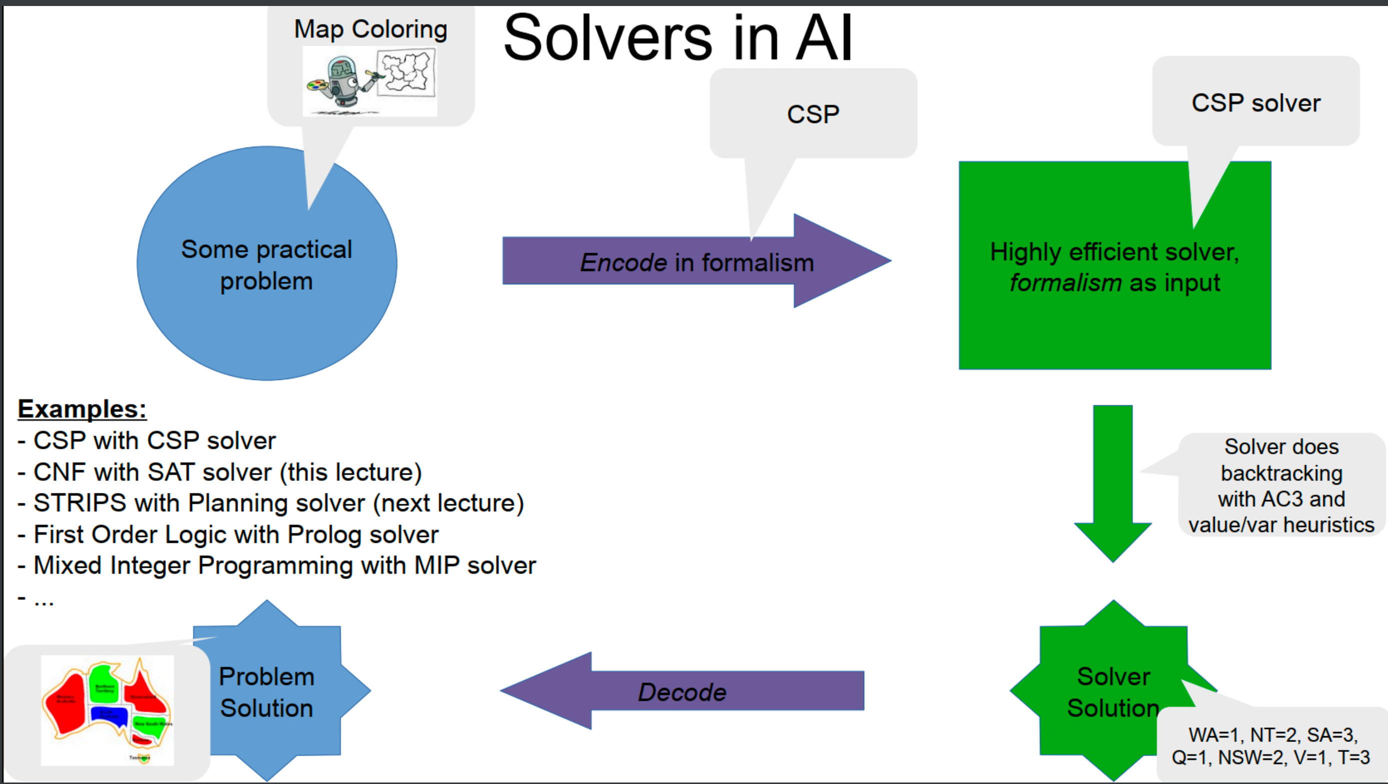
SATs

solvers in AI



CNF: conjuntive normal form

symbol - a

literal - a / (not) a

clause - **disjunction** of literals

sentence - **conjunction** of clauses

DPLL: the core of modern SAT solvers

- early termination
- pure literals 一个变量的所有出现符号都相同
- unit clauses 一个clause只有一个变量

- Partmodel = partial assignment to the symbols

```
function DPLL(clauses,symbols,partmodel={}) – returns true or false
  if every clause in clauses is true in partmodel then
     return true
  if some clause in clauses is false in partmodel then
     return false
  P,value ←FIND-PURE-SYMBOL(symbols,clauses,partmodel)
  if P is not empty then
     return DPLL(clauses, symbols–P, partmodel ∪{P=value})
  P,value ←FIND-UNIT-CLAUSE(clauses,model)
  if P is not empty then
     return DPLL(clauses, symbols–P, partmodel ∪{P=value})
  P ← First(symbols); rest ← Rest(symbols)
  return or(DPLL(clauses,rest,partmodel ∪{P=true}),
              DPLL(clauses,rest,partmodel ∪{P=false}))
```

DFS search

propositional logic

- Given: a set of proposition symbols $\{X_1,X_2,\ldots, X_n\}$
  - (we often add True and False for convenience)
- Symbol X_i is a sentence
- If α is a sentence then $\neg\alpha$ is a sentence
- If α and β are sentences then $\alpha \wedge \beta$ is a sentence
- If α and β are sentences then $\alpha \vee \beta$ is a sentence
- If α and β are sentences then $\alpha \Rightarrow \beta$ is a sentence
- If α and β are sentences then $\alpha \Leftrightarrow \beta$ is a sentence
- And p.s. there are no other sentences!

- Let m be a *model* assigning true or false to $\{X_1, X_2, \ldots, X_n\}$

  (in CSP called: an assignment of values to the symbols/variables)
- If α is a proposition symbol, then its truth value is given in m
- $\neg\alpha$ is true in m iff α is false in m
- $\alpha \wedge \beta$ is true in m iff α is true in m and β is true in m
- $\alpha \vee \beta$ is true in m iff α is true in m or β is true in m
- $\alpha \Rightarrow \beta$ is true in m iff α is *false* in m or β is true in m
- $\alpha \Leftrightarrow \beta$ is true in m iff $\alpha \Rightarrow \beta$ is true in m and $\beta \Rightarrow \alpha$ is true in

every sentence in propositional logic can be translated into an equivalent CNF sencence!

Distributivity:

- (A v B) $\wedge$ C equals (A $\wedge$ C) v (B $\wedge$ C)
  - (I'm in FAI **or** I'm in DPSPAI) **and** I'm awake
  - (I'm in FAI **and** I'm awake) **or** (I'm in DPSPAI **and** I'm awake)
- (A $\wedge$ B) v C equals (A v C) $\wedge$ (B v C)

Also for negation, inverses both operator and arguments:

- $\neg$(A v B) equals ($\neg$A $\wedge$ $\neg$B)
  - **Not** (awake **and** hungry) equals **not** awake, **or not** hungry
- $\neg$(A $\wedge$ B) equals ($\neg$A v $\neg$B)

SAT problem

NP-Complete

SAT is an NP-complete problem (at least as hard as the hardest problems in NP)

P = polynomial time solvable (deterministic on Turing Machine)

NP = non-deterministic polynomial time solvable (can verify a solution in polynomial time)

P belongs to NP. but P=NP? remains unsolved entailment

Entailment: KB $|= \alpha$ ("KB entails α" or "α follows from KB") iff in every world where KB is true, α is also true

I.e., the KB-worlds are a subset of the α-worlds [***models***(KB) $\subseteq$ ***models***(α)]

KB限制更严格，可满足的worlds更少

proofs

- a proof is a demonstration of entailment between α and β
- sound / complete algorithm

  sound 正确性 证明的内容必须正确

  complete 完备性 所有可能正确的内容都可以被证明

method 1 : model checking

for every possible world, if KB is true make sure α is true

- KB $|= \alpha$
- iff KB $\Rightarrow \alpha$ is true in <u>all</u> worlds (a SAT solver searches <u>one</u> world, insufficient)
- iff $\neg$(KB $\Rightarrow \alpha$) is false in all worlds
- iff KB $\wedge \neg\alpha$ is false in all worlds, (i.e., a SAT solver finds <u>no</u> world)

归谬/ 反证法

method 2 : theorem-proving

search for a sequence of proof steps

- We want a way to <u>derive</u> new formulae at the syntactic level
  - KB $|= \alpha$, if we can derive α from KB, we are done
  - Equivalently, if we can derive *failure* from KB $\wedge \neg\alpha$, we are done

<u>resolution rule</u>

$$\frac{C_1 \cup \{l\}, C_2 \cup \{\neg l\}}{C_1 \cup C_2}$$

inference rule, read as "if top-part applies, then bottom part can be derived from it"

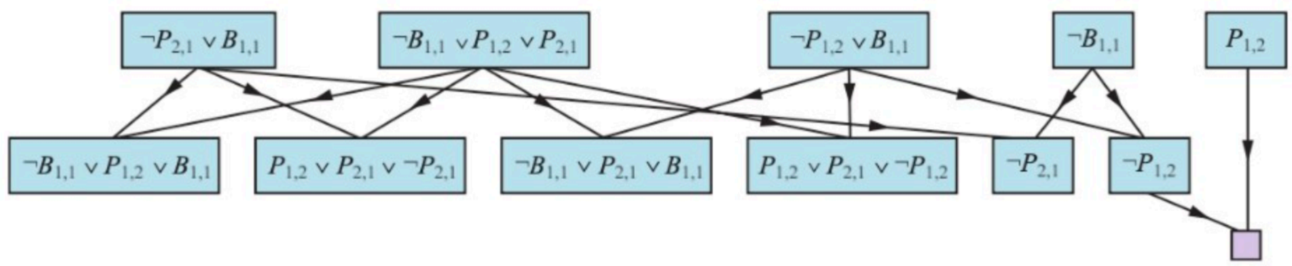
the resulting clause (bottom part) - resolvent

resolution algorithm

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

Figure 7.13 A simple resolution algorithm for propositional logic. PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.



“new 包含于 clauses”判断解析过程是否无法继续生成新的子句

即，子句集合已经达到闭包closure状态，解析过程无法找到空子句，则 $KB \wedge \neg\alpha$ 没有矛盾

sound and complete, but exponential time

relational databases

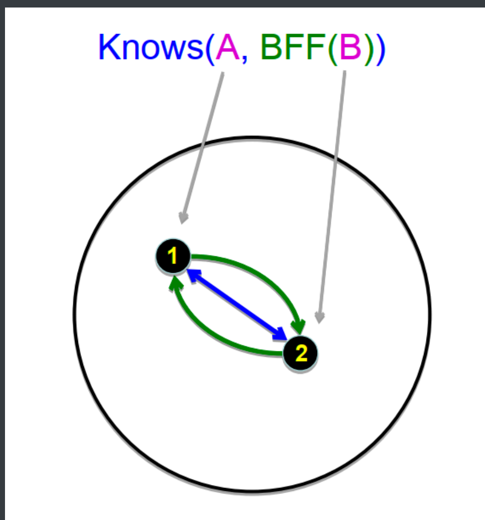
- syntax : fround relational sentences e.g. Sibling(Ali, Bo)
- semantics : sentences in the DB are true, everything else is false

first-order logic (FOL)

- First-order logic (FOL)
  - Syntax: $\forall x\ \exists y\ P(x,y) \land \neg Q(Joe,F(x)) \Rightarrow F(x)=F(y)$
  - Prop. Logic syntax and also $\forall$, $\exists$, $P(x,y)$, $F(x)$, $=$

a possible world for FOL consists of:

constant, function, predicate



a term: a constant / a function with terms as arguments / a variable

an atomic sentence: a predicate with terms as arguments / an equality between terms

complex sentences

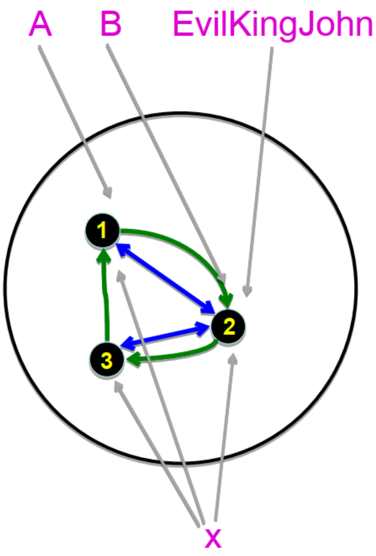
Syntax and semantics: Complex sentences

1) Sentences with logical connectives

$\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \Rightarrow \beta$, $\alpha \Leftrightarrow \beta$

2) Sentences with universal or existential quantifiers, e.g.,

- $\forall$x Knows(x, BFF(x))
  - True in world w iff true in ***all extensions*** of w where x is a variable that refers to an object in w
    - x -> **1**: Knows(**1**, BFF(**1**)) -> Knows(**1**,**2**) -> **T**
    - x -> **2**: Knows(**2**, BFF(**2**)) -> Knows(**2**,**3**) -> **T**
    - x -> **3**: Knows(**3**, BFF(**3**)) -> Knows(**3**,**1**) -> **F**



inference in FOL

- In FOL, we can go beyond just answering "yes" or "no"; given an existentially quantified query, return a ***substitution*** (or ***binding***) for the variable(s) such that the resulting sentence is entailed:
  - KB = $\forall$x Knows(x,Obama)
  - Query (does this follow from KB?) = $\exists$y$\forall$x Knows(x,y)
  - Answer = Yes, σ = {y/Obama}

Notation: $\alpha\sigma$ means applying substitution σ to sentence α

- E.g., if α= $\forall$x Knows(x,y) and σ = {y/Obama}, then $\alpha\sigma$ = $\forall$x Knows(x,Obama)

proof method 1: model-checking

- (KB $\wedge$ $\neg\alpha$) to propositional logic, then to CNF, then SAT solver (grounding)
- for cases like:
  - and $\forall$x Knows(Mother(x),x)
    - Knows(Mother(Obama),Obama) and Knows(Mother(Mother(Obama)),Mother(Obama)) ....

could lead to infinite loop, use depth-bounded model checking

- for *k* = 1 to infinity: use all possible terms of function nesting depth *k*, then check
  - If entailed, will find a contradiction for some finite k; if not, may continue forever: ***semi-decidable***

Bounded model checking is NOT complete, only refutation-complete

- not complete 非完备，有可能找不到所有可能正确的内容
- 但refutation- complete，可以找到矛盾

proof method 2: theorem-proving

- The general rule is a version of Modus Ponens:
  - Given $\alpha \Rightarrow \beta$ and α', where $\alpha'\sigma = \alpha\sigma$ for some substitution σ, conclude $\beta\sigma$
    - σ is {x/Socrates}
  - Given Knows(x,Obama) and Knows(y,z) $\Rightarrow$ Likes(y,z)
    - σ is {y/x, z/Obama}, conclude Likes(x,Obama)