7. Midterm Review!

7. Midterm Review!

a. Recursive Definitions

Recursive Definitions have THREE clauses

- Consider an arbitrary recursively-defined set $\mathcal{S}$

Recursive Definitions have THREE clauses

- Consider an arbitrary recursively-defined set $\mathcal{S}$

  1. **Base clause**: typically the smallest element(s) of your set, e.g. the atomic sentences

Recursive Definitions have THREE clauses

- Consider an arbitrary recursively-defined set $\mathcal{S}$

1. **Base clause**: typically the smallest element(s) of your set, e.g. the atomic sentences

2. **Recursion clause**: If s is an element in $\mathcal{S}$, then so is s acted on by some basic operation(s) resulting in
   elements of *the next biggest* ***allowed*** *size*.

Recursive Definitions have THREE clauses

- Consider an arbitrary recursively-defined set $\mathcal{S}$

1. **Base clause**: typically the smallest element(s) of your set, e.g. the atomic sentences

2. **Recursion clause**: If s is an element in $\mathcal{S}$, then so is s acted on by some basic operation(s) resulting in elements of *the next biggest* ***allowed*** *size*.

3. **Closure clause**: Nothing else is an element of $\mathcal{S}$

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards e.g. “racecar”.

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards e.g. “racecar”.
- some trivial cases: “I”, “a” (we’re thinkin ‘base case material!’)

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards e.g. “racecar”.
- some trivial cases: “I”, “a” (we’re thinkin ‘base case material!’)
- Consider the set of all strings in the alphabet $\{a, b, c\}$

Recursively Defining Palindromes

- Define the set of palindromes using a recursive definition:

Recursively Defining Palindromes

- Define the set of palindromes using a recursive definition:
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:

Recursively Defining Palindromes

- Define the set of palindromes using a recursive definition:
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:
  1. **Base clause**: 'a', 'b', 'c', 'aa', 'bb', 'cc' are recursive palindromes

Recursively Defining Palindromes

- Define the set of palindromes using a recursive definition:
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:
  1. **Base clause**: 'a', 'b', 'c', 'aa', 'bb', 'cc' are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, $b * s * b$, and $c * s * c$ are recursive palindromes

Recursively Defining Palindromes

- Define the set of palindromes using a recursive definition:
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:
  1. **Base clause**: 'a', 'b', 'c', 'aa', 'bb', 'cc' are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, $b * s * b$, and $c * s * c$ are recursive palindromes
  3. **Closure clause**: Nothing else is a recursive palindrome in $\{a, b, c\}$

What happens if we place restrictions on palindromes?

- Consider the alphabet $\{i, e, t\}$

What happens if we place restrictions on palindromes?

- Consider the alphabet $\{i, e, t\}$
- Informally define the "*tieeit*–palindromes" as all those palindromes with '*tieeit*' in the middle, with string–length divisible by 6, starting with 't', and with 'e' as the third letter from the left. Recursively define these bad boys:

What happens if we place restrictions on palindromes?

- Consider the alphabet $\{i, e, t\}$
- Informally define the "*tieeit*–palindromes" as all those palindromes with '*tieeit*' in the middle, with string–length divisible by 6, starting with 't', and with 'e' as the third letter from the left. Recursively define these bad boys:
  1. **Base clause**: '*tieeit*' is a *tieeit*–palindrome

What happens if we place restrictions on palindromes?

- Consider the alphabet $\{i, e, t\}$
- Informally define the "$tieeit$–palindromes" as all those palindromes with '$tieeit$' in the middle, with string–length divisible by 6, starting with 't', and with 'e' as the third letter from the left. Recursively define these bad boys:
  1. **Base clause**: '$tieeit$' is a $tieeit$–palindrome
  2. **Recursion clause**: If s is a $tieeit$–palindrome, then $tie * s * eit$, $tee * s * tee$, and $tte * s * ett$ are $tieeit$–palindromes.

What happens if we place restrictions on palindromes?

- Consider the alphabet $\{i, e, t\}$
- Informally define the "*tieeit*–palindromes" as all those palindromes with '*tieeit*' in the middle, with string–length divisible by 6, starting with 't', and with 'e' as the third letter from the left. Recursively define these bad boys:
  1. **Base clause**: '*tieeit*' is a *tieeit*–palindrome
  2. **Recursion clause**: If s is a *tieeit*–palindrome, then $tie * s * eit$, $tee * s * tee$, and $tte * s * ett$ are *tieeit*–palindromes.
  3. **Closure clause**: Nothing else is a *tieeit*–palindrome

7. Midterm Review!

b. Induction on Strings

Complete Induction (a.k.a. ‘strong induction’)

- We do induction on string length, considering members from a recursively defined set (e.g. palindromes over some alphabet)

Complete Induction (a.k.a. ‘strong induction’)

- We do induction on string length, considering members from a recursively defined set (e.g. palindromes over some alphabet)
  Base Case: Prove the property holds in the base case(s) (of the recursive definition—elements of shortest string length(s))

Complete Induction (a.k.a. ‘strong induction’)

- We do induction on string length, considering members from a recursively defined set (e.g. palindromes over some alphabet)
  Base Case: Prove the property holds in the base case(s) (of the recursive definition—elements of shortest string length(s))
  Induction Hypothesis: Assume that the property holds for members of length n, where base-case-index $\leq n < k$.

Complete Induction (a.k.a. 'strong induction')

- We do induction on string length, considering members from a recursively defined set (e.g. palindromes over some alphabet)
  Base Case: Prove the property holds in the base case(s) (of the recursive definition—elements of shortest string length(s))
  Induction Hypothesis: Assume that the property holds for members of length n, where base-case-index $\leq n < k$.
  Induction Step: Consider an arbitrary member Δ of length $k >$base case length(s). Show that Δ has the property of interest.

Complete Induction (a.k.a. 'strong induction')

- We do induction on string length, considering members from a recursively defined set (e.g. palindromes over some alphabet)
  Base Case: Prove the property holds in the base case(s) (of the recursive definition—elements of shortest string length(s))
  Induction Hypothesis: Assume that the property holds for members of length n, where base-case-index $\leq n < k$.
  Induction Step: Consider an arbitrary member Δ of length $k >$base case length(s). Show that Δ has the property of interest.
- Note that there must be a smaller string δ such that $\Delta = blah * \delta * blah'$, where the 'blah's arise from the recursion clause. (all the ways of making a longer Δ from a shorter δ)

Complete Induction (a.k.a. 'strong induction')

- We do induction on string length, considering members from a recursively defined set (e.g. palindromes over some alphabet)
  Base Case: Prove the property holds in the base case(s) (of the recursive definition—elements of shortest string length(s))
  Induction Hypothesis: Assume that the property holds for members of length n, where base-case-index $\leq n < k$.
  Induction Step: Consider an arbitrary member Δ of length $k >$base case length(s). Show that Δ has the property of interest.
- Note that there must be a smaller string δ such that $\Delta = blah * \delta * blah'$, where the 'blah's arise from the recursion clause. (all the ways of making a longer Δ from a shorter δ)
- Remember to note that since δ has length $< k$, it falls within the induction hypothesis and hence has the property of interest.

Example: Prove by induction that...

- Show: any *tieeit*−palindrome has string length divisible by 6.

Example: Prove by induction that...

- Show: any *tieeit*−palindrome has string length divisible by 6.
  Base Case: the base case has a single element, namely '*tieeit*' of string length 6, which is clearly divisible by 6.

Example: Prove by induction that...

- Show: any *tieeit*−palindrome has string length divisible by 6.
  Base Case: the base case has a single element, namely '*tieeit*' of string length 6, which is clearly divisible by 6.
  Induction Hypothesis: Assume that every *tieeit*−palindrome of length n, where $6 \leq n < k$, has the property, i.e. is divisible by 6.
  Show that the property holds for any *tieeit*−palindrome of length $k > 6$.

Example: Prove by induction that...

- Show: any *tieeit*−palindrome has string length divisible by 6.
  Base Case: the base case has a single element, namely '*tieeit*' of string length 6, which is clearly divisible by 6.
  Induction Hypothesis: Assume that every *tieeit*−palindrome of length n, where $6 \leq n < k$, has the property, i.e. is divisible by 6. Show that the property holds for any *tieeit*−palindrome of length $k > 6$.
  Induction Step: consider an arbitrary *tieeit*−palindrome Δ of length $k > 6$. Then there must exist a shorter *tieeit*−palindrome δ such that Δ equals either $tie * \delta * eit$, $tee * \delta * tee$, or $tte * \delta * ett$. Since δ has length $< k$, the string length of δ is divisible by 6 by the induction hypothesis. In each case, we add 6 letters to δ to form Δ. Hence, the string length of Δ is also divisible by 6.

7. Midterm Review!

c. Induction on SL Sentences

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If Φ and Ψ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective

     '**Induction step**' (There are five cases to consider:)

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property (‘**base case**’)
  2. If Φ and Ψ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
     ‘**Induction step**’ (There are five cases to consider:)
     (i) $\sim\Phi$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If Φ and Ψ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective

     '**Induction step**' (There are five cases to consider:)

     (i) $\sim\Phi$

     (ii) $(\Phi \,\&\, \Psi)$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If Φ and Ψ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
     '**Induction step**' (There are five cases to consider:)
     (i) $\sim\Phi$
     (ii) $(\Phi \& \Psi)$
     (iii) $(\Phi \vee \Psi)$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If Φ and Ψ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
     '**Induction step**' (There are five cases to consider:)
     (i) $\sim\Phi$
     (ii) $(\Phi \,\&\, \Psi)$
     (iii) $(\Phi \vee \Psi)$
     (iv) $(\Phi \supset \Psi)$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If Φ and Ψ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
     '**Induction step**' (There are five cases to consider:)
     (i) $\sim\Phi$
     (ii) $(\Phi \,\&\, \Psi)$
     (iii) $(\Phi \vee \Psi)$
     (iv) $(\Phi \supset \Psi)$
     (v) $(\Phi \equiv \Psi)$

Example: Induction on SL, using disjunctive syllogism

Prove the following by induction on wffs of SL. Don’t forget to explicitly state the **base case** and the **induction step**!

1. If a wff doesn’t contain any binary connectives, then it is contingent.

Example: Induction on SL, using disjunctive syllogism

Prove the following by induction on wffs of SL. Don't forget to explicitly state the **base case** and the **induction step**!

1. If a wff doesn't contain any binary connectives, then it is contingent.
2. hint: say that a wff is *baller* if it either contains a binary connective or is contingent.
   - Use induction to show that every wff is baller.

Example: Induction on SL, using disjunctive syllogism

Prove the following by induction on wffs of SL. Don't forget to explicitly state the **base case** and the **induction step**!

1. If a wff doesn't contain any binary connectives, then it is contingent.
2. hint: say that a wff is *baller* if it either contains a binary connective or is contingent.
   - Use induction to show that every wff is baller.
3. Final step (disjunctive syllogism): if (i) every wff is baller and (ii) a given wff doesn't contain any binary connectives, then (iii) it must be baller in virtue of being contingent.

Baller Induction: Base Case

Base case: this comprises the atomic sentences. So consider an arbitrary atomic sentence.

Baller Induction: Base Case

Base case: this comprises the atomic sentences. So consider an arbitrary atomic sentence.

- Note that it is contingent: there exists a TVA where it is true, and a distinct TVA where it is false.

Baller Induction: Base Case

Base case: this comprises the atomic sentences. So consider an arbitrary atomic sentence.

- Note that it is contingent: there exists a TVA where it is true, and a distinct TVA where it is false.
- Hence, every member in the base case is baller

Baller Induction: Induction Step

Induction Step (using our 'lazy SL' schema): consider two arbitrary well-formed formula Φ and Ψ that have the property, i.e. are baller. Show that any way of forming a more complex wff from these two also has the property.

Induction Step (using our 'lazy SL' schema): consider two arbitrary well-formed formula Φ and Ψ that have the property, i.e. are baller. Show that any way of forming a more complex wff from these two also has the property.

i. (a) If Φ is contingent, then so is its negation (its negation is true whenever Φ is false and false whenever Φ is true).
   (b) If Φ has a binary connective, then so does its negation $\sim\Phi$.

Baller Induction: Induction Step

Induction Step (using our ‘lazy SL’ schema): consider two arbitrary well-formed formula Φ and Ψ that have the property, i.e. are baller. Show that any way of forming a more complex wff from these two also has the property.

i. (a) If Φ is contingent, then so is its negation (its negation is true whenever Φ is false and false whenever Φ is true).
(b) If Φ has a binary connective, then so does its negation $\sim\Phi$.

ii.-v. Then, consider the four cases coming from connecting Φ and Ψ with a binary connective (& , $\vee$, $\supset$, or $\equiv$). Clearly, each of these wffs has a binary connective and so is baller.

Baller Induction: Induction Step

Induction Step (using our 'lazy SL' schema): consider two arbitrary well-formed formula Φ and Ψ that have the property, i.e. are baller. Show that any way of forming a more complex wff from these two also has the property.

i. (a) If Φ is contingent, then so is its negation (its negation is true whenever Φ is false and false whenever Φ is true).
   (b) If Φ has a binary connective, then so does its negation $\sim\Phi$.

ii.-v. Then, consider the four cases coming from connecting Φ and Ψ with a binary connective (& , $\vee$, $\supset$, or $\equiv$). Clearly, each of these wffs has a binary connective and so is baller.

- Hence, every wff is baller.

Baller Induction: Induction Step

Induction Step (using our ‘lazy SL’ schema): consider two arbitrary well-formed formula Φ and Ψ that have the property, i.e. are baller. Show that any way of forming a more complex wff from these two also has the property.

i. (a) If Φ is contingent, then so is its negation (its negation is true whenever Φ is false and false whenever Φ is true).
(b) If Φ has a binary connective, then so does its negation $\sim\Phi$.

ii.-v. Then, consider the four cases coming from connecting Φ and Ψ with a binary connective (& , $\vee$, $\supset$, or $\equiv$). Clearly, each of these wffs has a binary connective and so is baller.

- Hence, every wff is baller.

- Hence, if a well-formed formula does not contain any binary connectives, since it is still baller, it must be contingent (disjunctive syllogism)

Non-lazy Induction Schema for SL

Induction Hypothesis: assume that every SL wff of string-length n, where $1 \leq n < k$ has the property, i.e. is baller. Show that an arbitrary wff of length k is baller.

Non-lazy Induction Schema for SL

Induction Hypothesis: assume that every SL wff of string-length n, where $1 \leq n < k$ has the property, i.e. is baller. Show that an arbitrary wff of length k is baller.

Induction Step: Consider an arbitrary SL wff Δ of length k. Then by the recursive definition of SL wffs, there must exist shorter wffs Φ and Ψ such that Δ equals either (i) $\sim\Phi$, (ii) $\Phi \,\&\, \Psi$, (iii) $\Phi \vee \Psi$, (iv) $\Phi \supset \Psi$, or (v) $\Phi \equiv \Psi$.

Non-lazy Induction Schema for SL

Induction Hypothesis: assume that every SL wff of string-length n, where $1 \leq n < k$ has the property, i.e. is baller. Show that an arbitrary wff of length k is baller.

Induction Step: Consider an arbitrary SL wff Δ of length k. Then by the recursive definition of SL wffs, there must exist shorter wffs Φ and Ψ such that Δ equals either (i) $\sim\Phi$, (ii) $\Phi \,\&\, \Psi$, (iii) $\Phi \vee \Psi$, (iv) $\Phi \supset \Psi$, or (v) $\Phi \equiv \Psi$.

- Proceed to show that in each case, Δ has the property, i.e. is baller.

Key Fact about Conditionals

- Consider a conditional $P \supset Q$. If a truth value assignment satisfies the consequent Q, then it satisfies the conditional (regardless of the truth value of P)

7. Midterm Review!

d. Trees

Tree-contradictions and Tree-tautologies

Tree-contradiction

- A sentence Φ is a tree-contradiction if there is a tree that starts with Φ that has only closed branches

Tree-contradictions and Tree-tautologies

Tree-contradiction

- A sentence Φ is a tree-contradiction if there is a tree that starts with Φ that has only closed branches
- (definition only requires the existence of a single such tree)

Tree-contradictions and Tree-tautologies

Tree-contradiction

- A sentence Φ is a tree-contradiction if there is a tree that starts with Φ that has only closed branches
- (definition only requires the existence of a single such tree)
- (it says nothing about *all* trees starting with Φ)

Tree-contradictions and Tree-tautologies

Tree-contradiction

- A sentence Φ is a tree-contradiction if there is a tree that starts with Φ that has only closed branches
- (definition only requires the existence of a single such tree)
- (it says nothing about *all* trees starting with Φ)

Tree-tautology

- A sentence Ψ is a tree-tautology if there is a tree that starts with $\sim\Psi$ that has only closed branches

Tree-contradictions and Tree-tautologies

Tree-contradiction

- A sentence Φ is a tree-contradiction if there is a tree that starts with Φ that has only closed branches
- (definition only requires the existence of a single such tree)
- (it says nothing about *all* trees starting with Φ)

Tree-tautology

- A sentence Ψ is a tree-tautology if there is a tree that starts with $\sim\Psi$ that has only closed branches
- i.e., provided that $\sim\Psi$ is a tree-contradiction

Tree-contradictions and Tree-tautologies

Tree-contradiction

- A sentence Φ is a tree-contradiction if there is a tree that starts with Φ that has only closed branches
- (definition only requires the existence of a single such tree)
- (it says nothing about *all* trees starting with Φ)

Tree-tautology

- A sentence Ψ is a tree-tautology if there is a tree that starts with $\sim\Psi$ that has only closed branches
- i.e., provided that $\sim\Psi$ is a tree-contradiction
- In this case, we write '$\vdash_{STD} \Psi$'

Tree-contradictions and Tree-tautologies

Tree-contradiction

- A sentence Φ is a tree-contradiction if there is a tree that starts with Φ that has only closed branches
- (definition only requires the existence of a single such tree)
- (it says nothing about *all* trees starting with Φ)

Tree-tautology

- A sentence Ψ is a tree-tautology if there is a tree that starts with $\sim\Psi$ that has only closed branches
- i.e., provided that $\sim\Psi$ is a tree-contradiction
- In this case, we write '$\vdash_{STD} \Psi$'
- (again: this definition requires the existence of single such tree)

Tree-valid vs. Tree-invalid

- Consider an argument with premises given by a set Γ of wffs and conclusion Φ (i.e. Γ could be multiple sentences)

Tree-valid vs. Tree-invalid

- Consider an argument with premises given by a set Γ of wffs and conclusion Φ (i.e. Γ could be multiple sentences)
- Construct a tree with the following root: all sentences in Γ along with $\sim\Phi$ (i.e. the NEGATION of the conclusion)

Tree-valid vs. Tree-invalid

- Consider an argument with premises given by a set Γ of wffs and conclusion Φ (i.e. Γ could be multiple sentences)
- Construct a tree with the following root: all sentences in Γ along with $\sim\Phi$ (i.e. the NEGATION of the conclusion)
- Next, apply the tree-rules until either

Tree-valid vs. Tree-invalid

- Consider an argument with premises given by a set Γ of wffs and conclusion Φ (i.e. Γ could be multiple sentences)
- Construct a tree with the following root: all sentences in Γ along with $\sim\Phi$ (i.e. the NEGATION of the conclusion)
- Next, apply the tree-rules until either

1.) **Each branch closes**, in which case the argument is **tree-valid**

  - In this case, we write $\Gamma \vdash_{STD} \Phi$

Tree-valid vs. Tree-invalid

- Consider an argument with premises given by a set Γ of wffs and conclusion Φ (i.e. Γ could be multiple sentences)
- Construct a tree with the following root: all sentences in Γ along with $\sim\Phi$ (i.e. the NEGATION of the conclusion)
- Next, apply the tree-rules until either

1.) **Each branch closes**, in which case the argument is **tree-valid**

- In this case, we write $\Gamma \vdash_{STD} \Phi$

2.) You have a complete open branch, in which case the argument is **tree-invalid**

Using trees to check for Validity

Since most homework problems follow this pattern, let's make it really explicit!

1. Add each premise to the root (number each line)

Don't forget to **justify each new node** by citing the line you are resolving and the rule you are applying

Using trees to check for Validity

Since most homework problems follow this pattern, let's make it really explicit!

1. Add each premise to the root (number each line)
2. Add the **NEGATION** of the conclusion to the root

Don't forget to **justify each new node** by citing the line you are resolving and the rule you are applying

Using trees to check for Validity

Since most homework problems follow this pattern, let's make it really explicit!

1. Add each premise to the root (number each line)
2. Add the **NEGATION** of the conclusion to the root
3. Resolve sentences until either:

Don't forget to **justify each new node** by citing the line you are resolving and the rule you are applying

Using trees to check for Validity

Since most homework problems follow this pattern, let's make it really explicit!

1. Add each premise to the root (number each line)
2. Add the **NEGATION** of the conclusion to the root
3. Resolve sentences until either:
    - **Each branch closes**, in which case the argument is **valid**

Don't forget to **justify each new node** by citing the line you are resolving and the rule you are applying

Using trees to check for Validity

Since most homework problems follow this pattern, let's make it really explicit!

1. Add each premise to the root (number each line)
2. Add the **NEGATION** of the conclusion to the root
3. Resolve sentences until either:
    - **Each branch closes**, in which case the argument is **valid**
    - You have a complete open branch $\Rightarrow$ the argument is **invalid**

Don't forget to **justify each new node** by citing the line you are resolving and the rule you are applying

Using trees to check for Validity

Since most homework problems follow this pattern, let's make it really explicit!

1. Add each premise to the root (number each line)
2. Add the **NEGATION** of the conclusion to the root
3. Resolve sentences until either:
    - **Each branch closes**, in which case the argument is **valid**
    - You have a complete open branch $\Rightarrow$ the argument is **invalid**

Don't forget to **justify each new node** by citing the line you are resolving and the rule you are applying

Using trees to check for Validity

Since most homework problems follow this pattern, let's make it really explicit!

1. Add each premise to the root (number each line)
2. Add the **NEGATION** of the conclusion to the root
3. Resolve sentences until either:
    - **Each branch closes**, in which case the argument is **valid**
    - You have a complete open branch $\Rightarrow$ the argument is **invalid**

Don't forget to **justify each new node** by citing the line you are resolving and the rule you are applying

Remember that a branch closes whenever a sentence and its negation appear in its nodes (these need not be atomic sentences)

Using trees to check Tautologies

Likewise for whether a sentence is a tautology:

1. Add the **NEGATION** of the sentence to the root

Using trees to check Tautologies

Likewise for whether a sentence is a tautology:

1. Add the **NEGATION** of the sentence to the root
2. Resolve sentences until either:

Using trees to check Tautologies

Likewise for whether a sentence is a tautology:

1. Add the **NEGATION** of the sentence to the root
2. Resolve sentences until either:
    - **Each branch closes**, in which case the sentence is **tautologous** (semantic aside: it's impossible to make the sentence false)

Using trees to check Tautologies

Likewise for whether a sentence is a tautology:

1. Add the **NEGATION** of the sentence to the root
2. Resolve sentences until either:
   - **Each branch closes**, in which case the sentence is **tautologous** (semantic aside: it's impossible to make the sentence false)
   - You have a complete open branch, in which case the sentence is NOT a tautology
     (semantic aside: it is possible to satisfy the sentence's negation, so it's possible to make the sentence in question false)

Trees: easy things to forget

- Remember to number each line, i.e. each horizontal row of sentences gets its own line number

Trees: easy things to forget

- Remember to number each line, i.e. each horizontal row of sentences gets its own line number
- Remember to justify each child node by citing the line number of the sentence you are resolving and listing the rule you are using

Trees: easy things to forget

- Remember to number each line, i.e. each horizontal row of sentences gets its own line number
- Remember to justify each child node by citing the line number of the sentence you are resolving and listing the rule you are using
- Remember to justify any closed branches ($\times$) by citing the line numbers of a sentence and its negation on that branch

Trees: easy things to forget

- Remember to number each line, i.e. each horizontal row of sentences gets its own line number
- Remember to justify each child node by citing the line number of the sentence you are resolving and listing the rule you are using
- Remember to justify any closed branches ($\times$) by citing the line numbers of a sentence and its negation on that branch
- Don't apply any semantic equivalencies! e.g. to resolve $\sim(\sim D \vee E)$ you write $\sim\sim D$ stacked on $\sim E$. You cannot immediately write 'D'. To get D, you would have to apply the double negation rule to $\sim\sim D$

7. Midterm Review!

e. Trees Metalogic: Testing Alternative Rules

Soundness vs. completeness

- Both soundness and completeness are if–then statements (i.e. of the form $P \supset Q$):

Soundness vs. completeness

- Both soundness and completeness are if–then statements (i.e. of the form $P \supset Q$):
  - **Sound**: If $\Gamma \vdash_{STD} \Theta$, then $\Gamma \vDash \Theta$

Soundness vs. completeness

- Both soundness and completeness are if-then statements (i.e. of the form $P \supset Q$):
  - **Sound**: If $\Gamma \vdash_{STD} \Theta$, then $\Gamma \vDash \Theta$
  - **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD} \Theta$

Soundness vs. completeness

- Both soundness and completeness are if–then statements (i.e. of the form $P \supset Q$):
  - **Sound**: If $\Gamma \vdash_{STD} \Theta$, then $\Gamma \vDash \Theta$
  - **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD} \Theta$
- Hence, they are logically equivalent to their contrapositives:

Soundness vs. completeness

- Both soundness and completeness are if–then statements (i.e. of the form $P \supset Q$):
  - **Sound**: If $\Gamma \vdash_{STD} \Theta$, then $\Gamma \vDash \Theta$
  - **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD} \Theta$
- Hence, they are logically equivalent to their contrapositives:
  - Contrapositive of $(P \supset Q)$ is $({\sim}Q \supset {\sim}P)$

Soundness vs. completeness

- Both soundness and completeness are if–then statements (i.e. of the form $P \supset Q$):
  - **Sound**: If $\Gamma \vdash_{STD} \Theta$, then $\Gamma \vDash \Theta$
  - **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD} \Theta$
- Hence, they are logically equivalent to their contrapositives:
  - Contrapositive of $(P \supset Q)$ is $({\sim}Q \supset {\sim}P)$
  - **Soundness**: If $\Gamma \nvDash \Theta$, then $\Gamma \nvdash_{STD} \Theta$

Soundness vs. completeness

- Both soundness and completeness are if–then statements (i.e. of the form $P \supset Q$):
  - **Sound**: If $\Gamma \vdash_{STD} \Theta$, then $\Gamma \vDash \Theta$
  - **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD} \Theta$
- Hence, they are logically equivalent to their contrapositives:
  - Contrapositive of $(P \supset Q)$ is $(\sim Q \supset \sim P)$
  - **Soundness**: If $\Gamma \nvDash \Theta$, then $\Gamma \nvdash_{STD} \Theta$
  - **Completeness**: If $\Gamma \nvdash_{STD} \Theta$, then $\Gamma \nvDash \Theta$

Some Definitions

- "$\Gamma \nvdash_{STD} \Theta$" means that *it is not the case that* the argument from Γ to Θ is **tree-valid**

Some Definitions

- "$\Gamma \nvdash_{STD} \Theta$" means that *it is not the case that* the argument from Γ to Θ is **tree-valid**
- i.e., *it is not the case that* there exists a tree with root $\Gamma \cup \{\sim\Theta\}$ that possesses **all closed branches**

Some Definitions

- "$\Gamma \nvdash_{STD} \Theta$" means that *it is not the case that* the argument from Γ to Θ is **tree-valid**
- i.e., *it is not the case that* there exists a tree with root $\Gamma \cup \{\sim\Theta\}$ that possesses **all closed branches**
- Equivalently: ANY tree with root $\Gamma \cup \{\sim\Theta\}$ possesses
  at least one complete open branch

Some Definitions

- "$\Gamma \nvdash_{STD} \Theta$" means that *it is not the case that* the argument from Γ to Θ is **tree-valid**
- i.e., *it is not the case that* there exists a tree with root $\Gamma \cup \{\sim\Theta\}$ that possesses **all closed branches**
- Equivalently: ANY tree with root $\Gamma \cup \{\sim\Theta\}$ possesses **at least one complete open branch**
- (Aside: this is NOT the same as saying that the argument is **tree-invalid**, since that only requires the existence of a single tree with a complete open branch)

Modifying system STD

- What if we had chosen an alternative rule(s)?

Modifying system STD

- What if we had chosen an alternative rule(s)?
- Simplest case: we swap out one of our nine rules for a different rule, i.e. for a given connective or negated connective.

Modifying system STD

- What if we had chosen an alternative rule(s)?
- Simplest case: we swap out one of our nine rules for a different rule, i.e. for a given connective or negated connective.
- Call our modified system 'STD^*'

Modifying system STD

- What if we had chosen an alternative rule(s)?
- Simplest case: we swap out one of our nine rules for a different rule, i.e. for a given connective or negated connective.
- Call our modified system 'STD^*'
- Would our modified system STD^* remain Sound?

Modifying system STD

- What if we had chosen an alternative rule(s)?
- Simplest case: we swap out one of our nine rules for a different rule, i.e. for a given connective or negated connective.
- Call our modified system 'STD^*'
- Would our modified system STD^* remain Sound?
- Would our modified system STD^* remain Complete?

Some Study Advice

- I recommend writing out on a ‘cheat sheet’ the following steps for checking whether a modified rule preserves (i) soundness or (ii) completeness

Some Study Advice

- I recommend writing out on a ‘cheat sheet’ the following steps for checking whether a modified rule preserves (i) soundness or (ii) completeness
- In each case, note on your sheet the method for extending the (i) soundness or (ii) completeness proof

Some Study Advice

- I recommend writing out on a 'cheat sheet' the following steps for checking whether a modified rule preserves (i) soundness or (ii) completeness
- In each case, note on your sheet the method for extending the (i) soundness or (ii) completeness proof
- Note the method for constructing a counterexample, i.e. what it would take to have (i) a counterexample to soundness or (ii) a counterexample to completeness

Some Study Advice

- I recommend writing out on a 'cheat sheet' the following steps for checking whether a modified rule preserves (i) soundness or (ii) completeness
- In each case, note on your sheet the method for extending the (i) soundness or (ii) completeness proof
- Note the method for constructing a counterexample, i.e. what it would take to have (i) a counterexample to soundness or (ii) a counterexample to completeness
- You probably want to have this information right in front of you during the exam, rather than to be scrambling looking for it

Checking Soundness: “top-down” reasoning

- **Heuristic for Soundness checking**: consider an **arbitrary** truth value assignment (TVA) that satisfies the sentence being resolved (i.e. *makes true* the sentence ‘at the top’ of the rule)

Checking Soundness: “top-down” reasoning

- **Heuristic for Soundness checking**: consider an **arbitrary** truth value assignment (TVA) that satisfies the sentence being resolved (i.e. *makes true* the sentence ‘at the top’ of the rule)
- Ask whether this TVA guarantees that **at least one** child-node is satisfied, i.e. the sentences on that child-node are all made true

Checking Soundness: “top-down” reasoning

- **Heuristic for Soundness checking**: consider an **arbitrary** truth value assignment (TVA) that satisfies the sentence being resolved (i.e. *makes true* the sentence ‘at the top’ of the rule)
- Ask whether this TVA guarantees that **at least one** child-node is satisfied, i.e. the sentences on that child-node are all made true
- If ‘**yes**’, then the rule preserves soundness: proceed to extend our soundness proof for this case (reasoning in terms of a TVA ‘$\mathcal{I}$’).

Checking Soundness: “top-down” reasoning

- **Heuristic for Soundness checking**: consider an **arbitrary** truth value assignment (TVA) that satisfies the sentence being resolved (i.e. *makes true* the sentence ‘at the top’ of the rule)
- Ask whether this TVA guarantees that **at least one** child-node is satisfied, i.e. the sentences on that child-node are all made true
- If ‘**yes**’, then the rule preserves soundness: proceed to extend our soundness proof for this case (reasoning in terms of a TVA ‘$\mathcal{I}$’).
- If ‘**no**’, then the rule BREAKS soundness: proceed to **construct a counter-example** using the rule

Counterexamples to Soundness

- To show soundness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences

Counterexamples to Soundness

- To show soundness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

Counterexamples to Soundness

- To show soundness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to soundness:

Counterexamples to Soundness

- To show soundness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to soundness:
  1.) Choose a satisfiable root, i.e. a **consistent** set of sentences

Counterexamples to Soundness

- To show soundness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to soundness:
  1.) Choose a satisfiable root, i.e. a **consistent** set of sentences
  2.) Apply the modified rule in the tree; you may also use other rules that we've already shown preserve soundness

Counterexamples to Soundness

- To show soundness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to soundness:
  1.) Choose a satisfiable root, i.e. a **consistent** set of sentences
  2.) Apply the modified rule in the tree; you may also use other rules that we've already shown preserve soundness
  3.) Show that the tree **closes** (i.e. NO complete open branches)

Why such Counterexamples work

- **Sound**: If $\Gamma \vdash_{STD^*} \Theta$, then $\Gamma \models \Theta$

Why such Counterexamples work

- **Sound**: If $\Gamma \vdash_{STD^*} \Theta$, then $\Gamma \models \Theta$
- Counterexample: $\Gamma \vdash_{STD^*} \Theta$ but $\Gamma \not\models \Theta$:

Why such Counterexamples work

- **Sound**: If $\Gamma \vdash_{STD^*} \Theta$, then $\Gamma \vDash \Theta$
- Counterexample: $\Gamma \vdash_{STD^*} \Theta$ but $\Gamma \nvDash \Theta$:
    - '$\Gamma \vdash_{STD^*} \Theta$' means tree with root $\Gamma \cup \{\sim\Theta\}$ **CLOSES** (i.e. is tree-valid in system STD^*)

Why such Counterexamples work

- **Sound**: If $\Gamma \vdash_{STD*} \Theta$, then $\Gamma \vDash \Theta$
- Counterexample: $\Gamma \vdash_{STD*} \Theta$ but $\Gamma \nvDash \Theta$:
  - '$\Gamma \vdash_{STD*} \Theta$' means tree with root $\Gamma \cup \{\sim\Theta\}$ **CLOSES** (i.e. is tree-valid in system $STD*$)
  - '$\Gamma \nvDash \Theta$' means that $\Gamma \cup \{\sim\Theta\}$ is **CONSISTENT** (i.e. satisfiable) (i.e. there exists a TVA that makes the premises Γ true but the conclusion Θ false)

Why such Counterexamples work

- **Sound**: If $\Gamma \vdash_{STD^*} \Theta$, then $\Gamma \vDash \Theta$
- Counterexample: $\Gamma \vdash_{STD^*} \Theta$ but $\Gamma \nvDash \Theta$:
  - '$\Gamma \vdash_{STD^*} \Theta$' means tree with root $\Gamma \cup \{\sim\Theta\}$ **CLOSES** (i.e. is tree-valid in system STD^*)
  - '$\Gamma \nvDash \Theta$' means that $\Gamma \cup \{\sim\Theta\}$ is **CONSISTENT** (i.e. satisfiable) (i.e. there exists a TVA that makes the premises Γ true but the conclusion Θ false)
- If our system *were* sound, then whenever a tree closes, it *would* correspond to a semantically valid argument.

Checking Completeness: “bottom-up” reasoning

- **Heuristic for Completeness checking**: ask whether **EACH** child node (below) individually entails the sentence being resolved (i.e. the sentence ‘up top’, in the ‘parent-node’)

Checking Completeness: “bottom-up” reasoning

- **Heuristic for Completeness checking**: ask whether **EACH** child node (below) individually entails the sentence being resolved (i.e. the sentence ‘up top’, in the ‘parent-node’)

- If ‘**yes**’, then the rule preserves completeness: proceed to extend our completeness proof for this case (reasoning in terms of arbitrary TVA’s, one for each child node).

Checking Completeness: “bottom-up” reasoning

- **Heuristic for Completeness checking**: ask whether **EACH** child node (below) individually entails the sentence being resolved (i.e. the sentence ‘up top’, in the ‘parent-node’)
- If ‘**yes**’, then the rule preserves completeness: proceed to extend our completeness proof for this case (reasoning in terms of arbitrary TVA’s, one for each child node).
- If ‘**no**’, then the rule BREAKS completeness: proceed to **construct a counter-example** using the rule.

Counterexamples to Completeness

- To show completeness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences

Counterexamples to Completeness

- To show completeness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

Counterexamples to Completeness

- To show completeness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to completeness:

Counterexamples to Completeness

- To show completeness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to completeness:
  1.) Choose an **UNsatisfiable root**, i.e. an INconsistent set of sentences

Counterexamples to Completeness

- To show completeness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to completeness:
  1.) Choose an **UNsatisfiable root**, i.e. an INconsistent set of sentences
  2.) Apply the modified rule in the tree; you may also use other rules that we've already shown preserve completeness

Counterexamples to Completeness

- To show completeness fails, it is NECESSARY to provide a CONCRETE counterexample, using SL sentences
  - Further heuristic reasoning about TVAs is not enough!

- What you need for a counterexample to completeness:
  1.) Choose an **UNsatisfiable root**, i.e. an INconsistent set of sentences
  2.) Apply the modified rule in the tree; you may also use other rules that we've already shown preserve completeness
  3.) Show that the tree has a **complete open branch**, i.e. does NOT close

Why such Counterexamples work

- **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD^*} \Theta$

Why such Counterexamples work

- **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD*} \Theta$
- Counterexample: $\Gamma \vDash \Theta$ but $\Gamma \nvdash_{STD*} \Theta$

Why such Counterexamples work

- **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD*} \Theta$
- Counterexample: $\Gamma \vDash \Theta$ but $\Gamma \nvdash_{STD*} \Theta$
  - '$\Gamma \vDash \Theta$' means that $\Gamma \cup \{\sim\Theta\}$ is INconsistent (i.e. UNsatisfiable) (i.e. any TVA that makes Γ true makes the conclusion Θ true)

Why such Counterexamples work

- **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD*} \Theta$
- Counterexample: $\Gamma \vDash \Theta$ but $\Gamma \nvdash_{STD*} \Theta$
  - '$\Gamma \vDash \Theta$' means that $\Gamma \cup \{\sim\Theta\}$ is INconsistent (i.e. UNsatisfiable) (i.e. any TVA that makes Γ true makes the conclusion Θ true)
  - '$\Gamma \nvdash_{STD*} \Theta$' means that it is NOT the case that the argument is tree-valid in $STD*$ (a claim about ALL trees)

Why such Counterexamples work

- **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD^*} \Theta$
- Counterexample: $\Gamma \vDash \Theta$ but $\Gamma \nvdash_{STD^*} \Theta$
  - '$\Gamma \vDash \Theta$' means that $\Gamma \cup \{\sim\Theta\}$ is INconsistent (i.e. UNsatisfiable) (i.e. any TVA that makes Γ true makes the conclusion Θ true)
  - '$\Gamma \nvdash_{STD^*} \Theta$' means that it is NOT the case that the argument is tree-valid in STD^* (a claim about ALL trees)
  - From completeness proof, we know that if the system *were* complete, then we could use any **complete open branch** to construct a TVA that **satisfies** the root

Why such Counterexamples work

- **Complete**: If $\Gamma \vDash \Theta$, then $\Gamma \vdash_{STD^*} \Theta$
- Counterexample: $\Gamma \vDash \Theta$ but $\Gamma \nvdash_{STD^*} \Theta$
  - '$\Gamma \vDash \Theta$' means that $\Gamma \cup \{\sim\Theta\}$ is INconsistent (i.e. UNsatisfiable) (i.e. any TVA that makes Γ true makes the conclusion Θ true)
  - '$\Gamma \nvdash_{STD^*} \Theta$' means that it is NOT the case that the argument is tree-valid in STD^* (a claim about ALL trees)
  - From completeness proof, we know that if the system *were* complete, then we could use any **complete open branch** to construct a TVA that **satisfies** the root
  - Hence, if we construct a **complete open tree** with an **unsatisfiable root**, this is a reductio of completeness

A Common Mistake to Avoid!!!!

- You are very liable to forget to COMPLETE YOUR open branch!!!

A Common Mistake to Avoid!!!!

- You are very liable to forget to COMPLETE YOUR open branch!!!
- You only have a counterexample to completeness if you have a COMPLETE open branch with an unsatisfiable root

A Common Mistake to Avoid!!!!

- You are very liable to forget to COMPLETE YOUR open branch!!!
- You only have a counterexample to completeness if you have a COMPLETE open branch with an unsatisfiable root
- So make sure you FULLY RESOLVE every sentence on the open branch, until you can write that coveted up arrow '$\uparrow$'!

A Common Mistake to Avoid!!!!

- You are very liable to forget to COMPLETE YOUR open branch!!!
- You only have a counterexample to completeness if you have a COMPLETE open branch with an unsatisfiable root
- So make sure you FULLY RESOLVE every sentence on the open branch, until you can write that coveted up arrow '$\uparrow$'!
- When in doubt, just complete the whole tree

Liberal Conditional and Conservative Biconditional

*STD**: replace rule (⊃) with:

Liberal Conditional (L⊃)

$$
\begin{array}{lccl}
\text{m.} & & \Phi \supset \Psi & \\
\vdots & & \vdots & \\
& \swarrow & & \searrow \\
\text{j.} & {\sim}\Phi & & \Psi \vee \Phi \qquad \text{m L}\supset
\end{array}
$$

Liberal Conditional and Conservative Biconditional

*STD**: replace rule ($\supset$) with:

Liberal Conditional (L$\supset$)

m.	$\Phi \supset \Psi$		
$\vdots$	$\vdots$		
j.	$\sim\Phi$	$\Psi \vee \Phi$	m L$\supset$

STD†: replace rule ($\equiv$) with:

Conservative Biconditional (C$\equiv$)

m.	$\Phi \equiv \Psi$	
$\vdots$	$\vdots$	
j.	Φ	m C$\equiv$
j+1.	Ψ	

Conservative Biconditional

- Reason from the top-down:
  $\Phi \equiv \Psi$ is logically equivalent to $(\Phi \,\&\, \Psi) \vee ({\sim}\Phi \,\&\, {\sim}\Psi)$

Conservative Biconditional

- Reason from the top-down:
  $\Phi \equiv \Psi$ is logically equivalent to $(\Phi \,\&\, \Psi) \vee (\sim\Phi \,\&\, \sim\Psi)$
- The node below is logically equivalent to $(\Phi \,\&\, \Psi)$

Conservative Biconditional

- Reason from the top-down:
  $\Phi \equiv \Psi$ is logically equivalent to $(\Phi \,\&\, \Psi) \vee (\sim\Phi \,\&\, \sim\Psi)$
- The node below is logically equivalent to $(\Phi \,\&\, \Psi)$
- Note that the top does NOT entail the bottom! Given an interpretation that satisfies $\Phi \equiv \Psi$, it is NOT guaranteed to satisfy the sentence(s) in at least one branch below.

Conservative Biconditional

- Reason from the top-down:
  $\Phi \equiv \Psi$ is logically equivalent to $(\Phi \,\&\, \Psi) \vee (\sim\Phi \,\&\, \sim\Psi)$
- The node below is logically equivalent to $(\Phi \,\&\, \Psi)$
- Note that the top does NOT entail the bottom! Given an interpretation that satisfies $\Phi \equiv \Psi$, it is NOT guaranteed to satisfy the sentence(s) in at least one branch below.
- Hence, **to the counterexample!**
  (note that the problem REQUIRES this! can't stop won't stop!)

Counterexample to Soundness of $STD^{\dagger}$

- For a counterexample, need: $\Gamma \vdash_{STD^{\dagger}} \Theta$ but $\Gamma \nvDash \Theta$

Counterexample to Soundness of $STD^\dagger$

- For a counterexample, need: $\Gamma \vdash_{STD^\dagger} \Theta$ but $\Gamma \not\models \Theta$

  $STD^\dagger$: replace rule ($\equiv$) with *Conservative Biconditional* (C$\equiv$):

1.	$P \equiv P$	Premise (PR)
2.	$\sim P$	$\sim$Conclusion
3.	P	1 C$\equiv$
4.	P	
	$\times$	
	2, 3, so tree-valid in $STD^\dagger$	

Counterexample to Soundness of $STD^{\dagger}$

- For a counterexample, need: $\Gamma \vdash_{STD^{\dagger}} \Theta$ but $\Gamma \nvDash \Theta$

  $STD^{\dagger}$: replace rule ($\equiv$) with *Conservative Biconditional* (C$\equiv$):

$$
\begin{array}{lcl}
1. & P \equiv P & \text{Premise (PR)} \\
2. & \sim P & \sim\text{Conclusion} \\
 & | & \\
3. & P & 1\ \text{C}\equiv \\
4. & P & \\
 & \times & \\
 & \text{2, 3, so tree-valid in } STD^{\dagger} &
\end{array}
$$

- But $(P \equiv P) \nvDash P$ because $\{(P \equiv P), \sim P\}$ is consistent! Assign 'P' false! N.B.: your counterexample MUST use actual sentences of SL; not meta-variables!

Conservative Biconditional (bottoms up!)

- Reason from the bottom up: there is only one child node. So we only need its sentences to entail $\Phi \equiv \Psi$

Conservative Biconditional (bottoms up!)

- Reason from the bottom up: there is only one child node. So we only need its sentences to entail $\Phi \equiv \Psi$
- Sentences on bottom are equivalent to $(\Phi \,\&\, \Psi)$, which DOES entail $\Phi \equiv \Psi$. So we proceed to formally extend our completeness proof:

Conservative Biconditional (bottoms up!)

- Reason from the bottom up: there is only one child node. So we only need its sentences to entail $\Phi \equiv \Psi$
- Sentences on bottom are equivalent to $(\Phi \,\&\, \Psi)$, which DOES entail $\Phi \equiv \Psi$. So we proceed to formally extend our completeness proof:
  - **Formally**: since only one child node, both Φ and Ψ lie on the complete open branch O.

Conservative Biconditional (bottoms up!)

- Reason from the bottom up: there is only one child node. So we only need its sentences to entail $\Phi \equiv \Psi$
- Sentences on bottom are equivalent to $(\Phi \,\&\, \Psi)$, which DOES entail $\Phi \equiv \Psi$. So we proceed to formally extend our completeness proof:
  - **Formally**: since only one child node, both Φ and Ψ lie on the complete open branch O.
  - By induction hypothesis, both are true according to $\mathcal{I}$.

Conservative Biconditional (bottoms up!)

- Reason from the bottom up: there is only one child node. So we only need its sentences to entail $\Phi \equiv \Psi$
- Sentences on bottom are equivalent to $(\Phi \mathbin{\&} \Psi)$, which DOES entail $\Phi \equiv \Psi$. So we proceed to formally extend our completeness proof:
    - **Formally**: since only one child node, both Φ and Ψ lie on the complete open branch O.
    - By induction hypothesis, both are true according to $\mathcal{I}$.
    - Hence, $\Phi \equiv \Psi$ is true on $\mathcal{I}$, which is what we needed to show.

7. Midterm Review!

f. Translation/Symbolization in SL

Unless

- Translate ‘unless’ as ‘or’ (unless you want to make things needlessly complicated for yourself!)

Unless

- Translate ‘unless’ as ‘or’ (unless you want to make things needlessly complicated for yourself!)

- Example: I am not going to not Vote in the midterms, unless a meteor Destroys my polling place and the city of Cambridge does not give me an Alternative location.

Unless

- Translate ‘unless’ as ‘or’ (unless you want to make things needlessly complicated for yourself!)
- Example: I am not going to not Vote in the midterms, unless a meteor Destroys my polling place and the city of Cambridge does not give me an Alternative location.
- $\sim\sim V \vee (D \,\&\, \sim A)$

Provided that; lonely if; given: these flip the order!

- ‘Q provided that P’ is ‘If P, then Q’

Provided that; lonely if; given: these flip the order!

- ‘Q provided that P’ is ‘If P, then Q’
- So just like the “lonely if”!

Provided that; lonely if; given: these flip the order!

- ‘Q provided that P’ is ‘If P, then Q’
- So just like the “lonely if”!
- ‘Q if P’ is ‘If P, then Q’

Provided that; lonely if; given: these flip the order!

- ‘Q provided that P’ is ‘If P, then Q’
- So just like the “lonely if”!
- ‘Q if P’ is ‘If P, then Q’
- Likewise for ‘Q given P’: ‘If P, then Q’

Only if

- ‘P only if Q’ is ‘If P, then Q’

Only if

- 'P only if Q' is 'If P, then Q'
- so order is preserved:

Only if

- 'P only if Q' is 'If P, then Q'
- so order is preserved:
- Intuition: 'P only if Q' means 'If not Q, then not P', which is '$\sim Q \supset \sim P$', i.e. a contrapositive

Only if

- 'P only if Q' is 'If P, then Q'
- so order is preserved:
- Intuition: 'P only if Q' means 'If not Q, then not P', which is '$\sim Q \supset \sim P$', i.e. a contrapositive
- A contrapositive is logically equivalent to its conditional: $P \supset Q$

Just in Case

- Translate “just in case” as “if and only if”

Just in Case

- Translate “just in case” as “if and only if”
- Example: I will do Well in this class just in case I do not Party every night.

Just in Case

- Translate “just in case” as “if and only if”
- Example: I will do Well in this class just in case I do not Party every night.
- $W \equiv \sim P$

‘Although’ and ‘But’

- both ‘although’ and ‘but’ are translated as ‘and’

‘Although’ and ‘But’

- both ‘although’ and ‘but’ are translated as ‘and’
- i.e. schematized as $\&$

‘Although’ and ‘But’

- both ‘although’ and ‘but’ are translated as ‘and’
- i.e. schematized as &
- Example: Although I Meditate every day, I remain an Anxious person, but I Hope to overcome this in the future.

‘Although’ and ‘But’

- both ‘although’ and ‘but’ are translated as ‘and’
- i.e. schematized as &
- Example: Although I Meditate every day, I remain an Anxious person, but I Hope to overcome this in the future.
- $(M \& A) \& H$

7. Midterm Review!

g. Truth Tables: (In)Validity and (In)Equivalence

Proving an argument is Valid

- To use a truth table to prove that an argument is valid, one must:

Proving an argument is Valid

- To use a truth table to prove that an argument is valid, one must:
- Complete the entire truth table!

Proving an argument is Valid

- To use a truth table to prove that an argument is valid, one must:
- Complete the entire truth table!
- One must show that there is no truth value assignment that makes all of the premises true while making the conclusion false (i.e. no counterexamples to validity)

Proving an argument is Valid

- To use a truth table to prove that an argument is valid, one must:
- Complete the entire truth table!
- One must show that there is no truth value assignment that makes all of the premises true while making the conclusion false (i.e. no counterexamples to validity)
- Equivalently: every TVA that makes the premises true makes the conclusion true

Proving an argument is Valid

- To use a truth table to prove that an argument is valid, one must:
- Complete the entire truth table!
- One must show that there is no truth value assignment that makes all of the premises true while making the conclusion false (i.e. no counterexamples to validity)
- Equivalently: every TVA that makes the premises true makes the conclusion true
- (note that this ‘every’ claim is vacuously true if your conclusion is a tautology, since a tautology is not false on any TVA)

Proving an argument is Invalid

- To prove that an argument is invalid, it suffices to:

Proving an argument is Invalid

- To prove that an argument is invalid, it suffices to:
- Identify at least one truth value assignment that makes the premises true but the conclusion false

Proving an argument is Invalid

- To prove that an argument is invalid, it suffices to:
- Identify at least one truth value assignment that makes the premises true but the conclusion false
- In this case, unnecessary to complete the entire truth table

Proving an argument is Invalid

- To prove that an argument is invalid, it suffices to:
- Identify at least one truth value assignment that makes the premises true but the conclusion false
- In this case, unnecessary to complete the entire truth table
- *Carnap* will let you enter the relevant TVA as a counterexample to validity (make sure you understand the syntax for this!)

Proving that two sentences are Equivalent

- Using a truth table, to prove that two sentences are equivalent:

Proving that two sentences are Equivalent

- Using a truth table, to prove that two sentences are equivalent:

- You must complete the entire table!

Proving that two sentences are Equivalent

- Using a truth table, to prove that two sentences are equivalent:
- You must complete the entire table!
- You must show that the two sentences receive the same truth value for every possible truth value assignment (i.e. every row of the truth table)

Proving that two Sentences are Inequivalent

- Suffices to identify a single TVA where the sentences have different truth values

Proving that two Sentences are Inequivalent

- Suffices to identify a single TVA where the sentences have different truth values
- In this case, unnecessary to complete the entire truth table

Proving that two Sentences are Inequivalent

- Suffices to identify a single TVA where the sentences have different truth values
- In this case, unnecessary to complete the entire truth table
- *Carnap* will let you enter the relevant TVA as a counterexample to equivalence

7. Midterm Review!

h. Natural Deduction

Some rules to Definitely Understand

- Disjunction Elimination

Some rules to Definitely Understand

- Disjunction Elimination
- Conditional Introduction

Some rules to Definitely Understand

- Disjunction Elimination
- Conditional Introduction
- Negation introduction; Negation Elimination

Some rules to Definitely Understand

- Disjunction Elimination
- Conditional Introduction
- Negation introduction; Negation Elimination
- Understand when and how to start a subproof

Some rules to Definitely Understand

- Disjunction Elimination
- Conditional Introduction
- Negation introduction; Negation Elimination
- Understand when and how to start a subproof
- For rules that cite a subproof, remember to use a HYPHEN between line numbers in the justification (you are citing the ENTIRE subproof, even in those cases where the sub proof is itself only 2 lines long)

Using Conditional Introduction $\supset I$

- If you need to construct a conditional, typically you assume the antecedent, starting a subproof (tab in and write :AS after the antecedent)

Using Conditional Introduction $\supset I$

- If you need to construct a conditional, typically you assume the antecedent, starting a subproof (tab in and write :AS after the antecedent)

- Derive the consequent within the subproof

Using Conditional Introduction $\supset I$

- If you need to construct a conditional, typically you assume the antecedent, starting a subproof (tab in and write :AS after the antecedent)
- Derive the consequent within the subproof
- Exit the subproof by writing the conditional and justifying with $\supset I$

Common Mistake with Negation Rules

- Don't forget that you need both a sentence Ψ and $\sim\Psi$ WITHIN the subproof, i.e. under the assumption line

Common Mistake with Negation Rules

- Don't forget that you need both a sentence Ψ and $\sim\Psi$ WITHIN the subproof, i.e. under the assumption line
- Often, if you are using a sentence that has already occurred as one of the Ψ or $\sim\Psi$, you will need to reiterate it on a line within the subproof

Common Mistake with Negation Rules

- Don't forget that you need both a sentence Ψ and $\sim\Psi$ WITHIN the subproof, i.e. under the assumption line
- Often, if you are using a sentence that has already occurred as one of the Ψ or $\sim\Psi$, you will need to reiterate it on a line within the subproof
- *Carnap* seems to be a bit fussy: the Ψ and $\sim\Psi$ cannot themselves be separated by a subproof within your subproof