	G	K	T	C	E	F	Z	B	J	P	S
name	A												
	H					x	x	x	✓		x	x	
	M		x					x	x			x	x
	R		x							x	x	✓	x
chief	B	✓	x	x	x				x				
	J	x		x						x	x	x	x
	P	x							x				
	S	x				x	x	✓	x				
assistant	C	x	x	x	✓								
	E								x				
	F								x				
	Z								x				

Combine facts from different 4x4 grids: If XY and YZ then conclude XZ

600.325/425 Declarative Methods - J. Eisner 18

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
H					x	x	x	x				
M		x										
R	x	x							x	x	x	
chief	B	✓	x	x	x	x	x					
J	x	x	x									
P	x	x										
S	x											
assistant	C	x	x	x								
E												
F	x											
Z												

Combine facts from different 4x4 grids:
If XY ✓ and YZ ✗
then conclude XZ ✗

600.325/425 Declarative Methods - J. Eisner 19

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
H					x	x	x	x				
M		x										
R	x	x							x	x	x	
chief	B	✓	x	x	x	x	x					
J	x	x	x									
P	x	x										
S	x											
assistant	C	x	x	x								
E												
F	x											
Z												

Combine facts from different 4x4 grids:
If XY ✓ and YZ ✗
then conclude XZ ✗

600.325/425 Declarative Methods - J. Eisner 20

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
H					x	x	x	x				
M		x										
R	x	x							x	x	x	
chief	B	✓	x	x	x	x	x					
J	x	x	x									
P	x	x										
S	x											
assistant	C	x	x	x								
E												
F	x											
Z												

Combine facts from different 4x4 grids:
If XY ✓ and YZ ✗
then conclude XZ ✗

600.325/425 Declarative Methods - J. Eisner 21

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
H					x	x	x	x				
M		x										
R	x	x							x	x	x	
chief	B	✓	x	x	x	x	x					
J	x	x	x									
P	x	x										
S	x											
assistant	C	x	x	x								
E												
F	x											
Z												

No new conclusions from this one

Combine facts from different 4x4 grids:
If XY ✓ and YZ ✗
then conclude XZ ✗

600.325/425 Declarative Methods - J. Eisner 22

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
H					x	x	x	x				
M		x										
R	x	x							x	x	x	
chief	B	✓	x	x	x	x	x					
J	x	x	x									
P	x	x										
S	x											
assistant	C	x	x	x								
E												
F	x											
Z												

Now we can make some easy progress!

600.325/425 Declarative Methods - J. Eisner 23

	creature				assistant				chief			
	D	G	K	T	C	E	F	Z	B	J	P	S
name	A											
H					x	x	x	x	✓	✓	x	x
M		x										
R	x	x							x	x	x	
chief	B	✓	x	x	x	x	x					
J	x	x	x									
P	x	x										
S	x											
assistant	C	x	x	x								
E												
F	x											
Z												

Now we can make some easy progress!

600.325/425 Declarative Methods - J. Eisner 24

		creature				assistant				chief			
		D	G	K	T	C	E	F	Z	B	J	P	S
name	A												
	H					x	x	x	x	✓	✓	x	x
	M		x							x	✓	x	x
	R	x	x							x	x	✓	x
chief	B	✓	x	x	x	x							
	J	x	x	x						x	x	x	x
	P	x	x									x	x
	S	x				x	x	x	✓				
assistant	C	x	x	x	✓								
	E											x	
	F	x										x	
	Z								x				

Now we can make some easy progress!

600.325/425 Declarative Methods - J. Eisner 25



		creature				assistant				chief			
		D	G	K	T	C	E	F	Z	B	J	P	S
name	A												
	H					x	x	x	x	✓	✓	x	x
	M		x							x	✓	x	x
	R	x	x							x	x	✓	x
chief	B	✓	x	x	x	x							
	J	x	x	x						x	x	x	x
	P	x	x									x	x
	S	x				x	x	x	✓				
assistant	C	x	x	x	✓								
	E											x	
	F	x										x	
	Z								x				

The "transitivity rules" let us continue as before. (In fact, they'll let us completely merge name and chief: we don't need to keep track of both anymore.)

600.325/425 Declarative Methods - J. Eisner 26

But does the grid method always apply?

(assuming the statements given are enough to solve the problem)

"Each statement simply tells you to fill a particular cell with  or  – always true?"

People: Alice, Bob, Cindy
Roles: Hero, Villain, Jester

- Alice is not the Jester.
- The Villain and the Hero are legally married in Mississippi.

Let's try it on the board and see what happens ...

600.325/425 Declarative Methods - J. Eisner 27

But does the grid method always apply?

(assuming the statements given are enough to solve the problem)

- Alice is not the Jester.
- The Villain and the Hero are legally married in Mississippi.

The problem is that the second statement didn't tell us exactly which cells to fill in. It gave us choices.

If these English statements aren't just telling us about cells to fill in, what kinds of things are they telling us? Can we encode them formally using some little language? Then the computer can solve them ...

600.325/425 Declarative Methods - J. Eisner 28

Logical formulas (inductive definition)


- If A is a [boolean] variable, then A and ~A are "literal" formulas.
- If F is a formula, then so is ~F ("not F").
- If F and G are formulas, then so are
 - $F \wedge G$ ("F and G" – sometimes written F & G)
 - $F \vee G$ ("F or G" – sometimes written F | G)
 - $F \rightarrow G$ ("If F then G"; "F implies G")
 - $F \leftrightarrow G$ ("F if and only if G"; "F is equivalent to G")
 - $F \text{ xor } G$ ("F or G but not both"; "F differs from G")
- An "assignment" of values to vars: for each var, is it T or F?
 - Given an assignment, can compute truth of formula, bottom-up
- Let's try encoding the hero-villain problem ...

600.325/425 Declarative Methods - J. Eisner 29

Satisfying assignments

I hope you find assignment 1 to be satisfying

- An "assignment" of values to variables is a choice of T or F for each variable
 - Given an assignment, can compute truth of formula, bottom-up
- Satisfiability problem:**
 - Input: a formula
 - Output: a satisfying assignment to its variables, if one exists. Otherwise return "UNSAT".
- Traditional version:**
 - Input: a formula
 - Output: "True" if there exists a satisfying assignment, else "false."
 - Why isn't this useless in practice??**



600.325/425 Declarative Methods - J. Eisner 30

Another LSAT Practice Problem

- When the goalie has been chosen, the Smalltown Bluebirds hockey team has a starting lineup that is selected from two groups:
- 1st Group: John, Dexter, Bart, Erwin
- 2nd Group: Leanne, Roger, George, Marlene, Patricia
- Certain requirements are always observed (for the sake of balance, cooperation, and fairness).

Another LSAT Practice Problem

1st Group: John, Dexter, Bart, Erwin
2nd Group: Leanne, Roger, George, Marlene, Patricia

- Two players are always chosen from the 1st group. Three players are always chosen from the 2nd group. *"3 of 5" here. Could you efficiently encode "13 of 26"?*
 - George will only start if Bart also starts. *"3 of 5" here. Could you efficiently encode "13 of 26"?"*
 - Dexter and Bart will not start together.
 - If George starts, Marlene won't start.
 - The 4 fastest players are: John, Bart, George and Patricia. 3 of the 4 fastest players will always be chosen.
- Who always starts?
 - If Marlene starts, what subset of first-group players starts with her?
 - If George starts, who must also start? (M or J, D or L, D or J, J or P, M or R)
 - Which of these pairs cannot start together? (ED, GJ, RJ, JB, PM)
- These questions are different: \forall not \exists*

Encoding "at least 13 of 26"

(without listing all 38,754,732 subsets!)

A	B	C	...	L	M	...	Y	Z
A \geq 1	A-B \geq 1	A-C \geq 1		A-L \geq 1	A-M \geq 1		A-Y \geq 1	A-Z \geq 1
	A-B \geq 2	A-C \geq 2		A-L \geq 2	A-M \geq 2		A-Y \geq 2	A-Z \geq 2
		A-C \geq 3		A-L \geq 3	A-M \geq 3		A-Y \geq 3	A-Z \geq 3
				A-L \geq 12	A-M \geq 12		A-Y \geq 12	A-Z \geq 12
					A-M \geq 13		A-Y \geq 13	A-Z \geq 13

26 original variables A ... Z,
plus < 26² new variables
such as A-L \geq 3

- SAT formula should require that A-Z \geq 13 is true ... and what else?
- yadayada \wedge A-Z \geq 13 \wedge (A-Z \geq 13 \rightarrow (A-Y \geq 13 \vee (A-Y \geq 12 \wedge Z)))
 \wedge (A-Y \geq 13 \rightarrow (A-X \geq 13 \vee (A-X \geq 12 \wedge Y))) \wedge ...

Encoding "at least 13 of 26"

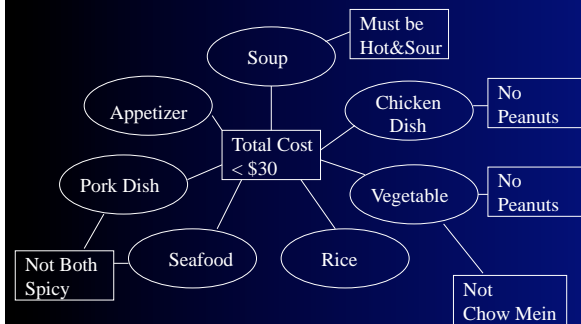
(without listing all 38,754,732 subsets!)

A	B	C	...	L	M	...	Y	Z
A \geq 1	A-B \geq 1	A-C \geq 1		A-L \geq 1	A-M \geq 1		A-Y \geq 1	A-Z \geq 1
	A-B \geq 2	A-C \geq 2		A-L \geq 2	A-M \geq 2		A-Y \geq 2	A-Z \geq 2
		A-C \geq 3		A-L \geq 3	A-M \geq 3		A-Y \geq 3	A-Z \geq 3
				A-L \geq 12	A-M \geq 12		A-Y \geq 12	A-Z \geq 12
					A-M \geq 13		A-Y \geq 13	A-Z \geq 13

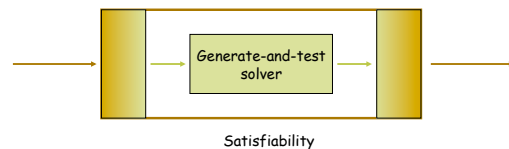
26 original variables A ... Z,
plus < 26² new variables
such as A-L \geq 3

- SAT formula should require that A-Z \geq 13 is true ... and what else?
- yadayada \wedge A-Z \geq 13 \wedge (A-Z \geq 13 \rightarrow (A-Y \geq 13 \vee (A-Y \geq 12 \wedge Z)))
 \wedge (A-Y \geq 13 \rightarrow (A-X \geq 13 \vee (A-X \geq 12 \wedge Y))) \wedge ...
one "only if" definitional constraint for each new variable

Chinese Dinner After the LSAT



Relation to generate-and-test



Relation to generate-and-test

Interreducible problems! (under polynomial encoding/decoding)
 So if we could find a fast solver for SAT, we'd solve half the world's problems (but create new problems, e.g., cryptography wouldn't work anymore) (neither would some theoretical computer scientists)
 So probably impossible. But a "pretty good" SAT solver would help the world.

600.325/425 Declarative Methods - J. Eisner 37

NP-completeness

These innermost problems are called NP-complete
 They are all interreducible with SAT
 A "pretty good" solver for any of them would help the world equally
 Many practically important examples: see course homepage for pointers
 You may well discover some yourself - most things you want are NP-complete

600.325/425 Declarative Methods - J. Eisner 38

Local register allocation

(an NP-complete planning problem from systems)

- Computer has N fast registers, and unlimited slow memory
- Given a straight sequence of computations using $M > N$ vars:
 - ... $x = y * z; w = x + y; \dots$
- Generate assembly-language code:
 - ##### $x = y * z$ #####
 - LOAD R2, (address of y)
 - LOAD R3, (address of z)
 - MUL R1 R2 R3
 - STORE (address of x), R1
 - ##### $w = x + y$ #####
 - LOAD R1, (address of x) *x is already in a register (R1)*
 - LOAD R2, (address of y) *y is already in a register (R2)*
 - ADD R3 R1 R2 *Clever to pick R3? Is z the best thing to overwrite?*
 - STORE (address of w), R3
- Can we eliminate or minimize loads/stores? *Reorder code?*

600.325/425 Declarative Methods - J. Eisner 39

Register allocation as a SAT problem

- Tempting to state everything a reasonable person would know:
 - Don't have the same variable in two registers
 - Don't store something unless you'll need it again ("liveness")
 - Don't store something that's already in memory
 - Only ever store the result of the computation you just did
 - ...
- Yes, looks like an optimal solution will observe these constraints
- But you don't have to state them!
- They are part of the solution, not part of the problem
 - (violating them is allowed, merely dumb)
- Stating these extra constraints *might speed up* the SAT solver
 - Helps the solver prune away stupid possibilities
 - But that's just an optimization you can add later
 - Might* slow things down if the overhead exceeds the benefit
 - A great SAT solver might discover these constraints for itself

600.325/425 Declarative Methods - J. Eisner 40

Assumptions about the input code

- Assume all statements look like $x = y * z$
 - Preprocess input to eliminate more complex statements:

$$a = b * c + d * e \rightarrow \begin{cases} \text{temp1} = b * c \\ \text{temp2} = d * e \\ a = \text{temp1} + \text{temp2} \end{cases}$$
 - Now all intermediate quantities are associated with variables
 - So if needed, we can store them to memory and reload them later
- Assume variables are constant once computed
 - Preprocess: If z takes on two different values, split it into z_1 and z_2

$$\begin{aligned} x &= y * z && \rightarrow \begin{cases} x = y * z_1 \\ z_2 = x + 1 \\ a = a + z \end{cases} \\ z &= x + 1 \\ a &= a + z \end{aligned}$$
 - Now if we have z in a register or in memory, we know which z it is

600.325/425 Declarative Methods - J. Eisner 41

How to set up SAT variables?

(one possibility)

Typical when encoding a planning problem

Other planning problems:

- Get dressed
- Solve Rubik's cube

- K time steps, M program vars, N registers**
 - SAT variable $2y3$ is true iff at time 2, y is in R3
 - SAT variable $2y$ is true iff at time 2, y is in memory
 - An assignment to all variables defines a "configuration" of the machine at each time t
 - How many variables total?
- How will we decode the SAT solution? (which only tells us what the configs are)
 - For each time t, decoder will generate a load-compute-store sequence of machine code to get to t's configuration from t-1's
 - To put some new values in registers, generate loads *preferred (faster than loads)*
 - To put other new values in registers, generate computes *(might use newly loaded registers; order multiple computes topologically)*
 - To put new values in memory, generate stores *(might use newly computed registers)*

600.325/425 Declarative Methods - J. Eisner 42

How to set up SAT variables? Allows dumb solution from a few slides ago (not most efficient, but proves the formula with K steps is SAT)

(one possibility)

- K time steps**
 - How do we pick K?
 - Large enough to compile input
 - Not too large, or solver is slow
 - Solution: Let K = # of input instructions; we know that's large enough ...
- How will we decode the SAT solution?**
 - For each time t, will generate a load-compute-store sequence of machine code to get to t's configuration from t-1's
 - To put some new values in registers, generate loads *preferred (faster)*
 - To put other new values in registers, generate computes *(might use newly loaded registers; order multiple computes topologically)*
 - To put new values in memory, generate stores *(might use newly computed registers)*

$x = y * z$ — Time 0 config — load-compute-store
 $w = x + y$ — Time 1 config — load-compute-store
 ...
 $t = w + x$ — Time K-1 config — load-compute-store
 — Time K config —

600.325/425 Declarative Methods - J. Eisner 43

Example: How do we get from t-1 to t?

	R1	R2	R3	Mem
Time t-1	y			y,z
Time t	x		z	x,y,z

- LOAD** R3, (address of z)
- MUL** R1 R1 R3 // compute x into R1, overwriting y
- STORE** (address of x), R1

For each time t, generate a load-compute-store sequence to get to t's configuration from t-1's

- To put some new values in registers, generate loads
- To put other new values in registers, generate computes *(might use newly loaded registers; order multiple computes topologically)*
- To put new values in memory, generate stores *(might use newly computed registers)*

— Time 0 —
 $x = y * z$
 — Time 1 —
 $w = x + y$
 — Time 2 —
 ...
 — Time K-1 —
 $t = w + x$
 — Time K —

600.325/425 Declarative Methods - J. Eisner 44

What is the SAT formula? This setup may even reorder or eliminate computations!

- Many constraints combined with "and":**
 - At time K, desired output vars are in mem/registers
 - At time 0, input vars are in mem/certain registers
 - ... but otherwise, no vars are held anywhere!
 - E.g., $\neg 0y \wedge 0y1 \wedge \dots \wedge \neg 0x \wedge \neg 0x1 \wedge \dots$
 - No two vars can be in same register at same time
 - If w is in a register at time t but not t-1, where $w = x * y$, then
 - either w was in memory at time t-1 (load)
 - or (better) x, y were also in registers at time t (compute)
 - If w is in memory at time t but not t-1, then
 - w was also in a register at time t (store)
- What's missing?** If these hold, we can find an adequate load-compute-store sequence for t-1 → t

600.325/425 Declarative Methods - J. Eisner 45

Where did we say to minimize loads/stores???

- Additional constraints:**
 - No loads
 - No stores
- But what if SAT then fails?**
 - Retry with a weaker constraint: "at most 1 load/store" ...
 - Continue by linear search or binary chop
 - If w is in a register at time t but not t-1, then
 - either w was in memory at time t-1
 - or (better) x, y were also in registers at time t
 - If w is in memory at time t but not t-1, then
 - w was also in a register at time t

600.325/425 Declarative Methods - J. Eisner 46

Alternatively, use MAX-SAT

- Formula to satisfy is a *conjunction* of several subformulas (known as "clauses")
- A **MAX-SAT** solver seeks an assignment that satisfies as many clauses as possible
- Such conjunctive formulas are very common:
 - each clause encodes a **fact** about the input, or a **constraint**
- Weighted MAX-SAT:** seek an assignment that satisfies a set of clauses with the **maximum total weight possible**
 - How does this relate to unweighted MAX-SAT?
 - Can you implement either version by wrapping the other?
- How do we encode our register problem?
 - Specifically, how do we implement "hard" vs. "soft" constraints?

600.325/425 Declarative Methods - J. Eisner 47

Simpler formulas, simpler solvers

- If A is a [boolean] variable, then A and $\neg A$ are "literal" formulas.
- If F and G are formulas, then so are
 - $F \wedge G$ ("F and G")
 - $F \vee G$ ("F or G")
 - $F \rightarrow G$ ("If F then G"; "F implies G")
 - $F \leftrightarrow G$ ("F if and only if G"; "F is equivalent to G")
 - $F \text{ xor } G$ ("F or G but not both"; "F differs from G")
 - $\neg F$ ("not F")
- Is all of this fancy notation really necessary?**
 - Or is some of it just syntactic sugar?
 - Can we write a front-end preprocessor that reduces the user's formula to a simpler notation? That would simplify solver's job.

600.325/425 Declarative Methods - J. Eisner 48

Simpler formulas, simpler solvers

- If A is a [boolean] variable, then A and $\neg A$ are "literal" formulas.
- If F and G are formulas, then so are
 - $F \wedge G$
 - $F \vee G$
 - $F \rightarrow G$ ($\neg F \vee G$)
 - $F \leftrightarrow G$ ($(F \rightarrow G) \wedge (G \rightarrow F)$ Alternatively: $(F \wedge G) \vee (\neg F \wedge \neg G)$)
 - $F \text{ xor } G$ ($(F \wedge \neg G) \vee (\neg F \wedge G)$)
 - $\neg F$ If not already a literal, can put it in form $\neg(G \wedge H)$ or $\neg(G \vee H)$ or $\neg\neg G$
Equivalent to: $\neg G \vee \neg H$, $\neg G \wedge \neg H$, G
- So we can strip down to \wedge and \vee on literals
 - Eliminate \leftrightarrow and xor, then \rightarrow , then eliminate \neg recursively
- Can we simplify any further?

Simpler formulas, simpler solvers

- Conjunctive normal form (CNF):
 - Conjunction of disjunctions of literals
 - $(A \vee B \vee \neg C) \wedge (\neg B \vee D \vee E) \wedge (\neg A \vee \neg D) \wedge \neg F \dots$
clauses
 - Can turn any formula into CNF
 - So all we need is a CNF solver (still NP-complete: no free lunch)
- Disjunctive normal form (DNF):
 - Disjunction of conjunctions of literals
 - $(A \wedge B \wedge \neg C) \vee (\neg B \wedge D \wedge E) \vee (\neg A \wedge \neg D) \vee \neg F \dots$
Wait a minute! Something wrong. Can you see a fast algorithm?
 - Can turn any formula into DNF
 - So all we need is a DNF solver (still NP-complete: no free lunch)

What goes wrong with DNF

- Satisfiability on a DNF formula takes only linear time:
 $(A \wedge B \wedge \neg C) \vee (\neg B \wedge D \wedge E) \vee (\neg A \wedge \neg D) \vee \neg F \dots$
- Proof that we can convert any formula to DNF
 - First convert to \vee/\wedge form, then do it recursively:
 - Base case: Literals don't need to be converted
 - To convert $A \vee B$, first convert A and B individually
 - $(A1 \vee A2 \vee \dots \vee An) \vee (B1 \vee B2 \vee \dots \vee Bm)$
 - $= A1 \vee A2 \vee \dots \vee An \vee B1 \vee B2 \vee \dots \vee Bm$
 - To convert $A \wedge B$, first convert A and B individually
 - $(A1 \vee A2 \vee \dots \vee An) \wedge (B1 \vee B2 \vee \dots \vee Bm)$
 - $= (A1 \wedge B1) \vee (A1 \wedge B2) \vee \dots \vee (A2 \wedge B1) \vee (A2 \wedge B2) \vee \dots \vee (An \wedge Bm)$
 - Hmm, this is quadratic blowup. What happens to $A \wedge B \wedge C \dots$?
 - $(A1 \vee A2) \wedge (B1 \vee B2) \wedge (C1 \vee C2) \wedge \dots = \text{what?}$
 - Exponential blowup – really just generate and test of all combinations
 - So it doesn't help that we have a linear DNF solver

Just the same thing goes wrong with CNF

- DNF and CNF seem totally symmetric: just swap \vee, \wedge
 - You can convert to CNF or DNF by essentially same algorithm
 - DNF blows up when you try to convert $A \wedge B \wedge C \dots$
 - CNF blows up when you try to convert $A \vee B \vee C \dots$
- But there is another way to convert to CNF! (Tseitin 1970)
 - Only quadratic blowup overall, not exponential
 - Introduce a new "switch variable" for each \vee
 - $(B1 \wedge B2 \wedge B3) \vee (C1 \wedge C2 \wedge C3)$ this is satisfiable iff ...
 - $= (Z \rightarrow (B1 \wedge B2 \wedge B3)) \wedge (\neg Z \rightarrow (C1 \wedge C2 \wedge C3)) \dots$ this is satisfiable (solver picks Z)
 - $= (\neg Z \vee (B1 \wedge B2 \wedge B3)) \wedge (Z \vee (C1 \wedge C2 \wedge C3))$ distribute \vee over \wedge as before, but gives 3+3 clauses, not $3 \cdot 3$
 - $= (\neg Z \vee B1) \wedge (\neg Z \vee B2) \wedge (\neg Z \vee B3) \wedge (Z \vee C1) \wedge (Z \vee C2) \wedge (Z \vee C3)$

Efficiently encode any formula as CNF

- We rewrote $(B1 \wedge B2 \wedge B3) \vee (C1 \wedge C2 \wedge C3)$
 $= (\neg Z \vee B1) \wedge (\neg Z \vee B2) \wedge (\neg Z \vee B3) \wedge (Z \vee C1) \wedge (Z \vee C2) \wedge (Z \vee C3)$
- Recursively eliminate \vee from $(A1 \wedge A2) \vee ((B1 \wedge B2) \wedge (C1 \wedge C2))$:
switching variable Z as before
 $= (A1 \wedge A2) \vee ((\neg Z \vee B1) \wedge (\neg Z \vee B2) \wedge (Z \vee C1) \wedge (Z \vee C2))$
switching variable Y
 $= (\neg Y \vee A1) \wedge (\neg Y \vee A2) \wedge (Y \vee \neg Z \vee B1) \wedge (Y \vee \neg Z \vee B2) \wedge (Y \vee Z \vee C1) \wedge (Y \vee Z \vee C2)$
- Any formula suffers at worst quadratic blowup. Why? Each of its literals simply becomes a clause with the switching variables that affect it:
- For example, we start with a copy of B1 that falls in the 2nd arg of the $Y \vee$ and the 1st arg of the $Z \vee$. So it turns into a clause $(Y \vee \neg Z \vee B1)$.

Efficiently encode any formula as CNF

- We rewrote $(B1 \wedge B2 \wedge B3) \vee (C1 \wedge C2 \wedge C3)$
 $= (\neg Z \vee B1) \wedge (\neg Z \vee B2) \wedge (\neg Z \vee B3) \wedge (Z \vee C1) \wedge (Z \vee C2) \wedge (Z \vee C3)$
- Recursively eliminate \vee from $(A1 \wedge A2) \vee ((B1 \wedge B2) \wedge (C1 \wedge C2) \wedge D)$:
switching variable Z as before
 $= (A1 \wedge A2) \vee ((\neg Z \vee B1) \wedge (\neg Z \vee B2) \wedge (Z \vee C1) \wedge (Z \vee C2) \wedge D)$
switching variable Y
 $= (\neg Y \vee A1) \wedge (\neg Y \vee A2) \wedge (Y \vee \neg Z \vee B1) \wedge (Y \vee \neg Z \vee B2) \wedge (Y \vee Z \vee C1) \wedge (Y \vee Z \vee C2) \wedge (Y \vee D)$
- Any formula suffers at worst quadratic blowup. Why? Each of its literals simply becomes a clause with the switching variables that affect it:
- For example, we start with a copy of B1 that falls in the 2nd arg of the $Y \vee$ and the 1st arg of the $Z \vee$. So it turns into a clause $(Y \vee \neg Z \vee B1)$.

So why can't we use this trick for DNF?

- We rewrote $(B1 \wedge B2 \wedge B3) \vee (C1 \wedge C2 \wedge C3)$ as a "short" CNF formula:
 $= (\neg Z \vee B1) \wedge (\neg Z \vee B2) \wedge (\neg Z \vee B3) \wedge (Z \vee C1) \wedge (Z \vee C2) \wedge (Z \vee C3)$
- But we can just switch \vee and \wedge – they're symmetric!
- So why not rewrite $(B1 \vee B2 \vee B3) \wedge (C1 \vee C2 \vee C3)$ as this "short" DNF?
 $= (\neg Z \wedge B1) \vee (\neg Z \wedge B2) \vee (\neg Z \wedge B3) \vee (Z \wedge C1) \vee (Z \wedge C2) \vee (Z \wedge C3)$
 - Because we'd get a polytime SAT solver and win the Turing Award.
 - And because the rewrite is clearly wrong. It yields something easy to satisfy.
- The CNF/DNF difference is because we introduced extra variables.
- In CNF, original formula was satisfiable if new formula is satisfiable for **either** $Z=\text{true}$ **or** $Z=\text{false}$. But wait! We're trying to switch **or** and **and**:
- In DNF, original formula is satisfiable if new formula is satisfiable for **both** $Z=\text{true}$ **and** $Z=\text{false}$. Look at the formulas above and see it's so!
 - Alas, that's not what a SAT checker checks. We'd have to call it many times (once per assignment to the switching variables Y, Z, \dots).

600.325/425 Declarative Methods - J. Eisner

55

From CNF-SAT to 3-CNF-SAT

- Most SAT solvers let you enter any CNF formula.
 - If you want to use an arbitrary formula, you have to convert it to CNF yourself.
 - Fortunately, many practical problems are naturally expressed as something close to CNF – a "database" of facts and constraints.
- Solvers could be even more annoyingly restrictive:
 - Any CNF formula can be simplified even further, so that every clause has at most 3 literals.
 - E.g., get rid of clauses like $(A1 \vee A2 \vee A3 \vee A4)$.
 - Of course 3-CNF-SAT is still NP-complete – no free lunch!
 - Note: 2-CNF-SAT can be solved in polynomial time.
 - How do we "simplify" a CNF formula to 3-CNF?
 - Would this trick work to get us to 2-CNF?

600.325/425 Declarative Methods - J. Eisner

56

Satisfiability vs. Tautology

- Satisfiability problem:
 - Given a formula F : it's really a boolean function
 - SAT asks whether $(\exists a) F(a)$ a is an assignment
 - Hard if F is in CNF, easy if F is in DNF
 - A typical problem in NP: generate+test, return "yes" if \exists
- Tautology problem:
 - TAUT asks whether $(\forall a) F(a)$
 - Can we solve this by wrapping SAT?
 - Equivalent to $\neg(\exists a) \neg F(a)$
 - So just negate the formula, call SAT, and negate result
 - And vice versa, we can solve SAT by wrapping TAUT
 - Hard if $\neg F$ is in CNF, easy if $\neg F$ is in DNF
 - Thus, hard if F is in DNF, easy if F is in CNF
 - A typical problem in co-NP: generate+test, return "yes" if \forall

600.325/425 Declarative Methods - J. Eisner

57

Quantified Satisfiability (QSAT)

- SAT asks whether $(\exists a) F(a)$ a is an assignment to all vars
- TAUT asks whether $(\forall a) F(a)$ a is an assignment to all vars
- They're the standard NP-complete and co-NP-complete problems.
- QSAT lets you ask, for example, whether
 $(\exists a) (\forall b) (\exists c) (\forall d) F(a,b,c,d)$ a,b,c,d are assignments to non-overlapping subsets of the vars
- Harder! Worse than NP-complete (outside both NP and co-NP).
- QSAT problems are the standard complete problems for the problem classes higher up in the polynomial hierarchy.
- Example: Can White force a win in 4 moves? That is: Is there an opening move for you $(\exists a)$ such that for whatever response Kasparov makes $(\forall b)$, you will have some reply $(\exists c)$ so that however Kasparov moves next $(\forall d)$, you've checkmated him in a legal game $(F(a,b,c,d))$?

600.325/425 Declarative Methods - J. Eisner

58