Computer Science Logic

Basic Intro

- Revise propositional logic from ELA
  - Truth tables
  - Formula trees
  - Valid / invalid functions (tautologies / contradictions)
  - Satisfiability
    - A is valid if ¬A is not satisfiable
    - A is satisfiable if ¬A is not valid
  - Consistency (in sets of formulae)

Logical Consequence

- B is a logical consequence of $A_1 \dots A_n$ if every interpretation that makes $A_1 \dots A_n$ true also makes B true
  - Notation: $A_1 \dots A_n \models B$
- Logical consequence can be reduced to validity:
  - $A_1 \dots A_n \models B$ iff $(A_1 \wedge \dots \wedge A_n) \rightarrow B$ is valid
  - A.k.A "Deduction Theorem"
- Logical consequence can be reduced to consistency:
  - $A_1 \dots A_n \models B$ iff the set $\{A_1, \dots, A_n, \neg B\}$ is <u>not</u> consistent

Logical Equivalence

- A and B are logically equivalent if they evaluate to the same truth value in every interpretation.
  - Notation: $A \equiv B$
- Logical equivalence can be reduced to validity:
  - $A \equiv B$ iff $A \leftrightarrow B$ is a valid formula

Validity and Effectiveness

- Is there an algorithm (an effective procedure) that can decide if a formula is valid?
- An algorithm must:
  - consist of an exact sequence of steps
  - not require any originality from the person/machine executing it
  - be universal, ie. must work for all formulae.

Boolean Logic is Decidable

- Validity and satisfiability for Boolean logic are decidable: a program can be written to take any formula or set of formulae and give a "yes/no" answer.

- But Why?

- In every formula, there are finitely many interpretations. We just compute each one and compare the results:
  - Every result is 1: valid
  - At least one result is 1: satisfiable.
- This can be done with the truth table algorithm
  - This is slow though, and the time complexity is exponential
  - n propositional values $\rightsquigarrow$ 2^n lines in the table
  - E.g. $n = 80$, 1000000 lines/second $\rightarrow$ 38 billion years.

The Tableau Procedure

- For any formula A, if it is not valid then an interpretation exists where A evaluates as 0.
  - This is called a counter-interpretation
- For any formula A, the tableau procedure tries to find a counter interpretation for A. via a set of simple steps (an algorithm).
- It is sound and complete.

- Overview

- Form a 2-column tableau, where...
  - Formulae we want to be 1 go on the left
  - Formulae we want to be 0 go on the right
- At each step, we "unfold" a formula in the table into simpler sub-formulae until the procedure stops.
- The procedure may stop if:
  - There are no more "unfoldable" formulae left and we have reached the propositional values.
    At this point we can turn the tableau into a counter-interpretation and say that the initial formula is NOT VALID.
  - We reach a contradictory tableau where the same formula appears on the left and right.
    This means that there is no counter-interpretation and we can say that the initial formula is VALID.

- Unfolding

- At each step, chose ANY formula for either column that is not a propositional value and has not been chosen before.
- Find the outermost logical connective.
- Apply the corresponding rule.

- Rules:

- There are 10 rules: 2 for each of $\vee$, $\wedge$, $\neg$, $\rightarrow$ and $\leftrightarrow$

- $\rightarrow$ false

1	0
↓	$A \rightarrow B$ ↓
A	B

- $\rightarrow$ true

1	0
$A \rightarrow B^{\bullet}$ ↓	↓
	A

OR

1	0
$A \rightarrow B^{\bullet}$ ↓	↓
B	

The dot on $A \rightarrow B^{\bullet}$ shows that there are different cases to pursue.

- $\neg$ false

1	0
↓	$\neg A$ ↓
A	

- $\neg$ true

1	0
$\neg A$ ↓	↓
	A

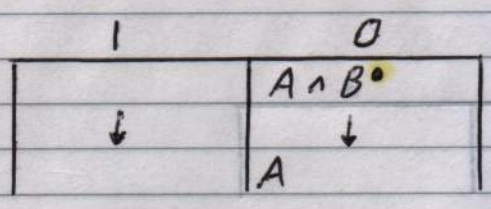
- $\vee$ false

1	0
↓	$A \vee B$ ↓
	A
	B

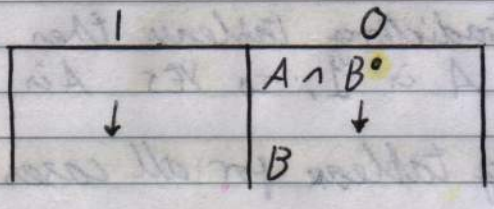
- $\wedge$ true

1	0
$A \wedge B$ ↓	↓
A	
B	

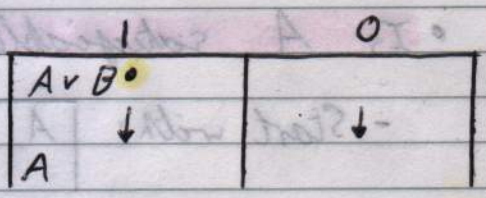
- $\wedge$ false



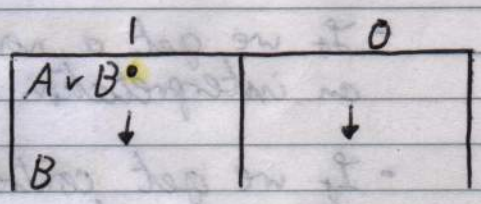
OR



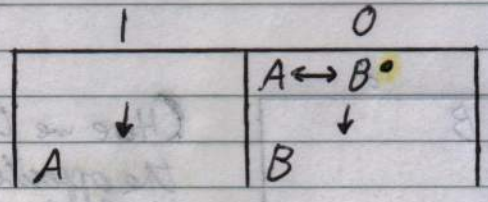
- $\vee$ true



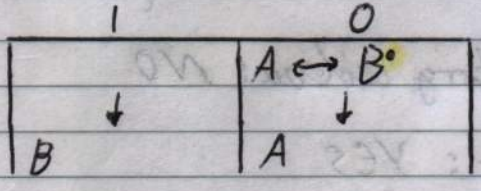
OR



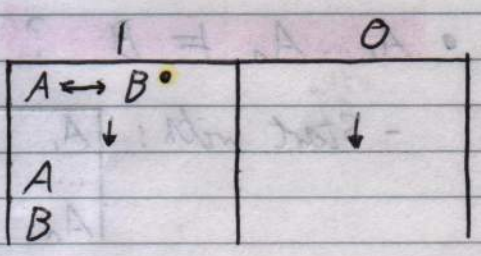
- $\leftrightarrow$ false



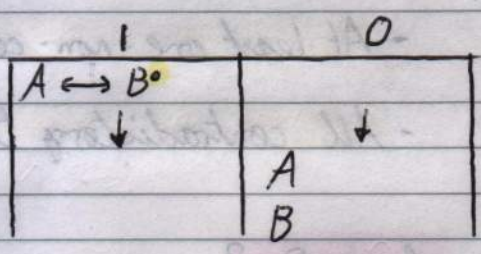
OR



- $\leftrightarrow$ true



OR



- When to Stop?

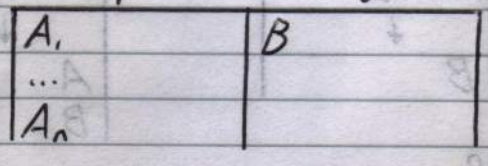
- If there is <u>at least one case</u> where no "unfoldable" formulae are left the initial formula is NOT VALID.
- If <u>every case</u> gives a contradictory tableau, the initial formula is valid.

- Satisfiability, Consequence and Equivalence?

- **Is A satisfiable?**
  - Start with:

  - Apply rules
  - If we get a non-contradictory tableau then it gives us an interpretation where A is 1, so YES A is satisfiable.
  - If we get contradictory tableax for all cases then NO A is not satisfiable.

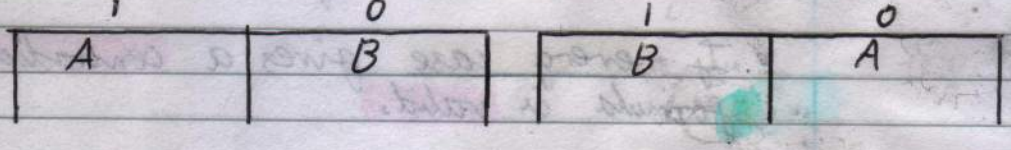
- **$A_1 \dots A_n \models B$?**
  - Start with:



(Here we try to prove the opposite, b/c this procedure is best for "is there a"-style questions)

  - Apply rules
  - At least one non-contradictory tableau: NO
  - All contradictory tableax: YES

- **$A \equiv B$?**
  - Again, we go after the opposite.
  - Start with: Case 1: Case 2:



  - Apply rules:
  - At least one non-contradictory tableau: NO
  - All contradictory tableax: YES

Boolean Functions

- An n-ary Boolean function f is a function where:
  - the **domain** is the set $\{0,1\}^n$ of all n-tuples of 0 and 1
  - the **range/co-domain** is the set $\{0,1\}$
- Symbolically: $f : \{0,1\}^n \to \{0,1\}$
- There are 2^{2^n} Boolean functions (n-ary).

- Unary Boolean Functions

$f_1(p) = 0$	the constant 0 function
$f_2(p) = 1$	the constant 1 function
$f_3(p) = p$	the identity function
$f_4(p) = \neg p$	the negation function

Boolean Decision Diagrams (BDDs)

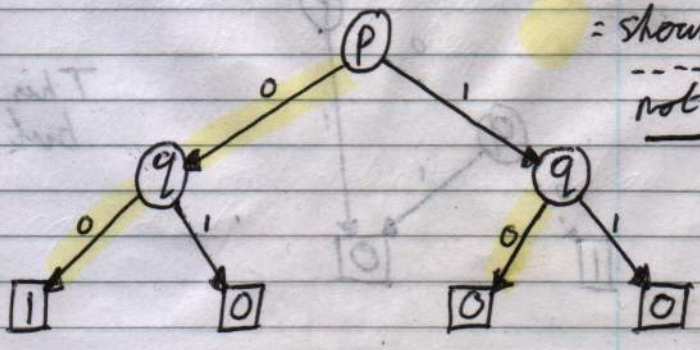
- Why Are They Needed?

- Truth tables are too large
- Formulae are compact, but to determine something like validity we would have to go through exponentially many interpretations.
- Tableax can have exponentially many cases.
- It is not known whether better representations exist that work in every case. But it makes sense to look for them, so we consider BDDs.

- Binary Decision Trees to Represent Boolean Functions

p	q	$\neg(p \vee q)$
0	0	1
0	1	0
1	0	0
1	1	0

=

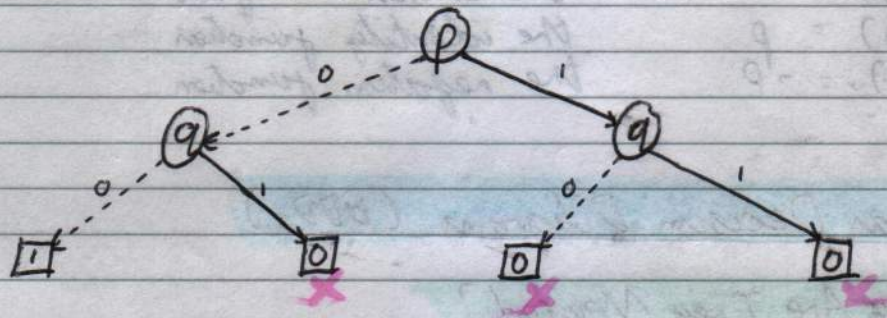


: should be not

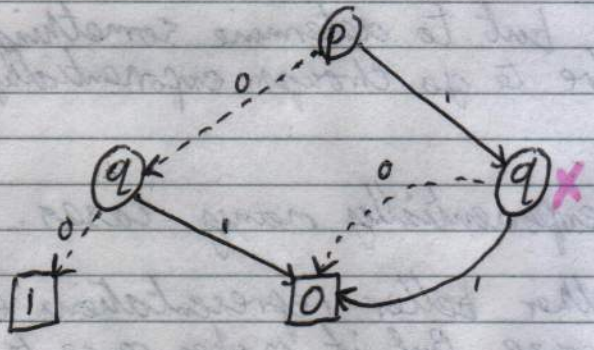
- Optimising Binary Decision Trees

- For an n-ary Boolean function a non-optimised binary decision tree will have 2^n paths.
- However, trees often have some redundancy that can be removed.
- We look at 3 techniques:
  - Removing duplicate terminals → Below.
  - Removing redundant tests → Below.
  - Removing duplicate non-terminals. → Next page.

Consider the tree for $\neg(p \vee q)$:

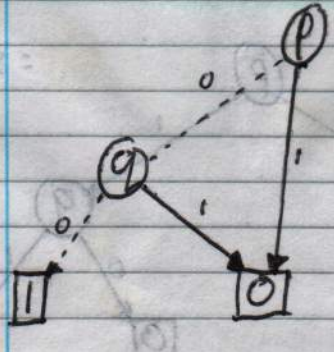


The three duplicate terminals ✗ can be combined:



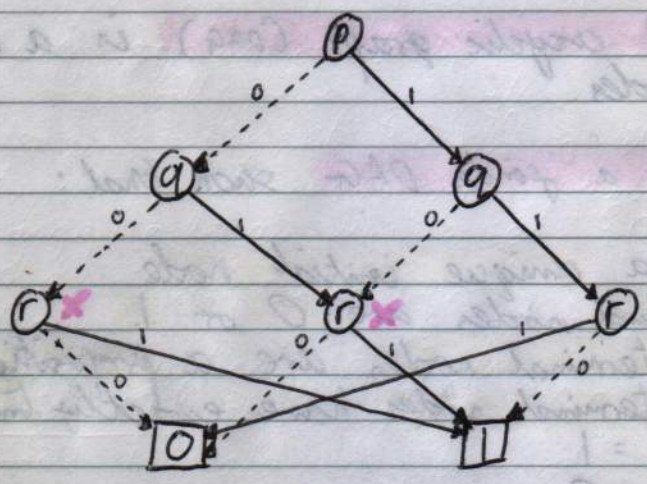
(NB. This is no longer a tree; it is a Binary Decision Diagram)

The redundant test ✗ can be removed:

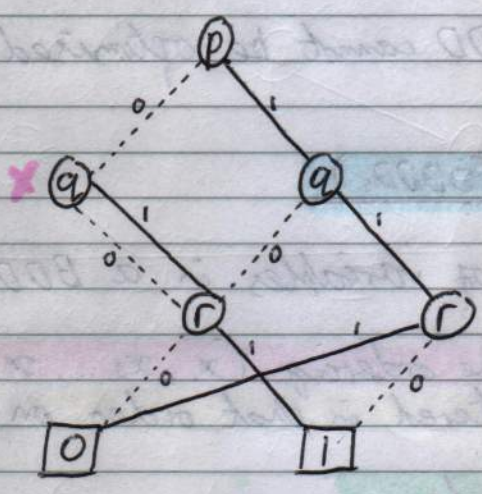


This BDD still shows $\neg(p \vee q)$ but is a lot simpler.

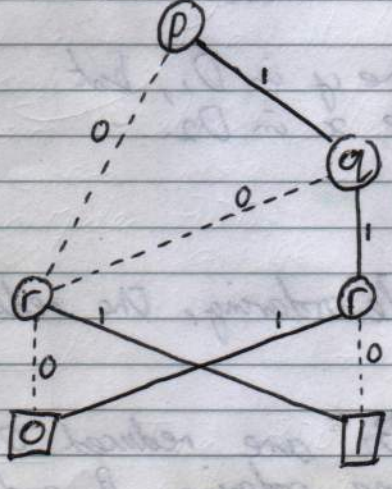
Consider the tree below:



The two duplicate non-terminals can be merged as they do the same thing:



The redundant test can now be removed:



This BDD contains the same information, but contains less nodes.

- BDD Specifics

- A directed acyclic graph (DAG) is a directed graph with no cycles.
- A BDD is a finite DAG such that:
  - It has a unique initial node
  - All terminal nodes are 0 or 1
  - All non-terminal nodes are a propositional value
  - All non-terminal nodes have exactly two edges leading away
    - o —— = 1
    - o ----- = 0
  - The same propositional value cannot appear twice in any path!
- A reduced BDD cannot be optimised ~~by~~ any further.

Ordered BDDs (OBDDs)

- The ordering of variables in a BDD can affect size.
- A BDD has the ordering $(x_1, x_2 \dots x_n)$ if variables are always encountered in that order on a path from the root.

- Compatible Orderings

- Two BDDs D_1 and D_2 have compatible orderings if there are no two variables x and y such that:
  - x comes before y in D_1, but
  - y comes before x in D_2.

- Important Notes

- Given a variable ordering, the reduced OBDD for a function is unique.
  - If B_1 and B_2 are reduced OBDDs for the same function with the same ordering, B_1 and B_2 are identical.
- If the function $f(p_1, \dots, p_n)$ does not depend on the value of p_i, then no reduced OBDD contains p_i.

Decision Algorithms with Reduced OBDDs

- **Deciding validity**
  - A is valid iff the reduced OBDD for A is [1].
- **Deciding satisfiability**
  - A is satisfiable iff the reduced OBDD for A is not [0].
- **Deciding logical equivalence**
  - A and B are equivalent iff the reduced OBDDs for A and B with compatible variable orderings have identical structure.
- **Deciding logical consequence**
  - $A_1 \dots A_n$ logically imply B iff the reduced OBDD for $(A_1 \wedge \dots \wedge A_n \wedge \neg B)$ is [0].

Algorithms on OBDDs

- We will look at:
  - apply
  - restrict
  - exists
- These are basic algorithms used in program verification.

- apply

- If we have two reduced OBDDs B_f and B_g with compatible variable ordering, representing the functions f and g, then apply$(*, B_f, B_g)$ computes the reduced OBDD B_{f*g} representing $f * g$ where $* \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$.
- **Initial Steps**
  - Determine the compatible variable ordering
  - Name every node in the input OBDDs so we can refer to them.

• Remaining Steps

- We build B_{f*g} recursively, starting at the root.
- At each step, we compare two nodes:
  - n_f from B_f (labelled $l(n_f)$)
  - n_g from B_g (labelled $l(n_g)$)
- We proceed according to one of four cases, stopping when we reach a terminal node on each path.
- The resulting OBDD B_{f*g} may not be reduced, so reduction steps are then applied if possible.
  - This can also be done during the process.

• Case One

- If n_f and n_g are terminal nodes:
  - We compute the truth value of $l(n_f) * l(n_g)$
  - Create a terminal node labelled by the result.

• Case Two

- If n_f and n_g are labelled by the same variable: (say, p)
  - Create a new node labelled with p,
  - Add a dashed line to the comparison $[lo(n_f), lo(n_g)]$
  - Add a solid line to the comparison $[hi(n_f), hi(n_g)]$

• Case Three

- If n_f is non-terminal, and n_g is terminal OR $l(n_f)$ is before $l(n_g)$ in the ordering:
  - Create a node labelled with $l(n_f)$
  - Add a dashed line to the comparison $[lo(n_f), n_g]$
  - Add a solid line to the comparison $[hi(n_f), n_g]$

• Case Four

- If n_g is non-terminal, and n_f is terminal or $\ell(n_g)$ is before $\ell(n_f)$ in the ordering:
  - Create a node labelled with $\ell(n_g)$
  - Add a dashed line to the comparison $[n_f, lo(n_g)]$
  - Add a solid line to the comparison $[n_f, hi(n_g)]$

• Example at End of Section

- restrict

• If A is a formula and p is a propositional variable, then...

- $A[0/p]$ is the formula A where all instances of p are swapped for 0
- $A[1/p]$ " " " " " " " " " " " " " " 1

• If B_A is the reduced OBDD for the formula A:

- restrict $(0, p, B_A)$ computes the OBDD $B_{A[0/p]}$
- restrict $(1, p, B_A)$ computes the OBDD $B_{A[1/p]}$

• restrict $(0, p, B_A)$

- For each node n in B_A labelled p:
  - redirect incoming edges to $lo(n)$
    - If n is the root, $lo(n)$ is the new root
  - remove n and any outgoing edges
  - remove nodes and edges that are no longer reachable from the root.
  - apply reductions if necessary

• restrict $(1, p, B_A)$

- As above, but redirect to $hi(n)$

• Example at end of Section

- existi

- Given a formula A and propositional value p,

  $\exists p A$ is shorthand for $A[0/p] \vee A[1/p]$

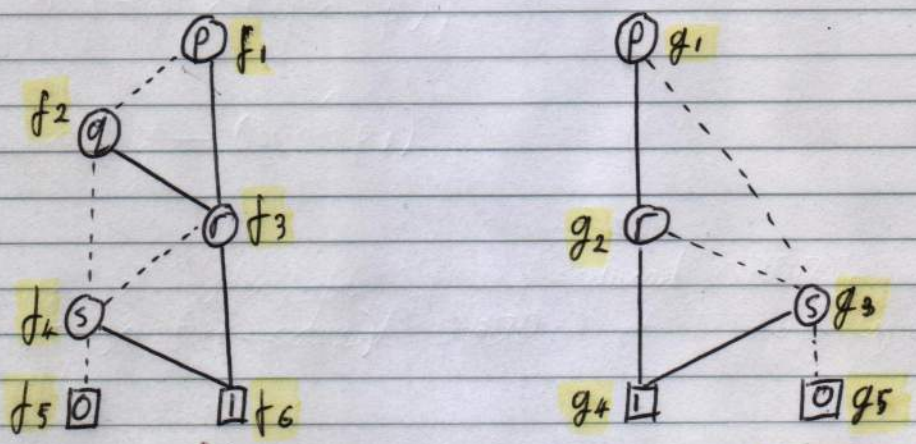
- Example:

  $\exists p(p \wedge q) \equiv (0 \wedge q) \vee (1 \wedge q) \equiv 0 \vee q \equiv q$

- existi (p, B_A) = apply $(\vee$, restrict $(0, p, B_A)$, restrict $(1, p, B_A))$

apply Example: $apply(\vee, B_f, B_g)$

Consider: B_f B_g



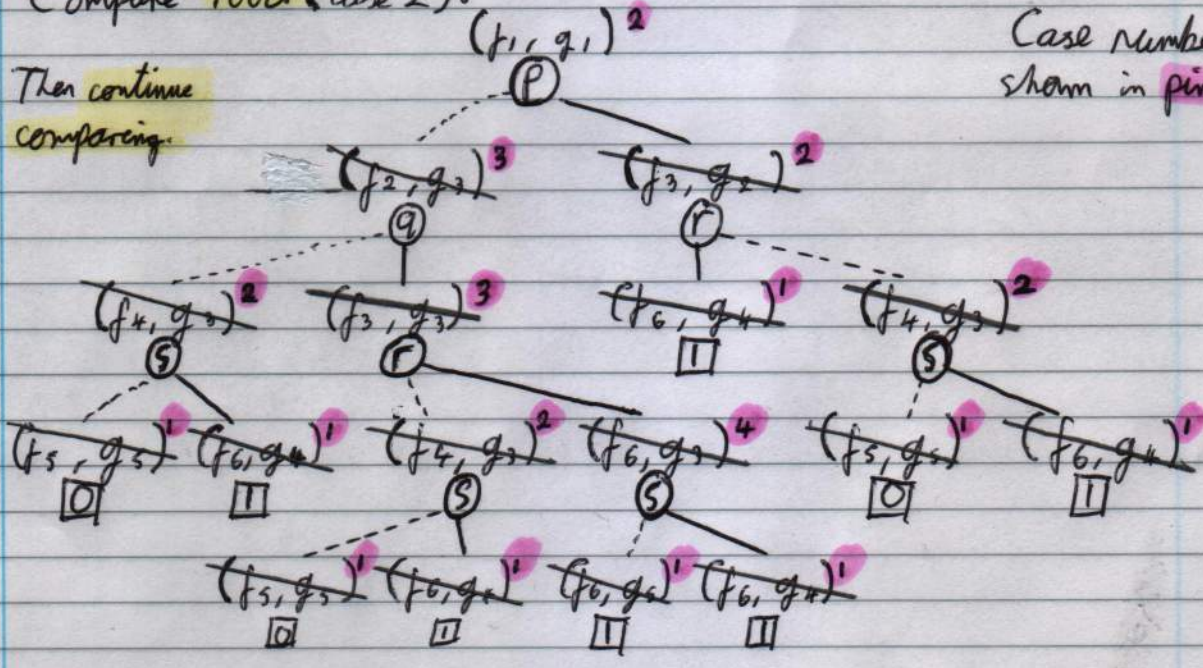
First, find the compatible variable ordering: (p, q, r, s)

Then, name nodes (above).

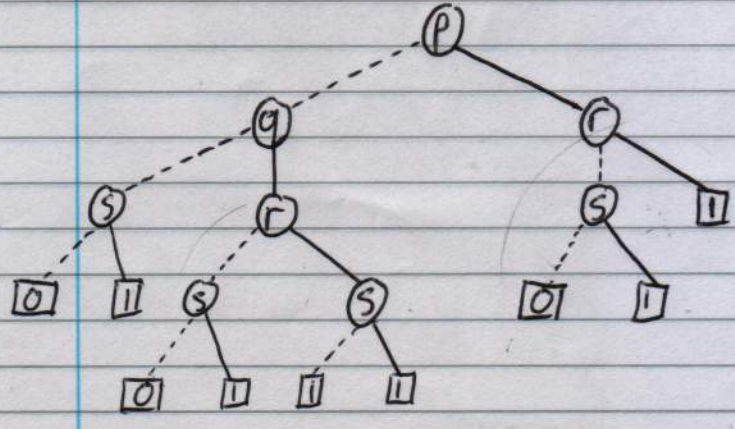
Compare roots (case 2):

Then continue comparing.

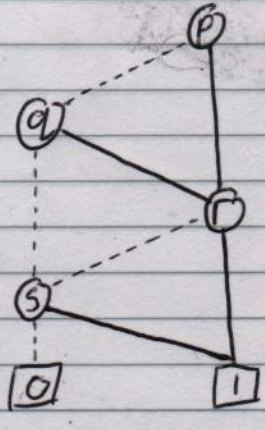
Case number shown in pink



This gives:

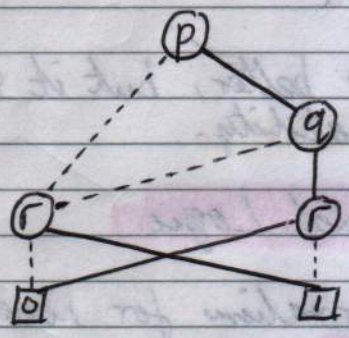


Which simplifies to:

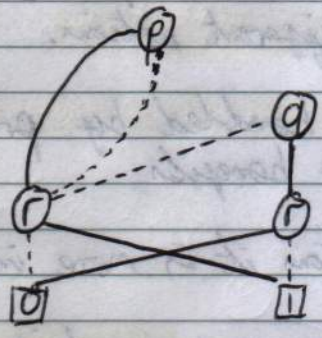


restrict Example restrict(O, q, B_A)

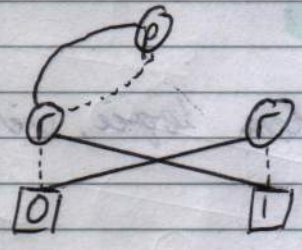
Consider: B_A



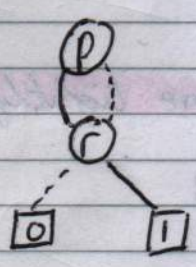
Redirect in-bound edges to lo(q): (lo, b/c O; hi if 1)



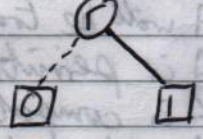
Remove q and out-bound edges:



Remove unreachable nodes:



Reduce:



Modal Logic

- Boolean logic is not always enough - for instance, it only allows finite conjunctions.
- Predicate logic is a little better, but it suffers from increased complexity and undecidability.
- Enter, stage left: **Modal Logic**
  - Modal logics are formalisms for handling different "modes" of truth - modalities.
  - There are many different kinds, each tailored to describe and reason about different notions.
  - All features could be handled by predicate logic, but modal logic has some benefits
    - For many applications it is more intuitive
    - In many cases they are effective (that is, they are decidable/programmable), unlike predicate logic.

Syntax of Modal Logic

- The same as normal Boolean logic, with two new unary connectives:

  $\Box$ (read: "box")

  $\Diamond$ (read: "diamond")

- $\Box$, $\Diamond$ and $\neg$ bind more tightly than $\vee$, $\wedge$ and $\rightarrow$
- Readings of $\Box$ and $\Diamond$:

$\Box A$	$\Diamond A$:
- I know A	- A is compatible with my knowledge
- A is necessary	- A is possible
- A will always be true	- A will be true at some time(s)
- A is obligatory	- A is permitted
- I believe A	- A is compatible with my beliefs
- A holds after every execution of P	- A holds after some execution of P.

Epistemic Reading

Modal Formula	Reading
$\Box\Box p$	p is known to be known
$\Box p \to p$	If p is known then p is true
$\Box p \to \Diamond p$	If Q knows p then p is consistent with Q's knowledge
$\Box p \vee \Box \neg p$	Either Q knows p or Q knows not p
$\Box(p \to q) \to (\Box p \to \Box q)$	"logical omniscience"
$\Box p \to \Box\Box p$	"positive introspection"
$\neg\Box p \to \Box\neg\Box p$	"negative introspection"

$\Box p$ reads as "p is known" or "agent Q knows p" or "I know p"

Deontic Readings

Deontic = "dealing with norms"

$\Box p$ reads as "p is obligatory" or "p ought to be true"

$\Diamond p$ reads as "p is permitted"

Modal Formula	Reading
$(\Box p \wedge \Box q) \to \Box(p \wedge q)$	If both p and q are obligatory, then so is the combination of p and q together
$\Box p \to \Diamond p$	If p is obligatory then p is permitted
$p \to \Box p$	If p is true then it is obligatory

Temporal Readings

- Temporal = dealing with time

$\Box p$ reads as "p will always be true in the future"

$\Diamond p$ reads as "p will be true some time in the future"

Modal Formula	Reading
$\Box p \to p$	If p will always be true then p is true now
$\Box(p \to \Box p)$	Once p is true, p will always be true
$\Diamond \Box p$	Eventually p will be true permanently
$\Box(p \to \Diamond q)$	Whenever p, there is always q afterwards
$\Box \Diamond 1$	Time has no end
$\Box 0$	Time has ended, now.

Terminating Program Executions

$\Box p$ reads as "p will be true after every execution of program P"

$\Diamond p$ reads as "p " " " " some " " " "

Modal Formula	Reading
$(\Diamond p \wedge \Box q) \to \Diamond(p \wedge q)$	If there is an execution ending with p true, and every execution ends with q true, then there is an execution that ends with p and q true.
$(\Diamond p \wedge \Diamond q) \to \Diamond(p \wedge q)$	If there are (possibly different) executions ending with p and q true, then there is an execution ending with $p \wedge q$ true.
$\Box(p \wedge \neg p)$	The program never ends.

Multi-Modal Logics

- **Multi-agent systems**: for the different agents Q_1, Q_2, etc, we have the modalities $\Box_1$, $\Box_2$, ... and $\Diamond_1$, $\Diamond_2$, ...
- **Reasoning about time**: for the future F and past P, we have the modalities $\Box_F$, $\Diamond_F$, $\Box_P$, $\Diamond_P$.
- **Reasoning about programs**: for the different programs P_1 and P_2, we have the modalities $\Box_{P_1}$, $\Diamond_{P_1}$, $\Box_{P_2}$, $\Diamond_{P_2}$.

"Possible World" Semantics

- To avoid ambiguity, the following definitions are assumed:
  - **necessity**: truth in **all** possible worlds.
  - **possibility**: truth in **at least one** possible world.
- We imagine a system of **possible worlds**, in which each world has **some** (maybe none) **alternatives**.
  - Each world follows the laws of Boolean logic:
    - Each world has its own interpretation of propositional values, each of which is 0 or 1.
    - Compound formulae follow normal truth tables.
  - Different worlds may have different interpretations of variables.
  - A formula $\Box p$ is true iff in a world w if p is true in **all worlds** alternative to w.
  - A formula $\Diamond p$ is true in a world w if p is true in **at least one world** alternative to w.

Kripke Frames

- A Kripke frame is a pair $F = (W, R)$, where

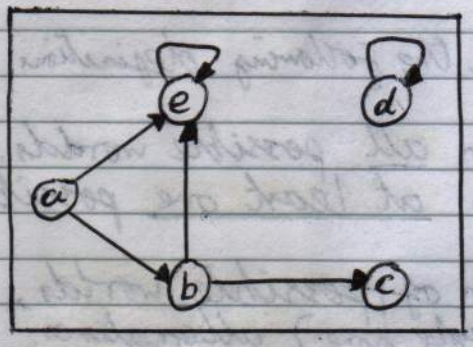
  W = a non-empty set of worlds (or states)

  R = the binary alternativeness/accessibility relation

- If $(a,b) \in R$, we say "a sees b" or
  "b is accessible from a" or
  "b is an alternative to a"

  - We also write aRb

- Eg.



$W = \{a, b, c, d, e\}$

$R = \{(a,b), (a,e), (e,e), (b,c), (d,d)\}$

Kripke Models

- A Kripke model is a triple $M = (W, R, l)$, where

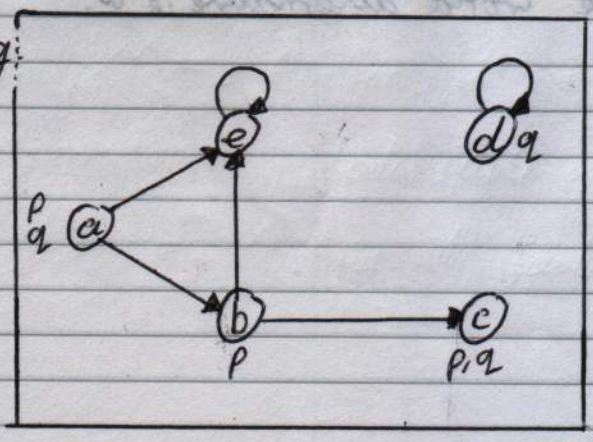
  (W, R) = a Kripke frame

  l = a function that labels each world with the propositional values that are true within it (aka. a valuation)

- We say that the model $M = (W, R, l)$ is based on the frame (W, R).

- There can be many models based on the same frame.

- Eg:



Meaning of Modal Formulae in Kripke Models

- For every modal formula A, model $\mathcal{M}$ and world w...

  $\mathcal{M} \models_w A$ "A is true in $\mathcal{M}$ at w"

  $\mathcal{M} \not\models_w A$ "A is false in $\mathcal{M}$ at w"

Evaluating Modal Formulae in Kripke Models

- In the model $\mathcal{M}$...
  - If p is in the labels of w, then $\mathcal{M} \models_w p$
  - If p is not in the labels of w, then $\mathcal{M} \not\models_w p$
  - $\mathcal{M} \not\models_w A$ iff $\mathcal{M} \models_w \neg A$
  - $\mathcal{M} \models_w A \wedge B$ iff $\mathcal{M} \models_w A$ and $\mathcal{M} \models_w B$
  - $\mathcal{M} \models_w \Diamond A$ iff there is a world u in $\mathcal{M}$ such that wRu and $\mathcal{M} \models_u A$.
    - If w is a "dead-end" then $\Diamond A$ is always false at w.
  - $\mathcal{M} \models_w \Box A$ iff $\mathcal{M} \models_v A$ for all worlds v in $\mathcal{M}$ such that wRv.
    - If w is a "dead-end" then $\Box A$ is always true at w

Truth and Satisfiability in Kripke Models

- A formula A is **true** in model $\mathcal{M}$ iff

  $\mathcal{M} \models_w A$ for all worlds w in $\mathcal{M}$

- A formula A is **satisfiable** in model $\mathcal{M}$ iff

  $\mathcal{M} \models_w A$ for at least one world w in $\mathcal{M}$

Validity and Satisfiability in Kripke Frames

- A formula A is valid in a frame $F = (W, R)$ iff...

  $\mathcal{M} \models_w A$ for all worlds w in all models $\mathcal{M}$ based on F.

- A formula A is satisfiable in a frame $F = (W, R)$ iff...

  $\mathcal{M} \models_w A$ for some world w in some model $\mathcal{M}$ based on F.

- The following are valid in all frames:

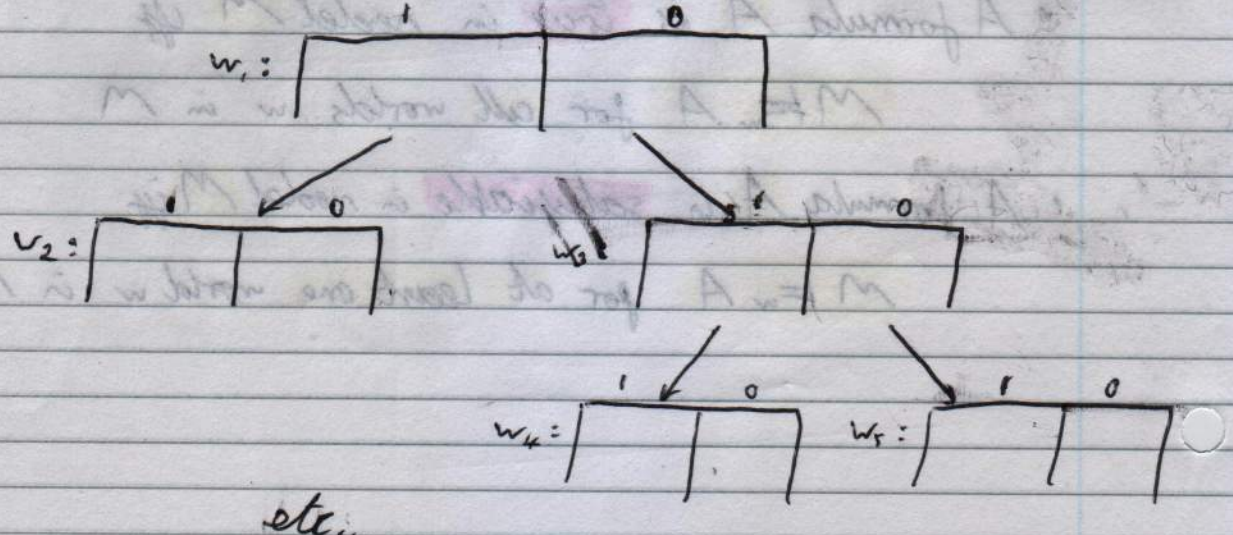
  - valid boolean logic formulae.
  - $\Box p \leftrightarrow \neg \Diamond \neg p$
  - $\Diamond p \leftrightarrow \neg \Box \neg p$
  - $\Box \top$
  - $\Box(p \to q) \to (\Box p \to \Box q)$ (named k)
  - $\Diamond(p \to q) \to (\Box p \to \Diamond q)$
  - $\Box(p \to q) \to (\Diamond p \to \Diamond q)$
  - $\Box(p \wedge q) \leftrightarrow (\Box p \wedge \Box q)$
  - $\Diamond(p \vee q) \leftrightarrow (\Diamond p \vee \Diamond q)$

Kripke Frames & The Tableau Algorithm.

- Given a modal formula A, we will try to find a countermodel to show that A is not valid in all frames.

  - That is, we will find a frame F, a model $\mathcal{M}$ based on F and a world w within $\mathcal{M}$ such that $\mathcal{M} \not\models_w A$.

- We will produce a tableau system; a tree-like Kripke frame where each world has its own 2-column tableau.



etc...

How to Start

Question: Is the formula $\neg\Box\neg p \rightarrow \Box\neg\Box p$ valid in all frames?

- We want a **countermodel**, where $\neg\Box\neg p \rightarrow \Box\neg\Box p$ is false.
- We don't know the world structure, but we know we want at least one world where $\neg\Box\neg p \rightarrow \Box\neg\Box p$ is false.
- This gives:

w_1: | $\neg\Box\neg p \rightarrow \Box\neg\Box p$ |

What Next?

- We **choose** <u>any</u> formula from one of the tableaux that is not a prop. value and has not been "processed" before.
- We find the outermost **logical connective** and apply the appropriate rule.
- A **rule** will:
  - THIS → put some sub-formulae into the columns of an existing tableau.
  - OR ALL OF THIS →
    - create a new world with a new tableau
    - connect the new world to an existing world
    - put some sub-formulae into the tableau of the new world.
- Sometimes a rule will result in **two cases** to explore at once; this creates **two entire new systems**.

When Does it End?

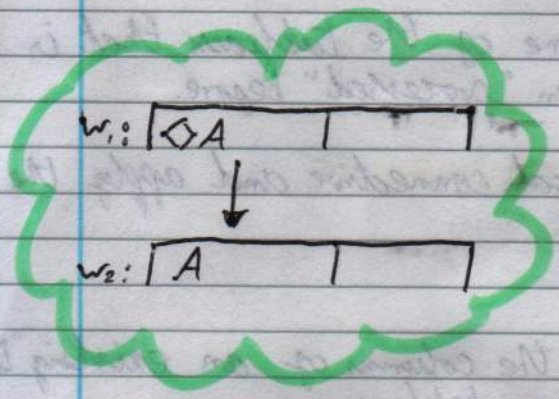
- 2 end cases:
  - When there are **no unfoldable formulae** left in **any** of the **tableaux** in one **system** (case), that system can be turned into a counter model.
  - When a **contradictory tableau** exists at **any world** in **every system**, we like this to mean that there is no countermodel. → See note on "Detecting Validity" later on.

The Rules

- There are 14 rules; 2 each for $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \Box$ and $\Diamond$
- The rules for $\neg, \vee, \wedge, \rightarrow$ and $\leftrightarrow$ are the same.
  - They work within a world:
    - the resulting formulae go into the same world
    - there are no changes in other worlds
- 4 new rules exist for $\Box$ true, $\Box$ false, $\Diamond$ true and $\Diamond$ false.

$\Diamond$ true

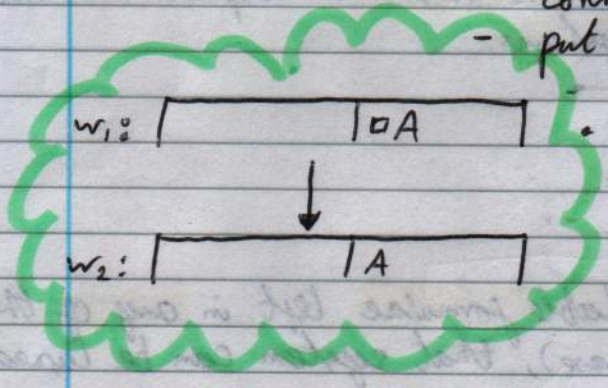
- For $\Diamond A$ true, we must:
  - create a new world
  - connect it to this one
  - put A in the new world true column



- This is because for $\Diamond A$ to be true there must be some future connected world where A is true.

$\Box$ false

- For $\Box A$ false, :
  - create a new world
  - connect it to this one
  - put A in the false column of the new world



- This is because for $\Box A$ to be false it must be connected to some future world where A is false.

□ true

"□A is true in w, if A is true in all v such that wRv"

- If no worlds are connected to w, do nothing.
- If any worlds are connected to w, place A in the true column of every world.

• This rule is "backtracking" - we must check □A true every time a rule might add a new connected world.

◇ false

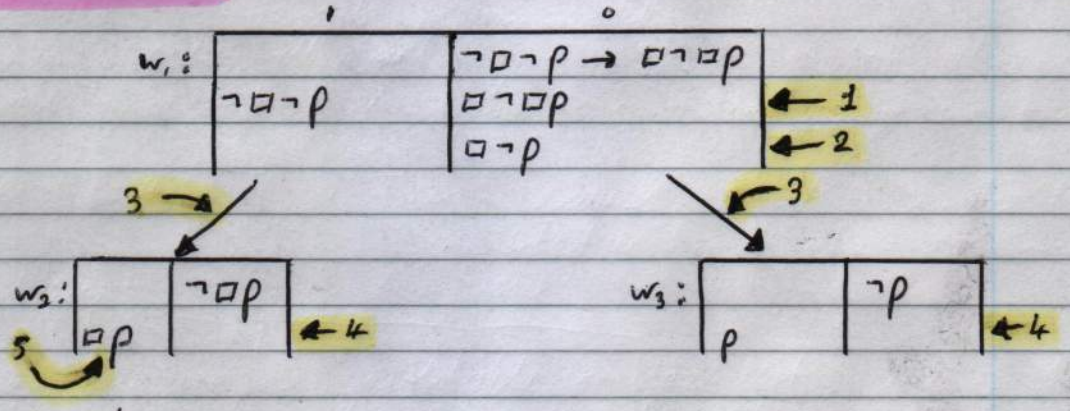
"◇A is false at w, if A is false in all v such that wRv"

For ◇A false in w:

- If no worlds are connected to w, do nothing.
- If any worlds are connected to w, place A in the false column of every world.

• This rule is also "backtracking".

Example Continued



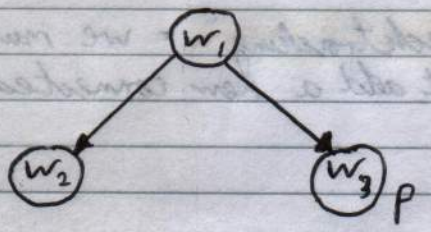
1. Apply → false.
2. Apply ¬ true.
3. Apply □ false twice
4. Apply ¬ false twice.
5. □ true gives nothing else.

6. We have found a countermodel, so we stop here.

Turning a Result into a Countermodel

- Each tableau is a world.
- Each world is labelled by the prop. variables that appeared in its true column.

From example:

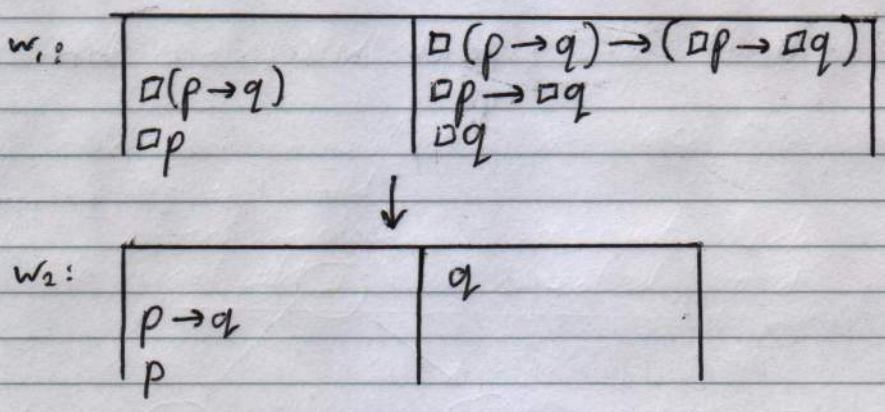


Detecting Validity

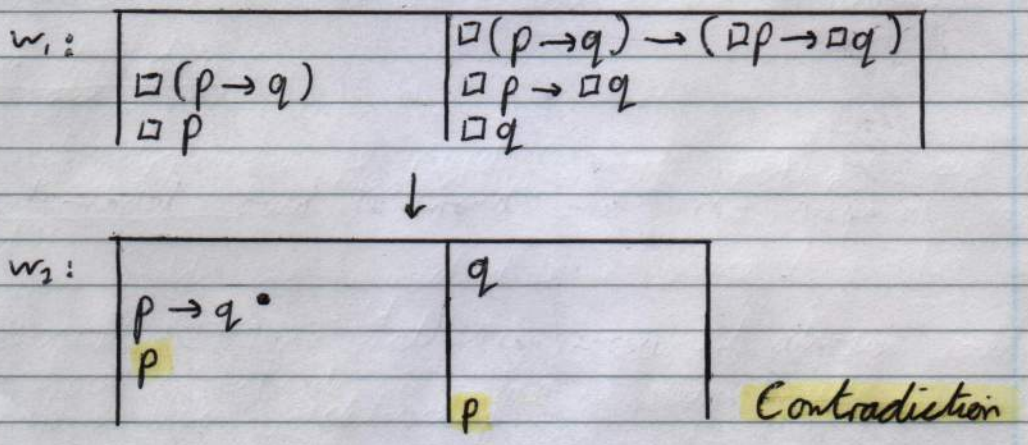
- If a contradiction can be found in every case, we say that no countermodel could be found and ∴ the original formula is valid.
- It is possible to prove that this argument always works, but such a proof is outside the scope of this course.

Another Example

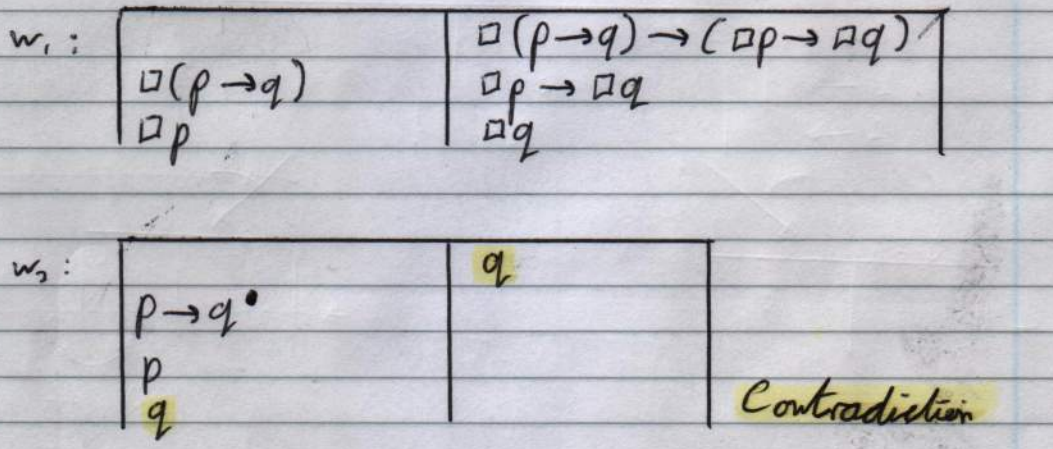
Is the formula $\Box(p \to q) \to (\Box p \to \Box q)$ valid in all frames?



Case 1:

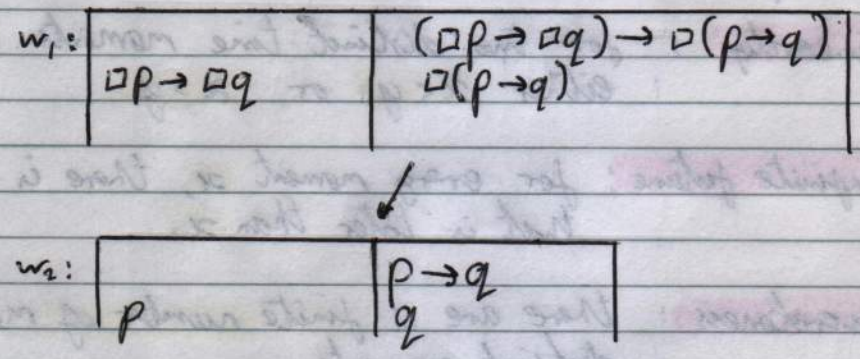


Case 2:

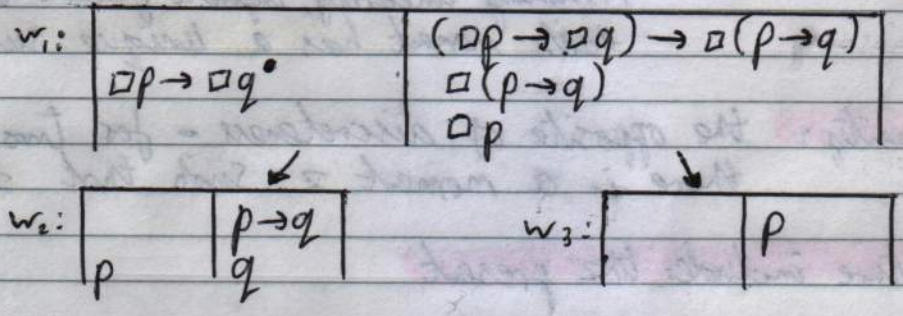


Yet Another Example

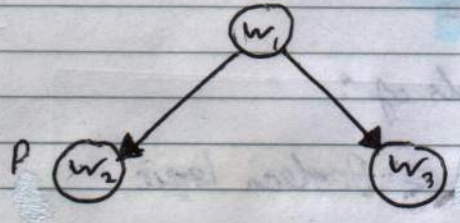
Is the formula $(\Box p \to \Box q) \to \Box(p \to q)$ valid in all frames?



Case 1:



This is a complete non-contradictory tableaux system, so we can turn it into a countermodel:



Temporal Logic

- Possible properties of temporal (time) moments:
  - **Linearity**: for two distinct time moments x and y, either $x < y$ or $x > y$
  - **Infinite future**: for every moment x, there is a moment y that is later than x.
  - **Discreteness**: there are a finite number of moments between two distinct moments.

    Assuming linearity, infinite future and discreteness, each moment has a unique successor moment.
  - **Density**: the opposite of discreteness - for two moments $x < y$, there is a moment z such that $x < z < y$.
  - Future includes the present
  - Future does not include the present.

Linear Temporal Logic

- The alphabet consists of:
  - all the symbols of Boolean logic
  - three new unary operators:

[F]	"always in the future"	} along with ¬, these bind more tightly.
⟨F⟩	"some time in the future"	
ⓣ	"tomorrow" or "next time"	

  - a new binary connector:

$\mathcal{U}$	"until"

LTL - Models

- LTL-models are just Kripke models based on the infinite frame:

- A model may be described:

$\mathcal{M}_2$: p is true at state s_{3k+2} for $k = 0, 1, 2, 3 \ldots$
q is true at state s_n for $n < 29$
r is true at every state.

- The $\mathcal{M} \models_{s_{12}} p$ notation carries forwards from Kripke Models.

Ⓣ
- $\mathcal{M} \models_{s_n} Ⓣ A$ iff $\mathcal{M} \models_{s_{n+1}} A$
  - Ⓣ behaves like $\Box$ or $\Diamond$.

[F]
- $\mathcal{M} \models_{s_n} [F]A$ iff $\mathcal{M} \models_{s_m} A$ for all $m \geq n$
  - $\mathcal{M} \not\models_{s_n} [F]A$ iff there is an $m \geq n$ such that $\mathcal{M} \not\models_{s_m} A$
  - $[F]A \rightarrow A$ is true at every state of every model

⟨F⟩
- $\mathcal{M} \models_{s_n} \langle F \rangle A$ iff there is an $m \geq n$ such that $\mathcal{M} \models_{s_m} A$
  - $A \rightarrow \langle F \rangle A$ for every state in every model
  - $Ⓣ A \rightarrow \langle F \rangle A$ for every state in every model

u
- $\mathcal{M} \models_{s_n} A \,u\, B$ iff a) there is $m \geq n$ such that $\mathcal{M} \models_{s_m} B$, and
  b) for all k with $n \leq k < m$, $\mathcal{M} \models_{s_k} A$
  - $A \,u\, B$ means "A will be true until B is true"
  - The following are true at every state of every model:
    - $B \rightarrow (A \,u\, B)$
    - $(A \,u\, B) \rightarrow \langle F \rangle B$
    - $(A \wedge Ⓣ B) \rightarrow (A \,u\, B)$

LTL Equivalences

- LTL formulae A and B are equivalent if $A \equiv B$ for every state in every model.

$- [F] \equiv \neg\langle F\rangle\neg A$

$- \langle F\rangle \equiv \neg\langle F\rangle\neg A$

$\neg \textcircled{\oplus} A \equiv \textcircled{\oplus}\neg A$

$[F](A \wedge B) \equiv [F]A \wedge [F]B$

$\langle F\rangle(A \vee B) \equiv \langle F\rangle A \vee \langle F\rangle B$

$[F]A \equiv A \wedge \textcircled{T}[F]A$

$\langle F\rangle A \equiv A \vee \textcircled{T}\langle F\rangle A$

$A \,u\, B \equiv B \vee (A \wedge \textcircled{T}(A \,u\, B))$

$A \,u\, B \equiv \langle F\rangle B \wedge \neg((\neg B)\, u\, (\neg B \wedge \neg A))$

LTL vs. Branching Time

- LTL considers only one linear "flow" or "path" of time.
- This is not adequate for modelling computational paths, as often there are many next states for any given state.
- Enter...

Computational Tree Logic (CTL)

- CTL is a type of branching time temporal logic.
- A CTL model is simply a Kripke model with no dead ends.
- Worlds in CTL models are thought to be time points or states in a computational system.
- The seriality condition (no dead ends) means that CTL models adopt the "future is infinite" model.

CTL Model Paths

- A path is an infinite sequence of subsequent states.
- Since there are no dead ends for each state s there is at least one path starting at s.
- A path can be understood as a possible "history" of events or a given computational path of a system.
- Every path in a CTL model works like an LTL model.

CTL Syntax

- The alphabet consists of:
  - The normal boolean connectives $\wedge \vee \rightarrow \leftrightarrow \neg$
  - The four symbols from LTL $[F] \langle F \rangle \bigcirc \mathcal{U}$
  - Two new unary symbols
    - $[path]$ "for all paths"
    - $\langle path \rangle$ "there is a path"
- Formulae are built inductively:
  - Any propositional variable is a CTL formula.
  - If A and B are CTL formulae, then so are...

$$\neg A \quad A \wedge B \quad A \vee B \quad A \rightarrow B \quad A \leftrightarrow B$$

$$[path][F]A \quad [path]\langle F \rangle A \quad [path]\bigcirc A \quad [path](A \,\mathcal{U}\, B)$$

$$\langle path \rangle [F]A \quad \langle path \rangle \langle F \rangle A \quad \langle path \rangle \bigcirc A \quad \langle path \rangle (A \,\mathcal{U}\, B)$$

  - Nothing else is a CTL formula.
- Note that this definition is more restrictive than other.
- $[path][F]$ $[path]\langle F \rangle$ $[path]\bigcirc$
  $\langle path \rangle [F]$ $\langle path \rangle \langle F \rangle$ $\langle path \rangle \bigcirc$
  } bind more tightly.

- Example:

Okay:	Not Okay:
$[path][F]p \rightarrow \langle path \rangle \textcircled{T} q$	$[path]p \rightarrow \textcircled{T} q$
$p \rightarrow (\langle path \rangle \langle F \rangle q \vee [path](p \, u \, r))$	$\textcircled{T}(p \rightarrow \langle path \rangle \textcircled{T} q)$
$\langle path \rangle [F][path]((\neg p) \, u \, r)$	$[F][path]p$
$\langle path \rangle ((q \rightarrow [path][F]q) \, u \, (\langle path \rangle \langle F \rangle q))$	$(q \rightarrow [path][F]q) \, u \, (\langle path \rangle \langle F \rangle q)$
	$(\langle path \rangle p) \, u \, q$

Mutual Exclusion in CTL

- Two processes share the same resource, R.
- The following prop. variables are chosen:

e1 / e2	→	Process 1 / 2 is doing something else
t1 / t2	→	" " " " trying to access R
a1 / a2	→	" " " " accessing R

- **Safety:** only one process may use R at any one time. There is no computational path where $a_1 \wedge a_2$ holds.

$\neg \langle path \rangle \langle F \rangle (a_1 \wedge a_2)$ or $[path][F] \neg (a_1 \wedge a_2)$

- **Non-blocking:** a process can always request access to R.

$[path][F](e_1 \rightarrow \langle path \rangle \textcircled{T} t_1)$

$[path][F](e_2 \rightarrow \langle path \rangle \textcircled{T} t_2)$

- **Liveness:** any process trying to access R will eventually be able to access it.

$[path][F](t_1 \rightarrow [path] \langle F \rangle a1)$

$[path][F](t_2 \rightarrow [path] \langle F \rangle a2)$

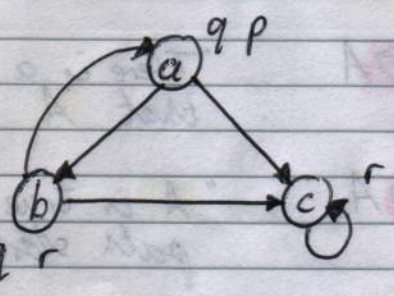
* Remember, future includes present.

Evaluating CTL Models.

- The $M \models_s A$ notation carries over from Kripke Models and LTL.
- $M \models_s \langle path \rangle \bigcirc A$ — "There is a path starting at s such that A is true at the next state."
- $M \models_s [path] \bigcirc A$ — "A is true at the next step of <u>every</u> path starting at s"
- $M \models_s \langle path \rangle \langle F \rangle A$ — "There is a path starting at s such that A is true in at least one of its future states" *
- $M \models_s [path] \langle F \rangle A$ — "For every path starting at s, there will be some future state on that path where A is true" *
- $M \models_s \langle path \rangle [F] A$ — "There is a path starting at s such that A is true at every future point on that path" *
- $M \models_s [path] [F] A$ — "For every path starting at s, every future state on that path has A true" *
- $M \models_s \langle path \rangle (A \mathcal{U} B)$ — "There is a path starting at s such that B is true at some future state, and until that state A is true at every state." *
- $M \models_s [path] (A \mathcal{U} B)$ — "For every path starting at s, B is true at some future state and A is true at every state until that future state" *

Basic CTL Examples

- Take this basic CTL model:



- $M \models_b \langle path \rangle \bigcirc (p \wedge q)$ because p and q are true at a.
- $M \not\models_b [path] \bigcirc (p \wedge q)$ consider the path $b \to c \to c \to c \to \dots$
- $M \not\models_a \langle path \rangle \langle F \rangle (p \wedge r)$ because there is no state in M where p and r hold.
- $M \models_a \langle path \rangle \bigcirc [path][F] r$ consider the path $a \to c \to c \to c \to c \to \dots$
- $M \models_a \langle path \rangle (q \, u \, (r \wedge q))$ consider the path $a \to b \to \dots$
- $M \not\models_b [path] (q \, u \, (r \wedge \neg q))$ consider the path $b \to a \to b \to a \to b \to \dots$ this path has no state where $(r \wedge \neg q)$ holds

Equivalencies in CTL

- In CTL, $A \equiv B$ if A and B have the same truth value for every state in every model.
- Every CTL formula can be transformed into an equivalent using only:

$\neg \quad \wedge \quad 1 \quad \langle path \rangle \bigcirc \quad [path] \langle F \rangle \quad \langle path \rangle u$

$A \vee B \equiv \neg(\neg A \wedge \neg B) \qquad A \to B \equiv \neg A \vee B \equiv \neg(A \wedge \neg B)$

$A \leftrightarrow B \equiv \neg(A \wedge \neg B) \wedge \neg(\neg A \wedge B)$

$[path] \bigcirc A = \neg \langle path \rangle \bigcirc \neg A \qquad \langle path \rangle [F] A \equiv \neg [path] \langle F \rangle \neg A$

$\langle path \rangle \langle F \rangle A \equiv \langle path \rangle (1 \, u \, A)$

$[path][F] A \equiv \neg \langle path \rangle \langle F \rangle \neg A \equiv \neg \langle path \rangle (1 \, u \, \neg A)$

$[path](A \, u \, B) \equiv [path] \langle B \rangle B \wedge \neg \langle path \rangle ((\neg B) \, u \, (\neg B \wedge \neg A))$

$\equiv \neg \langle path \rangle (1 \, u \, \neg B)) \wedge \neg \langle path \rangle ((\neg B) \, u \, (\neg B \wedge \neg A))$

The Labelling Algorithm for CTL

- Input: any finite CTL model M and any CTL formula A
  Output: the set of states in M where A is true.

- **Step 0: Pre-processing**

  - Transform A into a formula using only:

    $\neg$ $\wedge$ $\top$ $\langle path \rangle \bigcirc$ $[path]\langle F \rangle$ $\langle path \rangle U$

- **Step 1: Labelling**

  - The algo. labels each state s in M with the CTL-subformulae of A that are true in s.

  - The algo. goes through the subformulae "inside out", starting with the prop. variables.

  - We use a case analysis to decide labelling for each subformula, based on the outermost connective.

  - There are 7 cases.

- **Case 1**: SF that are propositional variables.

  - No labelling to do - these labels already exist in the model.

- **Case 2**: SF in the form $\neg B$

  - For each state s, label s with $\neg B$ if it is not labelled with B

- **Case 3**: SF in the form $B \wedge C$

  - For each state s, label s with $B \wedge C$ if it is labelled with B and C.

- **Case 4**: SF in the form $\top$

  - Label every state with $\top$.

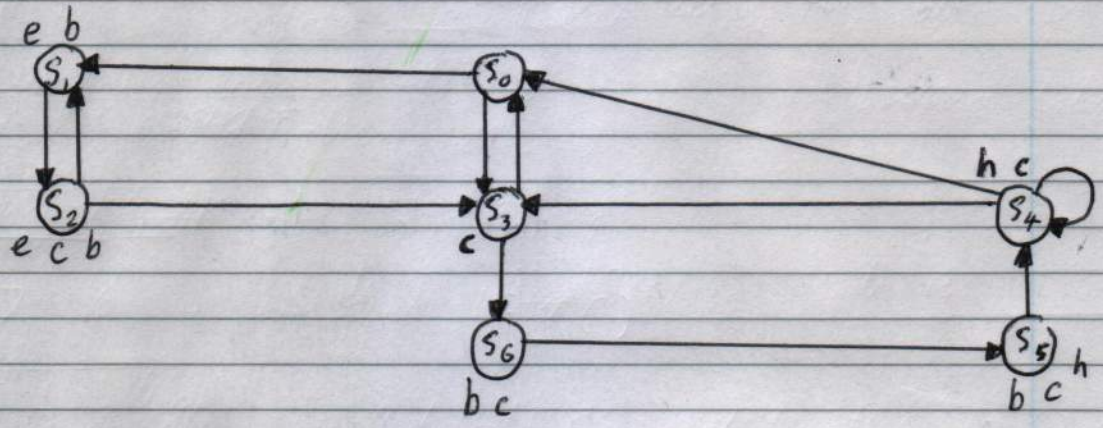
- **Case 5**: SF in the form $\langle path \rangle \bigcirc B$

  - For each state s, label s with $\langle path \rangle \bigcirc B$ if some $\rightarrow$-successor state is labelled with B.

- **Case 6:** SF in the form [path]⟨F⟩B
  - If any state is labelled with B, label it with [path]⟨F⟩B
  - Label any state s with [path]⟨F⟩B if all successor states of s are labelled with [path]⟨F⟩B.
  - Repeat until there is no change.
- **Case 7:** SF in the form ⟨path⟩(B U C)
  - If any state is labelled with C, label it with ⟨path⟩(B U C)
  - Label any state s with ⟨path⟩(B U C) if s is labelled with B and some successor state of s is labelled with ⟨path⟩(B U C)
  - Repeat until there is no change.
- **Properties of the Algorithm.**
  - The algorithm always gives the correct answer.
  - The algorithm runs in linear time, relative to both the size of M and the size of A.
  - Unfortunately, the size of M can be exponential to the size of A.
    - This creates the state explosion problem.
  - ∴ it is important to represent CTL models in efficient ways.
    - This can be done with the help of OBDDs...

CTL Labelling Algorithm Example

M:



A: $[path][F](b \rightarrow [path]\langle F\rangle h)$

$\equiv [path][F]\neg(b \wedge \neg[path]\langle F\rangle h)$

$\equiv \neg\langle path\rangle\langle F\rangle(b \wedge \neg[path]\langle F\rangle h)$

$\equiv \neg\langle path\rangle(1\ \mathcal{U}\ (b \wedge \neg[path]\langle F\rangle h))$

Sub-formulae: b, h, 1, $[path]\langle F\rangle h$, $\neg[path]\langle F\rangle h$, $b \wedge \neg[path]\langle F\rangle h$,

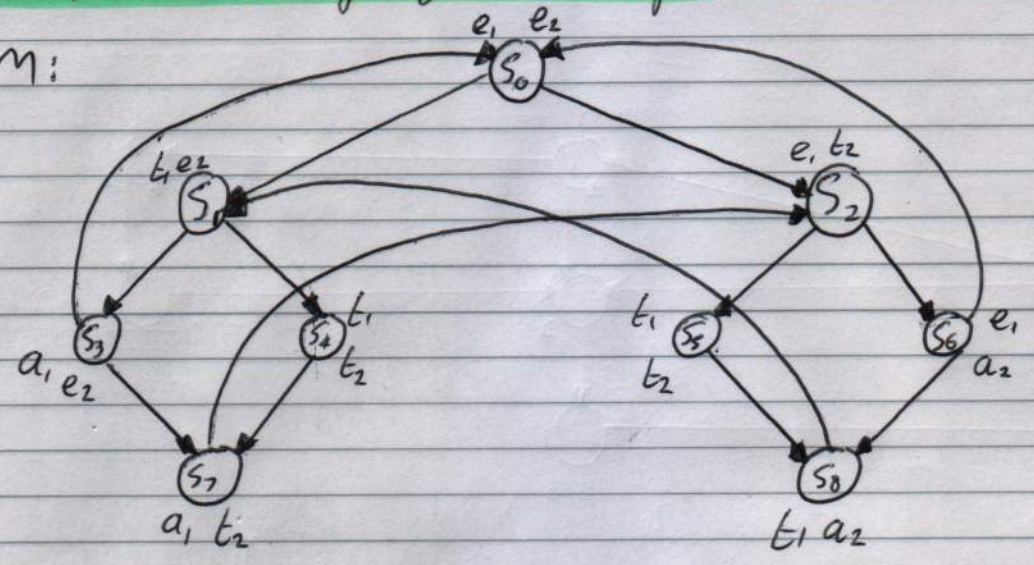
$\langle path\rangle(1\ \mathcal{U}\ (b \wedge \neg[path]\langle F\rangle h))$, $\neg\langle path\rangle(1\ \mathcal{U}\ (b \wedge \neg[path]\langle F\rangle h))$

- States l/w b: s_1, s_2, s_5, s_6
- States l/w h: s_4, s_5
- States l/w 1: $s_0, s_1, s_2, s_3, s_4, s_5, s_6$
- States l/w $[path]\langle F\rangle h$: first: s_4, s_5
  2nd: s_6
  overall: s_4, s_5, s_6
- States l/w $\neg[path]\langle F\rangle h$: s_0, s_1, s_2, s_3,
- States l/w $\langle path\rangle(1\ \mathcal{U}\ (b \wedge \neg[path]\langle F\rangle h))$: s_0, s_1, s_2, s_3, s_4, s_5, s_6 } Error around this step.
- States l/w $\neg\langle path\rangle(1\ \mathcal{U}\ (b \wedge \neg[path]\langle F\rangle h))$: none.

The formula A is not true in any states in the model M.

Another CTL Labelling Algorithm Example

M:



$A: \langle path \rangle \textcircled{T} \left(t_1 \wedge \langle path \rangle ((\neg a_2) \, u \, a_1) \right)$

Sub-formulae: t_1, a_1, a_2, $\neg a_2$, $\langle path \rangle ((\neg a_2) \, u \, a_1)$,

$t_1 \wedge \langle path \rangle ((\neg a_2) \, u \, a_1)$, $\langle path \rangle \textcircled{T} \left(t_1 \wedge \langle path \rangle ((\neg a_2) \, u \, a_1) \right)$

- States l/w t_1: s_1, s_4, s_5, s_8
- States l/w a_1: s_3, s_7
- States l/w a_2: s_6, s_8
- States l/w $\neg a_2$: $s_0, s_1, s_2, s_3, s_4, s_5, s_7$
- States l/w $\langle path \rangle ((\neg a_2) \, u \, a_1)$: s_3, s_7
  s_1, s_4
  s_0
  overall: s_0, s_1, s_3, s_4, s_7
- States l/w $t_1 \wedge \langle path \rangle ((\neg a_2) \, u \, a_1)$: s_1, s_4
- States l/w $\langle path \rangle \textcircled{T} \left(t_1 \wedge \langle path \rangle ((\neg a_2) \, u \, a_1) \right)$: s_0, s_1, s_8

$M \models_{s_0, s_1, s_8} A$