COMP3702/COMP7702
Artificial Intelligence

Module 2: Reasoning and planning with certainty — Part 2

Dr Archie Chapman

Semester 2, 2020

The University of Queensland
School of Information Technology and Electrical Engineering

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati

Week 6: Logistics

- **Assignment 2 has been released, due Sept 25**
- RiPPLE round 2 is open, will close Sept 18
- Tutorials 5 and 6 (next week) will help you with Assignment 2

Week 6 — Overview

Continue with Module 2: Reasoning and planning with certainty

Using logic to represent a problem.

Two types of problems:

- Satisfiability — focus on constraint satisfaction problems (continue from last week)
- Validity (later today)

Table of contents

Consistency algorithms

Consistency Algorithms: Prune the search space

Idea: prune the domains as much as possible before selecting values from them.

A variable is domain consistent (or 1-consistent) if no value of the domain of the node is ruled impossible by any of the constraints.

Example: Is the scheduling example domain consistent?
Variables: $X = \{A, B, C, D, E\}$ that represent the starting times of various activities. Domains: $dom_1 = \{1, 2, 3, 4\}, \forall i \in X$ Constraints: $(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge (C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge (E < C) \wedge (E < D) \wedge (B \neq D)$.

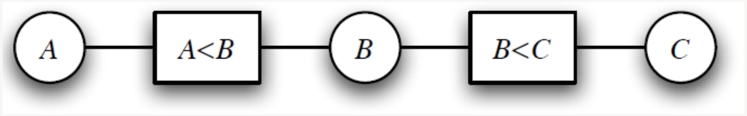
No, $dom_B = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.

Even better, we can propagate this information to other unassigned variables.

Constraint Network: a bipartite graph

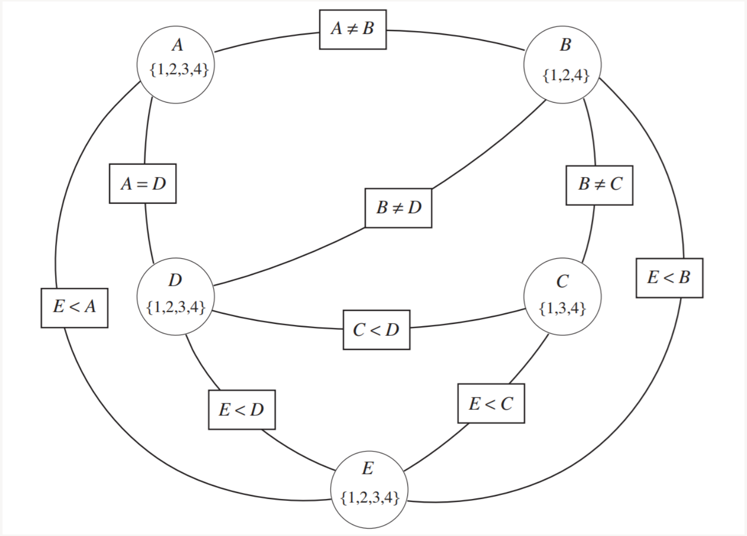
- There is a circle or oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each constraint that involves X.

Example *binary* constraint network:



Variables: $\{A, B, C\}$, Domains: (not specified)
Constraints: $r_1(A, B) = (A < B)$, $r_2(B, C) = (B < C)$
Arcs: $\langle A, r_1(A, B)\rangle$, $\langle B, r_1(A, B)\rangle$,...

Constraint Network: Scheduling example

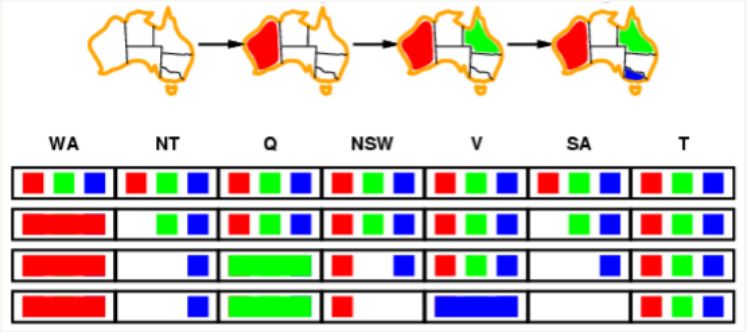


Arc-consistency

Recap: Forward checking

Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

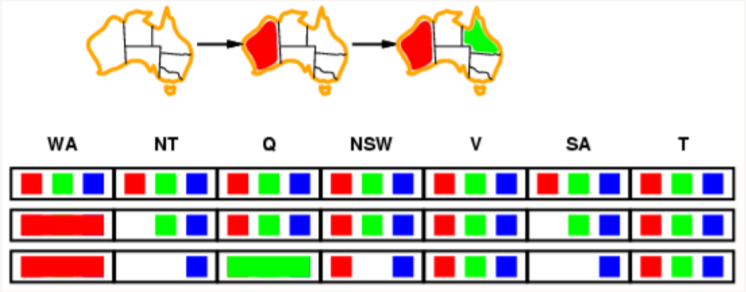


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

This is embedded in "vanilla" backtracking search

Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

- NT and SA cannot both be blue!
- Constraint propagation algorithms repeatedly enforce constraints locally. . .

Arc Consistency

Arc consistency is the simplest form of constraint propagation, which repeatedly enforces constraints locally.

Arc consistency

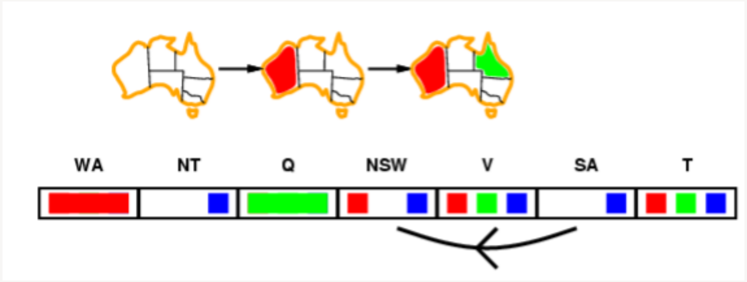
An arc $\langle X, r(X, \overline{Y})\rangle$ is **arc consistent** if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied. A network is arc consistent if all its arcs are arc consistent.

- What if arc $\langle X, r(X, \overline{Y})\rangle$ is *not* arc consistent?
- All values of X in $dom(X)$ for which there is no corresponding value in $dom(\overline{Y})$ can be deleted from $dom(X)$ to make the arc $\langle X, r(X, \overline{Y})\rangle$ consistent.
- This removal can never rule out any models (do you see why?)

Arc Consistency

Arc consistency is the simplest form of constraint propagation, which repeatedly enforces constraints locally.

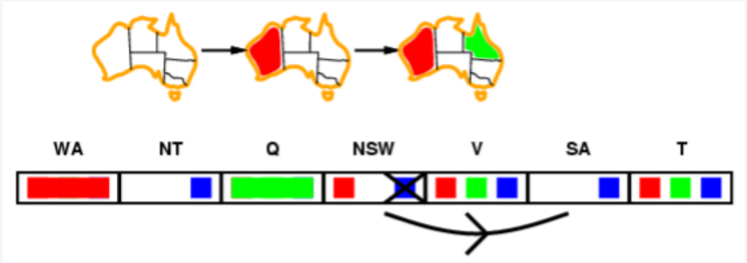
An arc $\langle X, r(X, \overline{Y})\rangle$ is consistent if for every value x of X there is some allowed y.



Arc Consistency

Arc consistency is the simplest form of constraint propagation, which repeatedly enforces constraints locally.

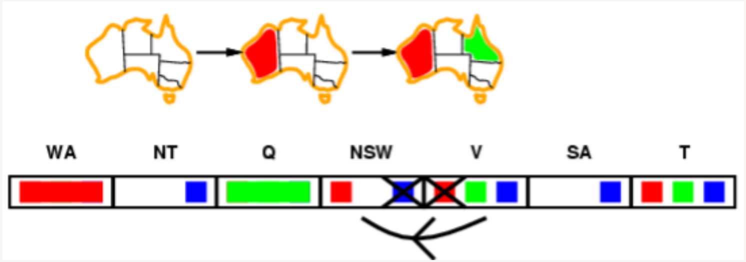
An arc $\langle X, r(X, \overline{Y}) \rangle$ is consistent if for every value x of X there is some allowed y.



Arc Consistency

Arc consistency is the simplest form of constraint propagation, which repeatedly enforces constraints locally.

An arc $\langle X, r(X, \overline{Y}) \rangle$ is consistent if for every value x of X there is some allowed y.

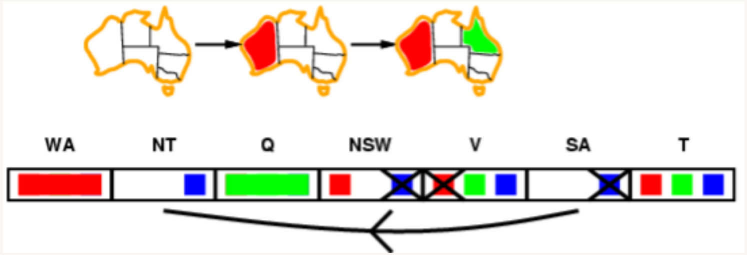


- If X loses a value, neighbors of X need to be rechecked

Arc Consistency

Arc consistency is the simplest form of constraint propagation, which repeatedly enforces constraints locally.

An arc $\langle X, r(X, \overline{Y})\rangle$ is consistent if for every value x of X there is some allowed y.



- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment (e.g. within backtracking search).

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again? An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Regardless of the order in which arcs are considered, we will terminate with the same result: an arc consistent network.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
    - One domain is empty $\Rightarrow$ no solution
    - Each domain has a single value $\Rightarrow$ unique solution
    - Some domains have more than one value $\Rightarrow$ there may or may not be a solution
      Need to solve this new (usually simpler) CSP: same constraints, domains have been reduced

Complexity of the arc consistency algorithm

Worst-case complexity of this procedure:

- let the max size of a variable domain be d
- let the number of constraints be e
- complexity is $O(ed^3)$

Some special cases are faster:

- e.g. if the constraint graph is a tree, arc consistency is $O(ed)$

Finding solutions when AC finishes

If some variable domains have more than one element $\Rightarrow$ search

- Advanced alternatives (beyond this course):
- Variable and arc ordering heuristics speed up arc consistency and search (by an order of magnitude).
- Split a domain, then recursively solve each part.
- Use *conditioning* (fix a variable and prune its neighbours' domains) or *cutset conditioning* (instantiate, in all ways, a set of variables such that the remaining constraint graph is a tree).
- Many other heuristics for exploiting graph structure.

Final note on hard and soft constraints

- Given a set of variables, assign a value to each variable that either
    - satisfies some set of constraints: **satisfiability problems** — "hard constraints"
    - minimises some cost function, where each assignment of values to variables has some cost: **optimisation problems** — "soft constraints"
    - For soft constraint optimisation problems, *value propagation* algorithms are used in the same way that constraint propagation algorithms can be used.
    - In fact, the abstract algebra underpinning these two methods — the generalised distributive law applied to c-semirings — is identical, and the same as belief propagation algorithms use in Bayes-nets, message passing schemes used for turbo-codes, and Nash propagation algorithms used in graphical games,...
- Many problems are a mix of hard and soft constraints (called constrained optimisation problems).

Propositions and Inference

Propositions and Inference

- What is logic?
- Propositional logic: Syntax and Semantics
- Example of using logic to represent a problem
- Two types of problems:
    - Validity
    - (Back to) satisfiability

What is logic?

- A formal language to represent sets of states
- A convenient abstraction for dealing with many states
- Recall in PRM: We can view a vertex in a roadmap to represent a set of “nearby” states
- Regardless of whether there’s a natural notion of “near” or not (i.e., not a metric space), we can use logic to group different states together
- E.g.:

  I have a laptop $\Rightarrow$includes any brand & model

  There is a laptop on the table $\Rightarrow$can be at any position on the table

Propositions

- An interpretation is an assignment of values to all variables.
- A model is an interpretation that satisfies the constraints (as in CSPs).
- Often we don't want to just find a model, but want to know what is true in all models.
- A proposition is statement that is true or false in each interpretation.

Why propositions?

- Specifying logical formulae is often more natural than filling in tables
- It is easier to check correctness and debug formulae than tables
- We can exploit the Boolean nature for efficient reasoning
- We need a language for asking queries (of what follows in all models) that may be more complicated than asking for the value of a variable
- It is easy to incrementally add formulae
- It can be extended to infinitely many variables with infinite domains (using logical quantification)

A formal language

The formal language representation and reasoning system is made up of:

- syntax: specifies the legal sentences

  E.g. `for x in range(len(input)):`
- semantics: specifies the meaning of the symbols

These are usually associated with a reasoning theory or proof procedure: a (nondeterministic) specification of how an answer can be produced.

Many types of logic:

- Propositional logic
- Predicate / first order logic
- High order logic

Semantics — Human's view of semantics

Step 1 Begin with a task domain.

Step 2 Choose atoms in the computer to denote propositions. These atoms have meaning to the knowledge base designer.

Step 3 Tell the system knowledge about the domain.

Step 4 Ask the system questions.

— The system can tell you whether the question is a logical consequence.

— You can interpret the answer with the meaning associated with the atoms.

Role of semantics

In computer:

- $light1\_broken \Leftarrow sw\_up$
  $\wedge power \wedge unlit\_light1$
- $sw\_up.$
- $power \Leftarrow lit\_light2$
- $unlit\_light1$
- lit_light2

Conclusion: $light1\_broken$

In user's mind:

- $light1\_broken$: light #1 is broken
- sw_up: switch is up
- $power$: there is power in the building
- $unlit\_light1$: light #1 isn't lit
- lit_light2: light #2 is lit

- The computer doesn't know the meaning of the symbols
- The user can interpret the symbol using their meaning

Simple language: propositional definite clauses

- An atom is a symbol (starting with a lower case letter)
- A sentence (or body) is an atom or is of the form $s_1 \wedge s_2$ where s_1 and s_2 are sentences.
- A definite clause is an atom or is a rule of the form $h \Leftarrow s$ where h is an atom and s is a sentence.
- A knowledge base is a set of definite clauses

Propositional logic — Syntax

An atomic proposition, or just an atom, is a symbol.

We use the convention that propositions consist of letters, digits and the underscore (_) and start with a lower-case letter.

Atoms are known to either be true or false

Are these propositions?

- What is the distance between Mars and Earth?
- $x + 2 = 2x$
- $x + 2 = 2x$ when $x = 1$
- $2x < 3x$

Atoms are often represented with a symbol called a *propositional variable*, e.g. p, q (see the notes in Assignment 0).

Propositional logic — Syntax

Complex propositions, or sentences, can be built from simpler propositions using logical connectives:
Bracket(()), negation ($\neg$), and ($\wedge$), or ($\vee$), implication ($\Rightarrow$), biconditional ($\Leftrightarrow$)

$\neg p$	"not p"	*negation*
$p \wedge q$	"p and q"	*conjunction*
$p \vee q$	"p or q or (p and q)"	*disjunction*
$p \Rightarrow q$	"if p then q"	*implication*
$p \Leftrightarrow q$	"p iff q"	*biconditional* or *equivalence*

Rules for evaluating truth

$\neg p$	is **TRUE** iff p is **FALSE**
$p \wedge q$	is **TRUE** iff p is **TRUE** and q is **TRUE**
$p \vee q$	is **TRUE** iff p is **TRUE** or q is **TRUE**
$p \Rightarrow q$	is **TRUE** iff p is **FALSE** or q is **TRUE**
$p \Rightarrow q$	is **FALSE** iff p is **TRUE** and q is **FALSE**
$p \Leftrightarrow q$	is **TRUE** iff $p \Rightarrow q$ and $q \Rightarrow p$

Semantics

- An interpretation I assigns a truth value to each atom.
- A sentence $s_1 \wedge s_2$ is true in I if s_1 is true in I and s_2 is true in I.
- A rule $h \Leftarrow b$ is false in I if b is true in I and h is false in I. The rule is true otherwise.
- A knowledge base KB is true in I if and only if every clause in KB is true in I.

Models and Logical Consequence

- A model of a set of clauses is an interpretation in which all the clauses are **TRUE**.
- If KB is a set of clauses and g is a conjunction of atoms, g is a logical consequence of KB, written $KB \models g$, if g is **TRUE** in every model of KB.
- That is, $KB \models g$ if there is no interpretation in which KB is **TRUE** and g is **FALSE**.

Simple Example

$$KB = \begin{cases} p \Leftarrow q. \\ q. \\ r \Leftarrow s. \end{cases}$$

	p	q	r	s	*Is a model?*
I_1	**TRUE**	**TRUE**	**TRUE**	**TRUE**	is a model of KB
I_2	**FALSE**	**FALSE**	**FALSE**	**FALSE**	not a model of KB
I_3	**TRUE**	**TRUE**	**FALSE**	**FALSE**	is a model of KB
I_4	**TRUE**	**TRUE**	**TRUE**	**FALSE**	is a model of KB
I_5	**TRUE**	**TRUE**	**FALSE**	**TRUE**	not a model of KB

Which of p, q, r, s logically follow from KB?

$KB \models p$, $KB \models q$, $KB \not\models r$, $KB \not\models s$

User's view of Semantics — How logic is used to represent a problem

Formulate information as propositional logic sentences, to create a knowledge base (KB)

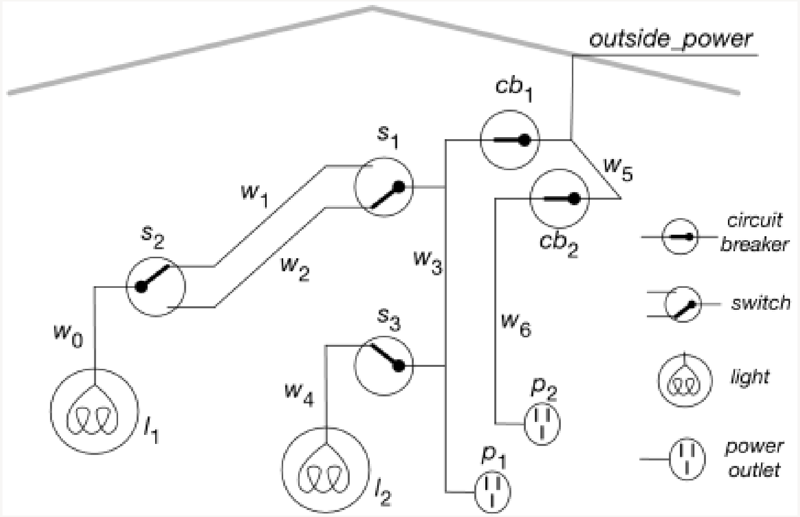
1. Choose a task domain: intended interpretation.
2. Associate an atom with each proposition you want to represent.
3. Tell the system clauses that are true in the intended interpretation: axiomatizing the domain.
4. Ask questions about the intended interpretation.
5. If $KB \models g$, then g must be true in the intended interpretation.
6. Users can interpret the answer using their intended interpretation of the symbols.

Computer's view of semantics

- The computer doesn't have access to the intended interpretation.
- All it knows is the knowledge base.
- The computer can determine if a formula is a logical consequence of KB.
- If $KB \models g$ then g must be true in the intended interpretation.
- If $KB \not\models g$ then there is a model of KB in which g is false. This could be the intended interpretation.

Deduction: Sentences entailed by the current KB can be added to the KB

Electrical Environment



Representing the Electrical Environment

$$
\begin{aligned}
& & lit\_l_1 &\Leftarrow live\_w_0 \land ok\_l_1 \\
light\_l_1. & & live\_w_0 &\Leftarrow live\_w_1 \land up\_s_2. \\
light\_l_2. & & live\_w_0 &\Leftarrow live\_w_2 \land down\_s_2. \\
down\_s_1. & & live\_w_1 &\Leftarrow live\_w_3 \land up\_s_1. \\
up\_s_2. & & live\_w_2 &\Leftarrow live\_w_3 \land down\_s_1. \\
up\_s_3. & & lit\_l_2 &\Leftarrow live\_w_4 \land ok\_l_2. \\
ok\_l_1. & & live\_w_4 &\Leftarrow live\_w_3 \land up\_s_3. \\
ok\_l_2. & & live\_p_1 &\Leftarrow live\_w_3. \\
ok\_cb_1. & & live\_w_3 &\Leftarrow live\_w_5 \land ok\_cb_1. \\
ok\_cb_2. & & live\_p_2 &\Leftarrow live\_w_6. \\
live\_outside. & & live\_w_6 &\Leftarrow live\_w_5 \land ok\_cb_2. \\
& & live\_w_5 &\Leftarrow live\_outside.
\end{aligned}
$$

Terminology

- A sentence is **valid**: Its truth value is **TRUE** for all possible interpretations

  E.g. $p \vee \neg p$
- A sentence is **satisfiable**: Its truth value is **TRUE** for at least one of the possible interpretations. E.g. $\neg p$

  Everything that's valid is also satisfiable
- A sentence is **unsatisfiable**: Its truth value is **FALSE** for all possible interpretations. E.g. $p \wedge \neg p$
- For propositional logic, we can always decide if a sentence is valid/satisfiable/unsatisfiable in finite time (decidable problems).

Validity

Terminology

A model of a sentence: An interpretation that makes the sentence to be true

A sentence A entails another sentence B (denoted as $A \models B$) iff every model of A is also a model of B ($A \Rightarrow B$ is valid)

$$\begin{tikzcd}
\text{A} \arrow[r, "\text{entails}"] \arrow[d, "\text{semantics}"'] & \text{B} \arrow[d, "\text{semantics}"] \\
\text{Interpretations} & \text{Interpretations} \\
\text{Models of A} \arrow[r, "\text{subset}"'] & \text{Models of B}
\end{tikzcd}$$

(Simple) Model checking

KB:

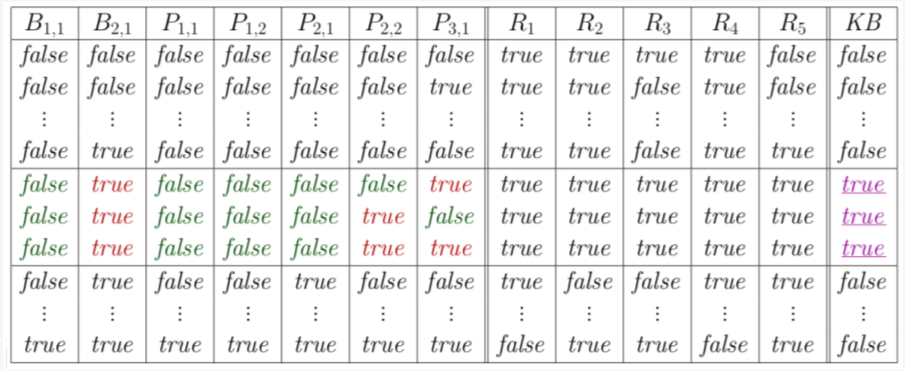
R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

Check if $\neg P_{1,2}$ is entailed by KB

(Simple) Model checking

Enumerate the models:

- All true/false values for $P_{1,1}, B_{1,1}, P_{1,2}, P_{2,1}, B_{2,1}, P_{2,2}, P_{3,1}$
- Check if $\neg P_{1,2}$ is true in all models where the knowledge base $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$ is true



$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

(Simple) Model checking

Sound: The result is correct

Complete: It always gives an answer

Complexity: n is # propositional variables

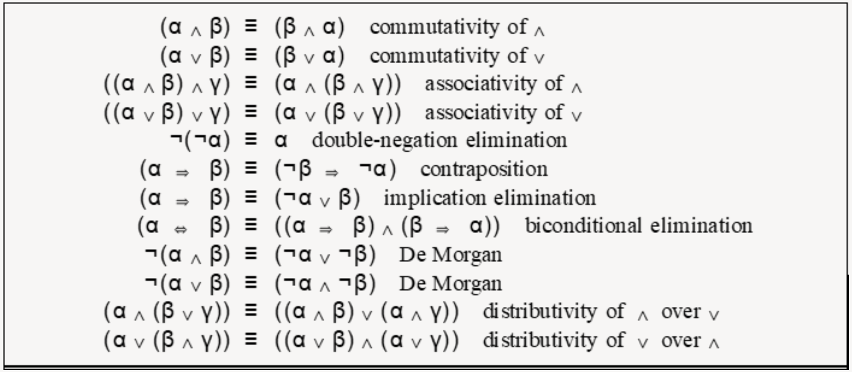
- Time: $O(2^n)$ $\Leftarrow$Bad!
- Space: $O(n)$

Theorem proving — Check validity without checking all models

Use logical equivalences:

α, β sentences (atomic or complex)

Two sentences are logically equivalent iff true in the same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$



$(\alpha \wedge \beta)$	$\equiv$	$(\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv$	$(\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv$	$(\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv$	$(\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv$	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv$	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

Theorem proving — Check validity without checking all models

Transformations for logical expressions:

Modus ponens (*mode that affirms*):

$$\frac{\alpha \Rightarrow \beta \qquad \alpha}{\beta}$$

Modus tollens (*mode that denies*):

$$\frac{\alpha \Rightarrow \beta \qquad \neg\beta}{\neg\alpha}$$

And-elimination (*for a conjunction, any of the conjuncts can be inferred*):

$$\frac{\alpha \wedge \beta}{\alpha}$$

Theorem proving — Natural deduction

KB:

S_1: $\neg P_{1,1}$

S_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

S_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

S_4: $\neg B_{1,1}$

S_5: $B_{2,1}$

Check if $\neg P_{1,2}$ is entailed by KB?

Theorem proving — Natural deduction

KB:

S_1: $\neg P_{1,1}$

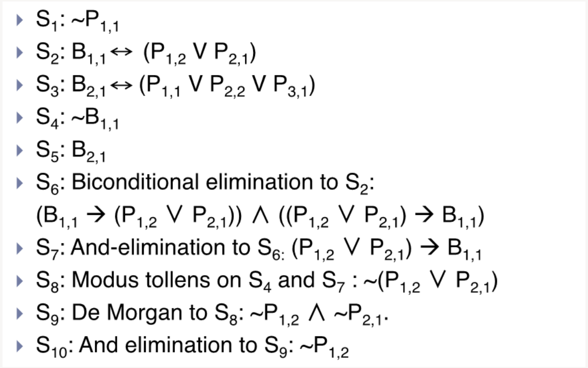
S_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

S_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

S_4: $\neg B_{1,1}$

S_5: $B_{2,1}$

Check if $\neg P_{1,2}$ is entailed by KB?



- S_1: $\sim P_{1,1}$
- S_2: $B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$
- S_3: $B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- S_4: $\sim B_{1,1}$
- S_5: $B_{2,1}$
- S_6: Biconditional elimination to S_2:
  $(B_{1,1} \rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \rightarrow B_{1,1})$
- S_7: And-elimination to $S_{6:}$ $(P_{1,2} \vee P_{2,1}) \rightarrow B_{1,1}$
- S_8: Modus tollens on S_4 and S_7 : $\sim(P_{1,2} \vee P_{2,1})$
- S_9: De Morgan to S_8: $\sim P_{1,2} \wedge \sim P_{2,1}$.
- S_{10}: And elimination to S_9: $\sim P_{1,2}$

Theorem proving —- Natural deduction using search

State space: All possible sets of sentences

Action space: All inference rules (see more soon)

World dynamics: Apply the inference rule to all sentences that match the above the line part of the inference rule. Become the sentence that lies below the line of the inference rule

Initial state: Initial knowledge base

Goal state: The state contains the sentence we're trying to prove

This procedure is sound, but it may not be complete, depending on whether we can provide a complete list of inference rules. However, when we can, the branching factor can be very high.

Resolution: A single inference rule (action)

Given α, β and γ sentences (atomic or complex):

$$\frac{\begin{array}{r}\alpha \vee \beta \\ \neg\beta \vee \gamma\end{array}}{\alpha \vee \gamma}$$

This single inference rule is sound and complete only when applied to propositional logic sentences written in Conjunctive Normal Form (CNF)

Conjunctive Normal Form (CNF)

Conjunctive Normal Form (CNF)

Conjunctions of disjunctions, e.g. $(\neg A \vee B) \wedge (C \vee D) \wedge (E \vee F)$

Some terminology:

- **Clause**: a disjunction of literals. e.g., $(\neg A \vee B)$
- **Literals**: variables or the negation of variables, e.g., $A, \neg A, B$

CNF is quite useful for model checking and for solving satisfiability problems too — in fact, we've seen it before! Where?

Every sentence in propositional logic can be written in CNF

Three step conversion:

1. Eliminate arrows using definitions
2. Drive in negations using De Morgan's Laws
3. Distribute OR over AND

E.g. Convert $(A \vee B) \Rightarrow (C \Rightarrow D)$ to CNF

1. Eliminate arrows: $\neg(A \vee B) \vee (\neg C \vee D)$
2. Drive in negations: $(\neg A \wedge \neg B) \vee (\neg C \vee D)$
3. Distribute OR over AND: $(\neg A \vee \neg C \vee D) \wedge (\neg B \vee \neg C \vee D)$

Automated theorem proving – Resolution refutation

Three steps:

1. Convert all sentences into CNF
2. Negate the desired conclusion
3. Apply resolution rule until: (i) derive false (a contradiction), or (ii) can't apply the rule anymore

Sound and complete (for propositional logic):

- If we derive a contradiction, the conclusion follows from the axioms
- If we can't apply any more, the conclusion cannot be proved from the axioms

Resolution refutation — Examples

KB: $(P \lor Q) \land (P \Rightarrow R) \land (Q \Rightarrow R)$
Does KB $\models R$?

KB: $(P \land \neg P)$
Does KB $\models R$?

Satisfiability (again)

Davis-Putnam-Logemann-Loveland (DPLL) tree search algorithm

An assign-and-simplify strategy

- Consider a search tree where at each level we consider the possible assignments to one variable, say V
- On one branch, we assume V is **FALSE** and on the other that it is **TRUE**
- Given an assignment for a variable, we can simplify the sentence and then repeat the process for another variable
- DPLL is a backtracking algorithm for systematically doing this

Davis-Putnam-Logemann-Loveland (DPLL) tree search algorithm

The algorithm is building a solution while trying assignments — you have a partial solution which might prove successful or not as you go.

Provides an ordering of which variables to check first — this is the key to its efficiency

Definitions:

- Clause: a disjunction of literals
- Literals: variables or the negation of variables
- Unit clause: clause which has exactly one literal which is still unassigned and the other (assigned) literals are all assigned to **FALSE**
- Pure variable: appears as positive only or negative only in S

If the current assignment is valid, you can determine the value of the unassigned pure variable in a unit clause, because the literal must be true

E.g. if we have $(X1 \vee X2 \vee X3) \wedge (\ X1 \vee \ \ X4 \vee X5) \wedge (\ X1 \vee X4)$ and we have already assigned $X1 =$ **FALSE**: then choose $X4$ and set to **TRUE**

Davis-Putnam-Logemann-Loveland (DPLL) tree search algorithm

Provides an ordering of which variables to check first

```
DPLL(Sentence $S$)
    If $S$ is empty, return TRUE
    If $S$ has an empty clause, return FALSE
    If $S$ has a unit clause $U$, return DPLL($S(U)$)
        Unit clause: consists of only 1 unassigned literal
        $S(U)$ means a simplified $S$ after a value is assigned to $U$
    If $S$ has a pure variable $U$, return DPLL($S(U)$)
        Pure variable: appears as positive only or negative only in $S$
    Pick a variable $v$,
    If DPLL($S(v)$) then return TRUE
    Else return DPLL($S( v)$)
```

Heuristics to pick the variable: Max #occurrences, Min size clauses. Basically, pick the most constrained variable first (if it's going to fail, better fail early)

DPLL

Sound (the result is correct)

Complete (it always gives an answer)

Speed and memory consumption depends a lot on:

- Which symbol is being assigned first
- Which assignment is being followed first

A **lot** of other methods, heuristics, and meta heuristics have been proposed for SAT problems, e.g. `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

Attributions and References

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati for their materials.

Other materials derived from Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 3E, Prentice Hall, 2009.

Some materials are derived from Chris Amato (Northeastern University), Tuomas Sandholm (CMU) and Kevin Leyton-Brown (UBC).

All remaining errors are Archie's — please email if you find any: archie.chapman@uq.edu.au