15-780: Graduate AI
Lecture 5. Logic, Planning

Geoff Gordon (this lecture)
Ziv Bar-Joseph
TAs Geoff Hollinger, Henry Lin

Review

Review

- *CSPs (definition, examples)*
  - *Sudoku, jobshop scheduling*
- *Over-, under-, critically-constrained*
- *Basic search for SAT & CSPs*

Search in SAT, CSPs

- *Constraint propagation / unit resolution*
- *Constraint learning from conflict clauses*
- *Variable ordering*
    - *activity, most-constrained variable*
- *Value ordering*
    - *least-constraining value*

Citation for MiniSAT

- *http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/cgi/MiniSat.ps.gz.cgi*
- *Also, the map-coloring applet that I linked last class appears to be offline*

Randomization

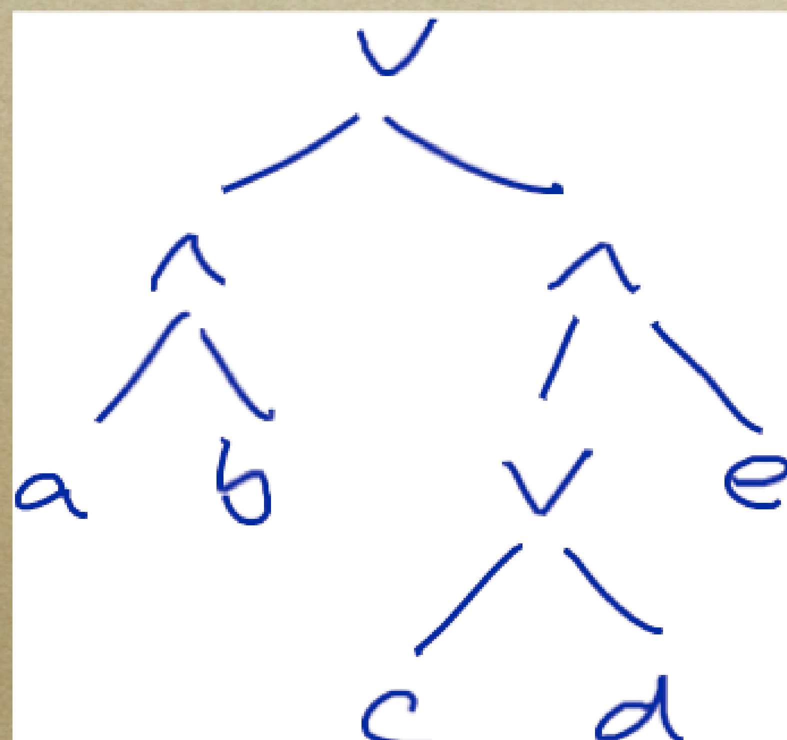
- *Random restarts for DFS-based (DPLL) search*
  - *avoiding doldrums*
- *WalkSAT*

Tseitin transformation

- *Put the following formula in CNF:*

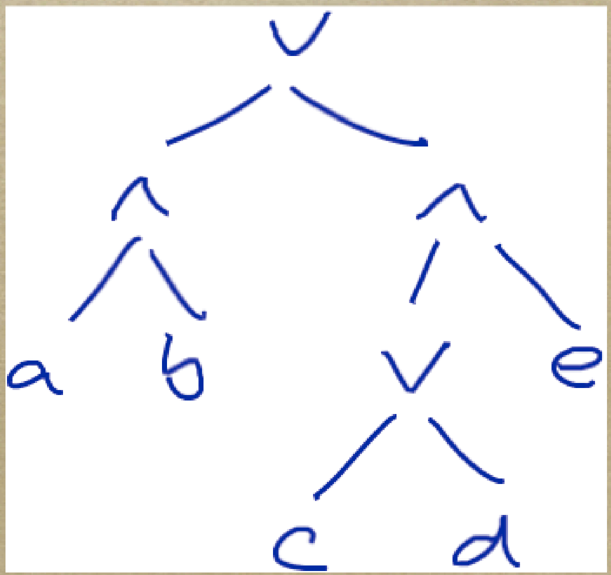
  $(a \wedge b) \vee ((c \vee d) \wedge e)$

- *Parse tree:*



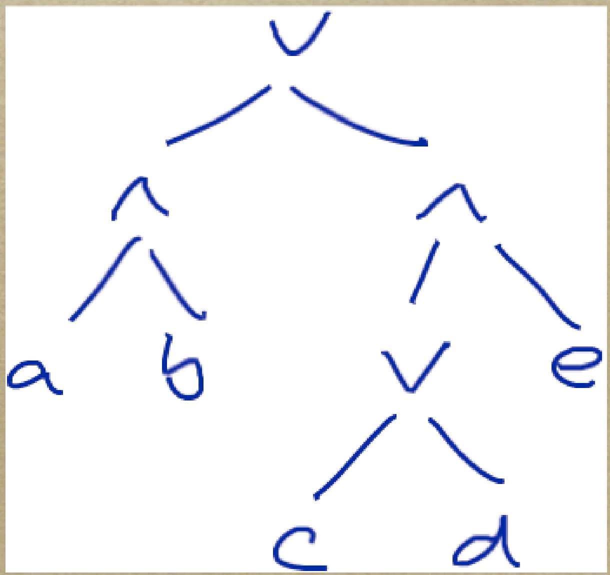
Tseitin transformation

- *Introduce temporary variables*
  - $x = (a \wedge b)$
  - $y = (c \vee d)$
  - $z = (y \wedge e)$



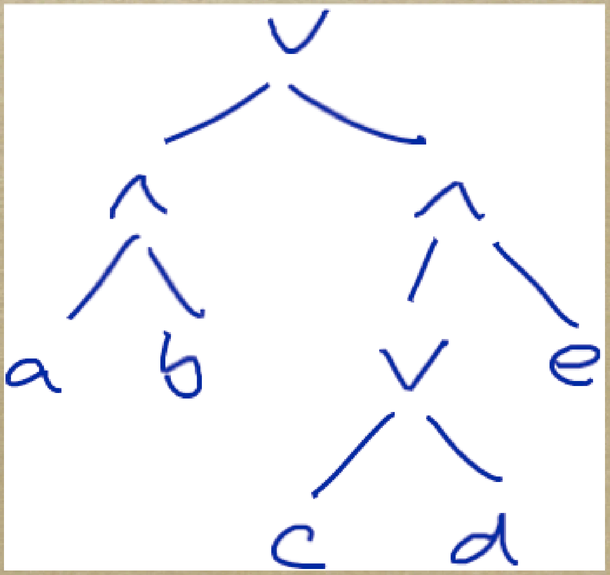
Tseitin transformation

- *To ensure* $x = (a \wedge b)$, *want*
  - $x \Rightarrow (a \wedge b)$
  - $(a \wedge b) \Rightarrow x$



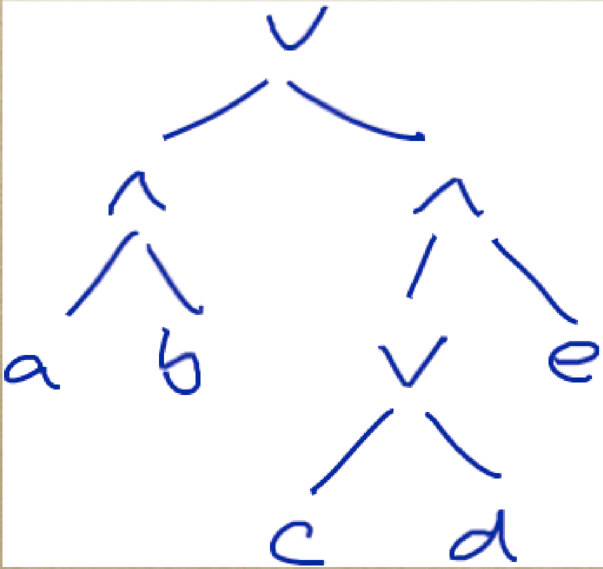
Tseitin transformation

- $x \Rightarrow (a \wedge b)$
- $(\neg x \vee (a \wedge b))$
- $(\neg x \vee a) \wedge (\neg x \vee b)$



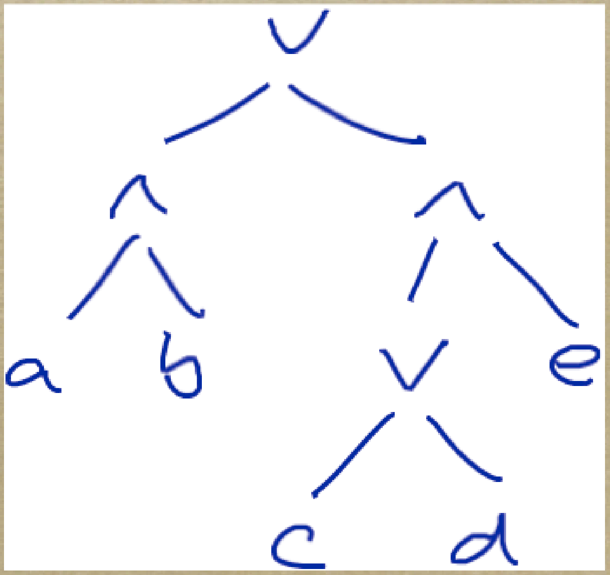
Tseitin transformation

- $(a \wedge b) \Rightarrow x$
- $(\neg(a \wedge b) \vee x)$
- $(\neg a \vee \neg b \vee x)$



Tseitin transformation

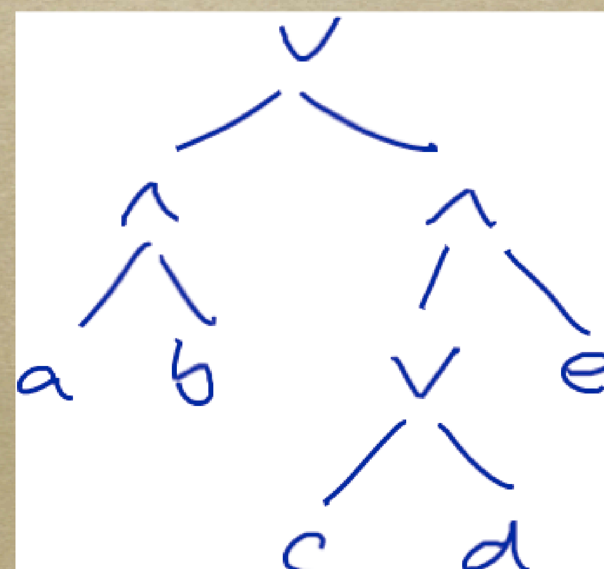
- *To ensure* $y = (c \vee d)$, *want*
  - $y \Rightarrow (c \vee d)$
  - $(c \vee d) \Rightarrow y$



Tseitin transformation

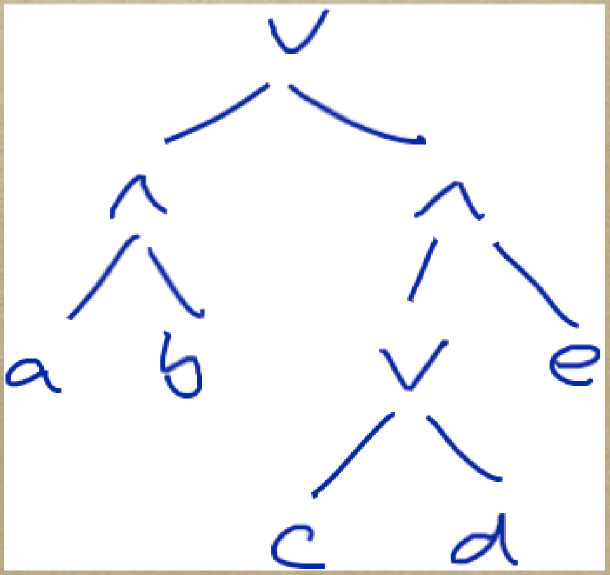
- $y \Rightarrow (c \lor d)$
- $(\neg y \lor c \lor d)$

- $(c \lor d) \Rightarrow y$
- $((\neg c \land \neg d) \lor y)$
- $(\neg c \lor y) \land (\neg d \lor y)$

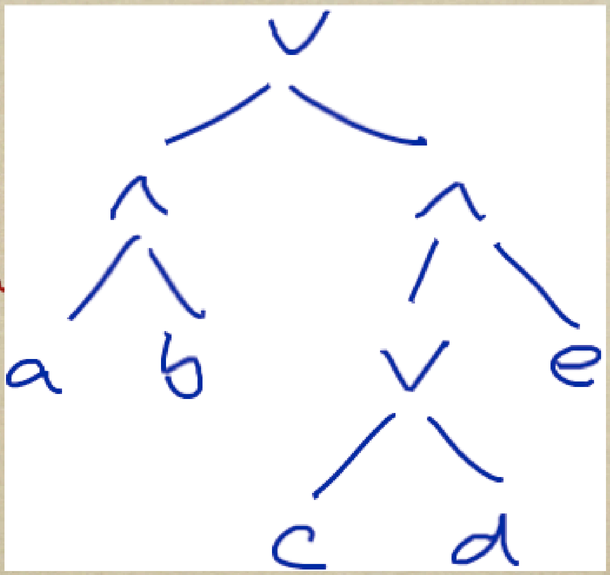


Tseitin transformation

- *Finally*, $z = (y \wedge e)$
- $z \Rightarrow (y \wedge e) \;\equiv\; (\neg z \vee y) \wedge (\neg z \vee e)$
- $(y \wedge e) \Rightarrow z \;\equiv\; (\neg y \vee \neg e \vee z)$



Tseitin end result



$(a \wedge b) \vee ((c \vee d) \wedge e) \ \equiv$

$(\neg x \vee a) \wedge (\neg x \vee b) \wedge (\neg a \vee \neg b \vee x) \wedge$

$(\neg y \vee c \vee d) \wedge (\neg c \vee y) \wedge (\neg d \vee y) \wedge$

$(\neg z \vee y) \wedge (\neg z \vee e) \wedge (\neg y \vee \neg e \vee z) \wedge$

$(x \vee z)$

HW questions

- *3(a) asks you to implement an "opaque" data structure for nodes*
- *This just means that there is a well-defined interface, and data structure is accessed only through interface*
- *E.g., definitions of pq_init, pq_set, pq_pop, pq_test are such an interface, so the priqueue we gave is opaque*

State numbering in maze

y \ x	1	2	3	4	5
1	1	6	11	16	21
2	2	7	12	17	22
3	3	8	13	18	23
4	4	9	14	19	24
5	5	10	15	20	25

- *This contradicts description in the text, but matches the code—updated text on web*

HW questions

- *Storing backpointers in A*, BFS, etc.*

Generic search

$S = \{\ start\ \}\quad M = \emptyset$

$While\ (S \neq \emptyset)$

$x \leftarrow some\ element\ of\ S,\ \ S \leftarrow S \setminus x$

$CheckSolution(x)$

$For\ y \in neighbors(x) \setminus M$

$S \leftarrow S \cup \{y\}\ ,\ backpointer(y) \leftarrow x$

$M = M \cup \{x\}$

HW questions

- *More questions?*

First-order logic

Predicates and objects

- *Interpret happy(John) or likes(Joe, pizza) as a **predicate** applied to some **objects***
- *Object = an object in the world*
- *Predicate = boolean-valued function of objects*
- *Zero-argument predicate plays same role that Boolean variable did before*

Functions

- ***Functions*** *map zero or more objects to another object*
  - *e.g., professor(15-780), last-common-ancestor(John, Mary)*
- *Zero-argument function is the same as an object—John v. John()*

Definitions

- ***Term*** *= expression referring to an object*
  - *John*
  - *left-leg-of(father-of(president-of(USA)))*
- ***Atom*** *= predicate applied to objects*
  - *happy(John)*
  - *raining*
  - *at(robot, Wean-5409, 11AM-Wed)*

Definitions

- ***Literal*** *= possibly-negated atom*
  - *happy(John), ¬happy(John)*
- ***Sentence*** *= literals joined by connectives like* ∧∨¬⇒
  - *raining*
  - *done(slides(780))* ⇒ *happy(professor)*

Models

- *Meaning of sentence: model $\mapsto$ {T, F}*
- *Models are now much more complicated*
  - *List of objects*
  - *Table of function values for each function mentioned in formula*
  - *Table of predicate values for each predicate mentioned in formula*

For example

KB describing example

- $alive(cat)$
- $ear\text{-}of(cat) = ear$
- $in(cat, box) \wedge in(ear, box)$
- $\neg in(box, cat) \wedge \neg in(cat, nil) \ldots$
- $ear\text{-}of(box) = ear\text{-}of(ear) = ear\text{-}of(nil) = nil$
- $cat \neq box \wedge cat \neq ear \wedge cat \neq nil \ldots$

Model of example

- *Objects: C, B, E, N*
- *Assignments:*
  - *cat: C, box: B, ear: E, nil: N*
  - *ear-of(C): E, ear-of(B): N, ear-of(E): N, ear-of(N): N*
- *Predicate values:*
  - *in(C, B), ¬in(C, C), ¬in(C, N), …*

Failed model

- *Objects: C, E, N*
- *Fails because there's no way to satisfy inequality constraints with only 3 objects*

Another possible model

- *Objects: C, B, E, N, X*
- *Extra object X could have arbitrary properties since it's not mentioned in KB*
- *E.g., X could be its own ear*

An embarrassment of models

- *In general, can be infinitely many models*
  - *unless KB limits number somehow*
- *Job of KB is to rule out models that don't match our idea of the world*

Aside: typed variables

- *KB illustrates need for data types*
- *Don't want to have to specify ear-of(box) or ¬in(cat, nil)*
- *Could design a type system*
  - *argument of happy() is of type* animate
- *Function instances which disobey type rules have value nil*

Quantifiers

- *So far, still can't say "all men are mortal"*
- *Add* ***quantifiers*** *and* ***object variables***
  - $\forall x.\ man(x) \Rightarrow mortal(x)$
  - $\neg \exists x.\ lunch(x) \wedge free(x)$
- $\forall$*: no matter how we fill in object variables, formula is still true*
- $\exists$*: there is some way to fill in object variables to make formula true*

Quantification

- *Now we have atoms with **free variables***
- *Adding quantifier for x is called **binding** x*
    - *In (∀x. likes(x, y)), x is bound, y is free*
- *Can add quantifiers and apply logical operations like ∧∨¬ in any order*
- *But must wind up with **ground** formula (no free variables)*

Scoping rules

- *Portion of formula where quantifier applies = **scope***
- *Variable is bound by **innermost** enclosing scope with matching name*
- *Two variables in different scopes can have same name—they are still different vars*

Scoping examples

- $(\forall x.\ happy(x)) \lor (\exists x.\ \neg happy(x))$
  - *Either everyone's happy, or someone's unhappy*
- $\forall x.\ (raining \land outside(x) \Rightarrow (\exists x.\ wet(x)))$
  - *The x who is outside may not be the one who is wet*

Semantics of ∀

- *Write (M / x: obj) for the model which is just like M except that variable x is assigned to the object obj*
- *M / x: obj is a **refinement** of M*
- *A sentence (∀x. S) is true in M if S is true in (M / x: obj) for any object obj in M*

Example

- *M has objects (A, B, C) and predicate happy(x) which is true for A, B, C*
- *Sentence* ∀*x. happy(x) is satisfied in M*
  - *since happy(A) is satisfied in M/x:A, happy(B) in M/x:B, happy(C) in M:x/C*

Semantics of ∃

- *A sentence (∃x. S) is true in M if there is some object obj in M such that S is true in model (M / x: obj)*

Example

- *M has objects (A, B, C) and predicate*
  - *happy(A) = happy(B) = True*
  - *happy(C) = False*
- *Sentence* ∃*x. happy(x) is satisfied in M*
- *Since happy(x) is satisfied in, e.g., M/x:B*

Quantifier nesting

- *English sentence "everybody loves somebody" is ambiguous*
- *Translates to logical sentences*
  - $\forall x.\ \exists y.\ loves(x, y)$
  - $\exists y.\ \forall x.\ loves(x, y)$

Reasoning in FOL

Entailment, etc.

- *As before, entailment, unsatisfiability, validity, etc. refer to all possible models*
- *So, can't in general determine entailment or validity by enumerating models*
  - *since there could be infinitely many*
- *Possible to search for satisfying assignment, but can't show unsatisfiable*

Propositionalization

- *However, people do use SAT-checkers for reasoning in FOL*
- *Turn FOL KB into one or more finite, propositional KBs, search for models in each*
- *More later*

Theorem provers

- *Theorem provers (formula-based search) also generalize to FOL*
- *Both model-based and formula-based searches generally work from KB in CNF*
- *CNF for FOL also called* ***clause form***

Generalizing CNF

- *All transformation rules for propositional logic still hold*
- *In addition, there is a "De Morgan's Law" for moving negations through quantifiers*

$$\neg \forall x.\, S \equiv \exists x.\, \neg S$$

$$\neg \exists x.\, S \equiv \forall x.\, \neg S$$

- *And, rules for getting rid of quantifiers*

Putting FOL KB in CNF

- *Eliminate ⇒, move ¬ in w/ De Morgan*
  - *but ¬ moves through quantifiers too*
- *Get rid of quantifiers (see below)*
- *Distribute ∧∨, or use Tseitin*

Do we really need ∃?

- $(\exists x)\ happy(x)$
- *happy(happy_person())*

- $(\forall y)\ (\exists x)\ loves(y, x)$
- *(∀y) loves(y, loved_one_of(y))*

Skolemization

Thoraf Albert Skolem
1887–1963

- *Called* ***Skolemization*** *(after Thoraf Albert Skolem)*

- *Eliminate ∃ using function of arguments of all enclosing ∀ quantifiers*

Getting rid of quantifiers

- *Standardize apart (avoid name collisions)*
- *Skolemize*
- *Drop* ∀ *(free variables implicitly universally quantified)*
- *Terminology: still called "free" even though quantification is implicit*

For example

- $(\forall x)\ man(x) \Rightarrow mortal(x)$
  - $(\neg man(x) \lor mortal(x))$
- $(\forall x)\ (honest(x) \Rightarrow happy(Diogenes))$
  - $(\neg honest(x) \lor happy(Diogenes))$
- $(\forall y)\ (\exists x)\ loves(y, x)$
  - $loves(y, f(y))$

Exercise

- $((\forall x)\ honest(x)) \Rightarrow happy(Diogenes)$

Exercise

- $((\forall x)\ honest(x)) \Rightarrow happy(Diogenes)$
- $\neg((\forall x)\ honest(x)) \vee happy(Diogenes)$
- $((\exists x)\ \neg honest(x)) \vee happy(Diogenes)$
- $\neg honest(foo()) \vee happy(Diogenes)$
- *foo() = "the guy who might not be honest"*

Theorem provers

Theorem provers

- *Theorem provers work as before:*
  - *add ¬S to KB*
  - *put in CNF*
  - *run resolution*
  - *if we get an empty clause, we've proven S by contradiction*
- *But, CNF and resolution have changed*

Generalizing resolution

- *Propositional:* $(\neg a \vee b) \wedge a \models b$
- *FOL:*

  $(\neg man(x) \vee mortal(x)) \wedge man(Socrates)$

  $\models mortal(Socrates)$

- *Difference: had to substitute* $x = Socrates$

Unification

- *Two FOL sentences **unify** with each other if there is a way to set their variables so that they are identical*
- *man(x), man(Socrates) unify using the substitution x = Socrates*

Unification examples

- *loves(x, x), loves(John, y) unify using*
  x = y = John
- *loves(x, x), loves(John, Mary) can't unify*
- *loves(uncle(x), y), loves(z, aunt(z)):*

Unification examples

- *loves(x, x), loves(John, y) unify using*
  x = y = John
- *loves(x, x), loves(John, Mary) can't unify*
- *loves(uncle(x), y), loves(z, aunt(z)):*
  - *z = uncle(x), y = aunt(uncle(x))*
  - *loves(uncle(x), aunt(uncle(x)))*

Most general unifier

- *May be many substitutions that unify two formulas*
- *MGU is unique (up to renaming)*
- *Finding it takes quadratic time*
  - *because of "occur check"*
  - *does a variable occur inside the formula that it's trying to unify with?*

First-order resolution

- *Given clauses* $(a \vee b \vee c)$, $(\neg c' \vee d \vee e)$
- *And a variable substitution V*
- *If* c / V *and* c' / V *are the same*
- *Then we can conclude*
- $(a \vee b \vee d \vee e) / V$

Proof by SAT

Proof by SAT

- *To prove S, put KB $\wedge$ $\neg$S in clause form*
- *Turn FOL KB into propositional KBs*
  - *in general, infinitely many*
- *Check each one in order*
- *Will turn out that, if any one is unsatisfiable, we have our proof*

Propositionalization

- *Given a FOL KB in clause form*
- *And a set of objects U (for **universe**)*
- *We can **propositionalize** KB under U by substituting elements of U for free variables in all combinations*

Propositionalization example

- $(\neg man(x) \vee mortal(x))$
- $mortal(Socrates)$
- $favorite\_drink(Socrates, hemlock)$
- $drinks(x, favorite\_drink(x))$

- $U = (Socrates, hemlock, Fred)$

Propositionalization example

- $(\neg man(Socrates) \lor mortal(Socrates))$
  $(\neg man(Fred) \lor mortal(Fred))$
  $(\neg man(hemlock) \lor mortal(hemlock))$
- $drinks(Socrates, favorite\_drink(Socrates))$
  $drinks(hemlock, favorite\_drink(hemlock))$
  $drinks(Fred, favorite\_drink(Fred))$
- $mortal(Socrates) \land favorite\_drink$
  $(Socrates, hemlock)$

Choosing a universe

- *To check a FOL KB, propositionalize it using some universe U*
- *Which universe?*

Herbrand Universe

- ***Herbrand universe*** *H of formula S:*
  - *start with all objects mentioned in S*
  - *or synthetic object X if none mentioned*
  - *apply all functions mentioned in S to all combinations of objects in H, add to H*
  - *repeat*

Herbrand Universe

- *E.g., loves(uncle(John), Mary)*
- *H = {John, Mary, uncle(John), uncle (Mary), uncle(uncle(John)), uncle(uncle (Mary)), … }*

Herbrand's theorem

- *If a FOL KB in clause form is unsatisfiable*
- *And H is its Herbrand universe*
- *Then the propositionalized KB is unsatisfiable for some* ***finite*** $U \subseteq H$

Converse of Herbrand

- *A. J. Robinson proved "lifting lemma"*
- *Write PKB for a propositionalization of KB*
- *Any resolution proof in PKB corresponds to a resolution proof in KB*
- *… so, if PKB is unsatisfiable, so is KB*

Proofs w/ Herbrand & Robinson

- *So, FOL KB is unsatisfiable* ***if and only if*** *there is a subset of Herbrand universe making PKB unsatisfiable*

Proofs w/ Herbrand & Robinson

- *To prove S, put KB $\wedge$ $\neg$S in clause form*
- *Build subsets of Herbrand universe in increasing order of size: U_1, U_2, …*
- *Propositionalize KB with U_i, check SAT*
- *If U_i unsatisfiable, we have our contradiction*
- *If U_i satisfiable, move on to U_{i+1}*

Making it faster

- *Restrict semantics so we only need to check one finite propositional KB*
- ***Unique names**: objects with different names are different (John ≠ Mary)*
- ***Domain closure**: objects without names given in KB don't exist*
- *Restrictions also make entailment, validity feasible*

Planning

Time

- *So far, have not modeled a changing world*
- *For KBs that evolve, add extra argument to each predicate saying when it was true*
  - *at(Robot, Wean5409)*
  - *at(Robot, Wean5409, 17)*

Operators

- *Given a representation like this, can define **operators** that change state*
- *E.g., given*
  - *at(Robot, Wean5409, 17)*
  - *moves(Robot, Wean5409, corridor, 17)*
- *could define an operator that implies*
  - *at(Robot, corridor, 18)*
  - *¬at(Robot, Wean5409, 18)*

Goals

- *Want our robot to, e.g., get sandwich*
- *Search for proof of has(Geoff, Sandwich, t)*
- *Analyze proof tree to find sequence of operators that make goal true*

Complications

- *This strategy yields lots of complications*
  - *need axioms describing natural numbers (for time)*
  - *frame axioms (facts don't appear or disappear unless we used an operator)*
  - *unique names, exactly one action per step, …*
- *Result is slow inference*

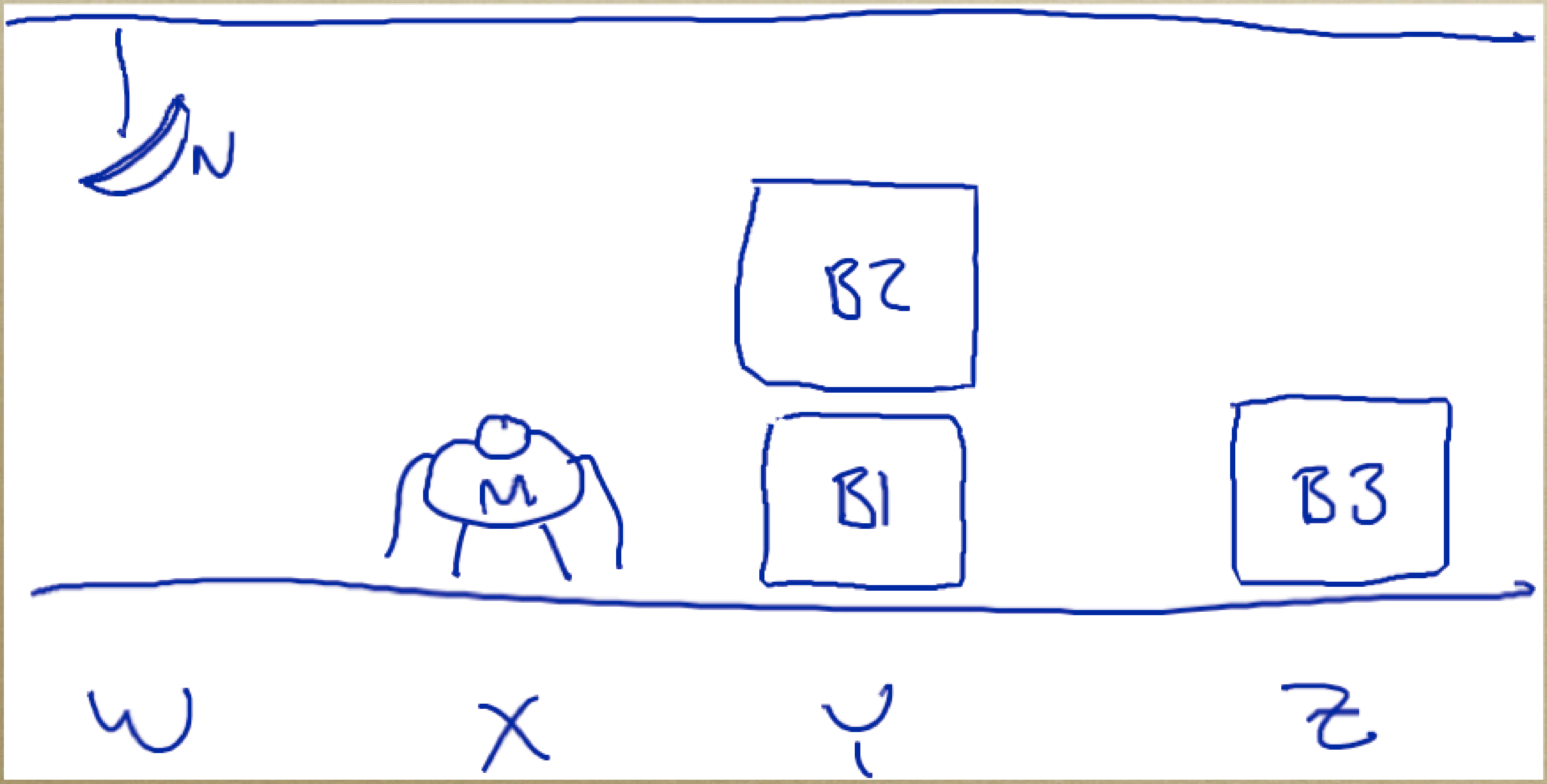
Planning

- *Alternate solution: define a subset of FOL especially for planning*
- *E.g., STRIPS language*
  - *no functions, limited quantification, …*
- *STanford Research Institute Problem Solver*

STRIPS

- *State of world at each time = { propositions }*
- *Each proposition is ground literal*
- *For brevity, list only true literals*
- *Time is implicit*

STRIPS state example



STRIPS state example

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, X)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *height(M, Low)*
- *height(N, High)*

STRIPS operators

- *Operator = { preconditions }, { effects }*
- *If preconditions are true at time t,*
  - *can apply operator at time t*
  - *effects will be true at time t+1*
  - *rest of state unaffected*
- *Basic STRIPS: one operator per step*

Quantification in operators

- *Preconditions of operator may contain variables (implicit ∀)*
- *Operator can apply if preconditions unify with state t (using binding X)*
- *state t+1 has e / X for each e in effects*

Operator example

- *Eat(target, p, l)*
  - *hungry(M), food(target), at(M, p), at(target, p), level(M, l), level(target, l)*
  - $\neg$*hungry(M), full(M),* $\neg$*at(target, p),* $\neg$*level(target, l)*

Operator example

- *Move(from, to)*
  - *at(M, from), level(M, Low)*
  - *at(M, to), ¬at(M, from)*
- *Push(object, from, to)*
  - *at(object, from), at(M, from), clear(object)*
  - *at(M, to), at(object, to), ¬at(object, from), ¬at(M, from)*

Operator example

- *Climb(object, p)*
  - *at(M, p), at(object, p), level(M, Low), clear(object)*
  - *level(M, High), ¬level(M, Low)*
- *ClimbDown()*
  - *level(M, High)*
  - *¬level(M, High), level(M, Low)*

Plan search

Plan search

- *Given a planning problem (start state, operator descriptions, goal)*
- *Run standard search algorithms to find plan*
- *Decisions: search state representation, neighborhood, search algorithm*