

First-Order Logic: Theory and Practice

Christoph Benzmüller

Freie Universität Berlin

Block Lecture, WS 2012, October 1-12, 2012

Propositional Logic

(this part of the lecture very closely follows Fitting's textbook)

Propositional Logic: Syntax

Definition — Syntax Propositional Logic

1

Let \mathbf{L} be a (countable) set of propositional letters.
The language \mathbf{P} is defined as

$$\begin{aligned} X, Y ::= \quad & \mathbf{L} \mid \perp \mid \top \mid \neg X \mid (X \vee Y) \mid (X \wedge Y) \mid (X \supset Y) \mid (X \subset Y) \mid \\ & (X \not\supset Y) \mid (X \not\subset Y) \mid (X \uparrow Y) \mid (X \downarrow Y) \mid (X \equiv Y) \mid (X \not\equiv Y) \end{aligned}$$

More formally: \mathbf{P} is the smallest set such that

1. if $X \in \mathbf{L}$ then $X \in \mathbf{P}$; moreover, $\perp, \top \in \mathbf{P}$
(X, \perp, \top are called *atomic formulas*)
2. if $X \in \mathbf{P}$ then $\neg X \in \mathbf{P}$
3. if $X, Y \in \mathbf{P}$ then $(X \circ Y) \in \mathbf{P}$ for $\circ \in \{\vee, \wedge, \supset, \dots\}$

Remark: We could add further connectives (or avoid some).

Remark: Brackets will be avoided if structure is clear in context.

Definition — Syntax Propositional Logic

1

Let \mathbf{L} be a (countable) set of propositional letters.
The language \mathbf{P} is defined as

$$\begin{aligned} X, Y ::= \quad & \mathbf{L} \mid \perp \mid \top \mid \neg X \mid (X \vee Y) \mid (X \wedge Y) \mid (X \supset Y) \mid (X \subset Y) \mid \\ & (X \not\supset Y) \mid (X \not\subset Y) \mid (X \uparrow Y) \mid (X \downarrow Y) \mid (X \equiv Y) \mid (X \not\equiv Y) \end{aligned}$$

More formally: \mathbf{P} is the smallest set such that

1. if $X \in \mathbf{L}$ then $X \in \mathbf{P}$; moreover, $\perp, \top \in \mathbf{P}$
(X, \perp, \top are called *atomic* formulas)
2. if $X \in \mathbf{P}$ then $\neg X \in \mathbf{P}$
3. if $X, Y \in \mathbf{P}$ then $(X \circ Y) \in \mathbf{P}$ for $\circ \in \{\vee, \wedge, \supset, \dots\}$

Remark: We could add further connectives (or avoid some).

Remark: Brackets will be avoided if structure is clear in context.

Definition — Syntax Propositional Logic

1

Let \mathbf{L} be a (countable) set of propositional letters.
The language \mathbf{P} is defined as

$$\begin{aligned} X, Y ::= \quad & \mathbf{L} \mid \perp \mid \top \mid \neg X \mid (X \vee Y) \mid (X \wedge Y) \mid (X \supset Y) \mid (X \subset Y) \mid \\ & (X \not\supset Y) \mid (X \not\subset Y) \mid (X \uparrow Y) \mid (X \downarrow Y) \mid (X \equiv Y) \mid (X \not\equiv Y) \end{aligned}$$

More formally: \mathbf{P} is the smallest set such that

1. if $X \in \mathbf{L}$ then $X \in \mathbf{P}$; moreover, $\perp, \top \in \mathbf{P}$
(X, \perp, \top are called *atomic* formulas)
2. if $X \in \mathbf{P}$ then $\neg X \in \mathbf{P}$
3. if $X, Y \in \mathbf{P}$ then $(X \circ Y) \in \mathbf{P}$ for $\circ \in \{\vee, \wedge, \supset, \dots\}$

Remark: We could add further connectives (or avoid some).

Remark: Brackets will be avoided if structure is clear in context.

Definition — Syntax Propositional Logic

1

Let \mathbf{L} be a (countable) set of propositional letters.
The language \mathbf{P} is defined as

$$\begin{aligned} X, Y ::= \quad & \mathbf{L} \mid \perp \mid \top \mid \neg X \mid (X \vee Y) \mid (X \wedge Y) \mid (X \supset Y) \mid (X \subset Y) \mid \\ & (X \not\supset Y) \mid (X \not\subset Y) \mid (X \uparrow Y) \mid (X \downarrow Y) \mid (X \equiv Y) \mid (X \not\equiv Y) \end{aligned}$$

More formally: \mathbf{P} is the smallest set such that

1. if $X \in \mathbf{L}$ then $X \in \mathbf{P}$; moreover, $\perp, \top \in \mathbf{P}$
(X, \perp, \top are called *atomic* formulas)
2. if $X \in \mathbf{P}$ then $\neg X \in \mathbf{P}$
3. if $X, Y \in \mathbf{P}$ then $(X \circ Y) \in \mathbf{P}$ for $\circ \in \{\vee, \wedge, \supset, \dots\}$

Remark: We could add further connectives (or avoid some).

Remark: Brackets will be avoided if structure is clear in context.

Definition — Syntax Propositional Logic

1

Let \mathbf{L} be a (countable) set of propositional letters.
The language \mathbf{P} is defined as

$$\begin{aligned} X, Y ::= \quad & \mathbf{L} \mid \perp \mid \top \mid \neg X \mid (X \vee Y) \mid (X \wedge Y) \mid (X \supset Y) \mid (X \subset Y) \mid \\ & (X \not\supset Y) \mid (X \not\subset Y) \mid (X \uparrow Y) \mid (X \downarrow Y) \mid (X \equiv Y) \mid (X \not\equiv Y) \end{aligned}$$

More formally: \mathbf{P} is the smallest set such that

1. if $X \in \mathbf{L}$ then $X \in \mathbf{P}$; moreover, $\perp, \top \in \mathbf{P}$
(X, \perp, \top are called *atomic* formulas)
2. if $X \in \mathbf{P}$ then $\neg X \in \mathbf{P}$
3. if $X, Y \in \mathbf{P}$ then $(X \circ Y) \in \mathbf{P}$ for $\circ \in \{\vee, \wedge, \supset, \dots\}$

Remark: We could add further connectives (or avoid some).

Remark: Brackets will be avoided if structure is clear in context.

Propositional Logic: Syntax

Definition — Syntax Propositional Logic

1

Let \mathbf{L} be a (countable) set of propositional letters.
The language \mathbf{P} is defined as

$$\begin{aligned} X, Y ::= \quad \mathbf{L} \mid \perp \mid \top \mid \neg X \mid (X \vee Y) \mid (X \wedge Y) \mid (X \supset Y) \mid (X \subset Y) \mid \\ (X \not\supset Y) \mid (X \not\subset Y) \mid (X \uparrow Y) \mid (X \downarrow Y) \mid (X \equiv Y) \mid (X \not\equiv Y) \end{aligned}$$

More formally: \mathbf{P} is the smallest set such that

1. if $X \in \mathbf{L}$ then $X \in \mathbf{P}$; moreover, $\perp, \top \in \mathbf{P}$
(X, \perp, \top are called *atomic* formulas)
2. if $X \in \mathbf{P}$ then $\neg X \in \mathbf{P}$
3. if $X, Y \in \mathbf{P}$ then $(X \circ Y) \in \mathbf{P}$ for $\circ \in \{\vee, \wedge, \supset, \dots\}$

Remark: We could add further connectives (or avoid some).

Remark: Brackets will be avoided if structure is clear in context.

Definition — Syntax Propositional Logic

1

Let \mathbf{L} be a (countable) set of propositional letters.
The language \mathbf{P} is defined as

$$\begin{aligned} X, Y ::= \quad \mathbf{L} \mid \perp \mid \top \mid \neg X \mid (X \vee Y) \mid (X \wedge Y) \mid (X \supset Y) \mid (X \subset Y) \mid \\ (X \not\supset Y) \mid (X \not\subset Y) \mid (X \uparrow Y) \mid (X \downarrow Y) \mid (X \equiv Y) \mid (X \not\equiv Y) \end{aligned}$$

More formally: \mathbf{P} is the smallest set such that

1. if $X \in \mathbf{L}$ then $X \in \mathbf{P}$; moreover, $\perp, \top \in \mathbf{P}$
(X, \perp, \top are called *atomic* formulas)
2. if $X \in \mathbf{P}$ then $\neg X \in \mathbf{P}$
3. if $X, Y \in \mathbf{P}$ then $(X \circ Y) \in \mathbf{P}$ for $\circ \in \{\vee, \wedge, \supset, \dots\}$

Remark: We could add further connectives (or avoid some).

Remark: Brackets will be avoided if structure is clear in context.

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ Induction steps:
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula.

Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principal of Structural Induction

2

Every formula of propositional logic has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** Every atomic formula has property \mathbf{Q} .
- ▶ **Induction steps:**
 - ▶ If X has property \mathbf{Q} so does $\neg X$
 - ▶ If X and Y have property \mathbf{Q} so does $(X \circ Y)$, where \circ is a binary connective.

Proof: Let $\mathbf{Q}^* \subseteq \mathbf{P}$ be the formulas that have property \mathbf{Q} . The basis step and the induction steps ensure that \mathbf{Q}^* meets the conditions 1, 2 and 3 in the definition of propositional formula. Since \mathbf{P} , the set of propositional formulas, is the smallest set meeting these conditions, we have $\mathbf{P} \subseteq \mathbf{Q}^*$. Hence, $\mathbf{P} = \mathbf{Q}^*$, i.e. every propositional formula has property \mathbf{Q} .

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Propositional Logic: Syntax

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Propositional Logic: Syntax

Theorem — Principle of Structural Recursion

3

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas.*
- ▶ **Recursion steps:**
 - ▶ *The value of f for $\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for $(X \circ Y)$, where \circ is a binary connective, is defined in terms of the values of f for X and Y .*

Proof: ... similar to above ...

Example — Degree-function d

4

$$d(P) = 0 \text{ for all atomic } P. \quad d(\neg X) = d(X) + 1.$$

$$d(X \circ Y) = d(X) + d(Y) + 1$$

Definition — Semantics of Propositional Logic

5

- ▶ Boolean valuation function $v : \mathbf{P} \longrightarrow \{\text{t}, \text{f}\}$
- for $X \in \mathbf{L}$ choose $v(X) \in \{\text{t}, \text{f}\}$
- v is fixed for \perp and \top : $v(\perp) = \text{f}, v(\top) = \text{t}$
- v is fixed for \neg : $v(\neg X) = \neg v(X)$
- v is fixed for $\vee, \wedge, \supset, \subset, \uparrow, \downarrow, \neq, \equiv$

	\vee	\wedge	\supset	\subset	\uparrow	\downarrow	\neq	\equiv	
t	t	t	t	t	f	f	f	t	f
f	t	f	f	t	t	f	t	f	t
f	t	f	t	f	t	f	t	f	t
f	f	f	t	t	t	t	f	t	f

Definition — Semantics of Propositional Logic

5

- ▶ Boolean valuation function $v : \mathbf{P} \longrightarrow \{\text{t}, \text{f}\}$
- for $X \in \mathbf{L}$ choose $v(X) \in \{\text{t}, \text{f}\}$
- v is fixed for \perp and \top : $v(\perp) = \text{f}, v(\top) = \text{t}$
- v is fixed for \neg : $v(\neg X) = \neg v(X)$
- v is fixed for $\vee, \wedge, \supset, \subset, \uparrow, \downarrow, \neq, \equiv$

	\vee	\wedge	\supset	\subset	\uparrow	\downarrow	\neq	\equiv
t	t	t	t	t	f	f	f	t
f	t	f	f	t	t	f	f	t
f	t	f	t	f	t	f	t	t
f	f	f	t	t	t	t	f	f

Definition — Semantics of Propositional Logic

5

- ▶ Boolean valuation function $v : \mathbf{P} \longrightarrow \{\text{t}, \text{f}\}$
- for $X \in \mathbf{L}$ choose $v(X) \in \{\text{t}, \text{f}\}$
- v is fixed for \perp and \top : $v(\perp) = \text{f}, v(\top) = \text{t}$
- v is fixed for \neg : $v(\neg X) = \neg v(X)$
- v is fixed for $\vee, \wedge, \supset, \subset, \uparrow, \downarrow, \neq, \equiv$

	\vee	\wedge	\supset	\subset	\uparrow	\downarrow	\neq	\equiv	
t	t	t	t	t	f	f	f	f	t
f	t	f	f	t	t	f	t	f	t
t	t	f	t	f	t	f	f	t	t
f	f	f	t	t	t	t	f	f	f

Evaluation of complex propositional formulas

... blackboard and/or exercises ...

Complete sets of connectives / Definability

- ▶ $\{\neg\}$ is not a complete set of connectives
- ▶ $\{\neg, \vee\}$ is a complete set of connectives
- ▶ $\{\neg, \supset\}$ is a complete set of connectives
- ▶ $\{\uparrow\}$ is a complete set of connectives
- ▶ $\{\downarrow\}$ is a complete set of connectives

... blackboard or exercises ...

Evaluation of complex propositional formulas

... blackboard and/or exercises ...

Complete sets of connectives / Definability

- ▶ $\{\neg\}$ is not a complete set of connectives
- ▶ $\{\neg, \vee\}$ is a complete set of connectives
- ▶ $\{\neg, \supset\}$ is a complete set of connectives
- ▶ $\{\uparrow\}$ is a complete set of connectives
- ▶ $\{\downarrow\}$ is a complete set of connectives

... blackboard or exercises ...

Definition — Tautology

6

Formula $X \in \mathbf{P}$ is a *tautology* if $v(X) = \mathbf{t}$ for every Boolean valuation v .

Definition — Satisfiable

7

A set S of propositional formulas is *satisfiable* if some Boolean valuation v maps every member of S to \mathbf{t} .

Example — Which are tautologies? And which are not?

8

1. $((X \supset Y) \supset X) \supset X$
2. ... see exercises 2.4.1–2.4.3 in Fitting ...

Definition — Tautology

6

Formula $X \in \mathbf{P}$ is a *tautology* if $v(X) = \mathbf{t}$ for every Boolean valuation v .

Definition — Satisfiable

7

A set S of propositional formulas is *satisfiable* if some Boolean valuation v maps every member of S to \mathbf{t} .

Example — Which are tautologies? And which are not?

8

1. $((X \supset Y) \supset X) \supset X$
2. ... see exercises 2.4.1–2.4.3 in Fitting ...

Definition — Tautology

6

Formula $X \in \mathbf{P}$ is a *tautology* if $v(X) = \mathbf{t}$ for every Boolean valuation v .

Definition — Satisfiable

7

A set S of propositional formulas is *satisfiable* if some Boolean valuation v maps every member of S to \mathbf{t} .

Example — Which are tautologies? And which are not?

8

1. $((X \supset Y) \supset X) \supset X$
2. ... see exercises 2.4.1–2.4.3 in Fitting ...

Propositional Logic: Duality

Definition — Duality

9

Let \circ and \bullet be binary connectives. We say \bullet is the *dual* of \circ if $\neg(X \circ Y) = (\neg X \bullet \neg Y)$ (for all $X, Y \in \mathbf{P}$). We say, \circ is dual to \bullet and \bullet is dual to \circ .

Definition — Dual formula

10

For $X \in \mathbf{P}$ we define X^d as follows: replace all \perp in X by \top and all \top in X by \perp . Moreover, replace all binary connectives \circ by their dual \bullet .

Propositional Logic: Duality

Definition — Duality

9

Let \circ and \bullet be binary connectives. We say \bullet is the *dual* of \circ if $\neg(X \circ Y) = (\neg X \bullet \neg Y)$ (for all $X, Y \in \mathbf{P}$). We say, \circ is dual to \bullet and \bullet is dual to \circ .

Definition — Dual formula

10

For $X \in \mathbf{P}$ we define X^d as follows: replace all \perp in X by \top and all \top in X by \perp . Moreover, replace all binary connectives \circ by their dual \bullet .

Notation

$F(P)$ means that formula $F \in \mathbf{P}$ may contain one or several occurrences of propositional letter $P \in \mathbf{L}$. $F(X)$ is the formula that is obtained from $F(P)$ by replacing all occurrences of P in F by X .

Theorem — Replacement Theorem (version one)

11

Let $X, Y, F(P) \in \mathbf{P}$. If $v(X) = v(Y)$ then $v(F(X)) = v(F(Y))$.

Proof: ... exercise ...

Theorem — Replacement Theorem (version two)

12

If $X \equiv Y$ is a tautology then $F(X) \equiv F(Y)$ is a tautology.

Proof: ... exercise ...

Notation

$F(P)$ means that formula $F \in \mathbf{P}$ may contain one or several occurrences of propositional letter $P \in \mathbf{L}$. $F(X)$ is the formula that is obtained from $F(P)$ by replacing all occurrences of P in F by X .

Theorem — Replacement Theorem (version one)

11

Let $X, Y, F(P) \in \mathbf{P}$. If $v(X) = v(Y)$ then $v(F(X)) = v(F(Y))$.

Proof: ... exercise ...

Theorem — Replacement Theorem (version two)

12

If $X \equiv Y$ is a tautology then $F(X) \equiv F(Y)$ is a tautology.

Proof: ... exercise ...

Notation

$F(P)$ means that formula $F \in \mathbf{P}$ may contain one or several occurrences of propositional letter $P \in \mathbf{L}$. $F(X)$ is the formula that is obtained from $F(P)$ by replacing all occurrences of P in F by X .

Theorem — Replacement Theorem (version one) 11

Let $X, Y, F(P) \in \mathbf{P}$. If $v(X) = v(Y)$ then $v(F(X)) = v(F(Y))$.

Proof: ... exercise ...

Theorem — Replacement Theorem (version two) 12

If $X \equiv Y$ is a tautology then $F(X) \equiv F(Y)$ is a tautology.

Proof: ... exercise ...

Definition — Negation Normal Form

13

A formula $F \in \mathbf{P}$ is in *negation normal form* if the only negation symbols in F occur in front of propositional letters.

Proposition

14

Every propositional formula can be put into a negation normal form.

Proof: ... exercise ... maybe we write an algorithm in the practical exercises ...

Definition — Negation Normal Form

13

A formula $F \in \mathbf{P}$ is in *negation normal form* if the only negation symbols in F occur in front of propositional letters.

Proposition

14

Every propositional formula can be put into a negation normal form.

Proof: ... exercise ... maybe we write an algorithm in the practical exercises ...

Idea (Smullyan): Use large set of basic connectives but reduce the work e.g. in proving theorems.

Idea (Smullyan): Use large set of basic connectives but reduce the work e.g. in proving theorems.

Idea (Smullyan): Use large set of basic connectives but reduce the work e.g. in proving theorems.

Remark: From now on, \equiv and $\not\equiv$ are considered as defined.

Definition — α - and β -Formulas

15

Conjunctive			Disjunctive		
α	α_1	α_2	β	β_1	β_2
$X \wedge Y$	X	Y	$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$\neg(X \vee Y)$	$\neg X$	$\neg Y$	$X \vee Y$	X	Y
$\neg(X \supset Y)$	X	$\neg Y$	$X \supset Y$	$\neg X$	Y
$\neg(X \subset Y)$	$\neg X$	Y	$X \subset Y$	X	$\neg Y$
$\neg(X \uparrow Y)$	X	Y	$X \uparrow Y$	$\neg X$	$\neg Y$
$X \downarrow Y$	$\neg X$	$\neg Y$	$\neg(X \downarrow Y)$	X	Y
$X \not\supset Y$	X	$\neg Y$	$\neg(X \not\supset Y)$	$\neg X$	Y
$X \not\subset Y$	$\neg X$	Y	$\neg(X \not\subset Y)$	X	$\neg Y$

Propositional Logic: Uniform Notation

Proposition — Uniform Notation 1

16

For every Boolean valuation v and for all α - and β -formulas we have:

- ▶ $v(\alpha) = v(\alpha_1) \wedge v(\alpha_2)$
- ▶ $v(\beta) = v(\beta_1) \vee v(\beta_2)$

Proof: ... exercise ...

Proposition — Uniform Notation 2

17

$\alpha \equiv (\alpha_1 \wedge \alpha_2)$ and $\beta \equiv (\beta_1 \vee \beta_2)$ are tautologies.

Proof: ... exercise ...

Propositional Logic: Uniform Notation

Proposition — Uniform Notation 1

16

For every Boolean valuation v and for all α - and β -formulas we have:

- ▶ $v(\alpha) = v(\alpha_1) \wedge v(\alpha_2)$
- ▶ $v(\beta) = v(\beta_1) \vee v(\beta_2)$

Proof: ... exercise ...

Proposition — Uniform Notation 2

17

$\alpha \equiv (\alpha_1 \wedge \alpha_2)$ and $\beta \equiv (\beta_1 \vee \beta_2)$ are tautologies.

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Induction 18

Every propositional formula has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** *Every atomic formula and its negation has property \mathbf{Q}*
- ▶ **Induction steps:**
 - ▶ *If X has property \mathbf{Q} , so does $\neg\neg X$.*
 - ▶ *If α_1 and α_2 have property \mathbf{Q} , so does α .*
 - ▶ *If β_1 and β_2 have property \mathbf{Q} , so does β .*

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Induction 18

Every propositional formula has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** *Every atomic formula and its negation has property \mathbf{Q}*
- ▶ **Induction steps:**
 - ▶ *If X has property \mathbf{Q} , so does $\neg\neg X$.*
 - ▶ *If α_1 and α_2 have property \mathbf{Q} , so does α .*
 - ▶ *If β_1 and β_2 have property \mathbf{Q} , so does β .*

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Induction 18

Every propositional formula has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** *Every atomic formula and its negation has property \mathbf{Q}*
- ▶ **Induction steps:**
 - ▶ *If X has property \mathbf{Q} , so does $\neg\neg X$.*
 - ▶ *If α_1 and α_2 have property \mathbf{Q} , so does α .*
 - ▶ *If β_1 and β_2 have property \mathbf{Q} , so does β .*

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Induction 18

Every propositional formula has a property, \mathbf{Q} , provided:

- ▶ **Basis step:** *Every atomic formula and its negation has property \mathbf{Q}*
- ▶ **Induction steps:**
 - ▶ *If X has property \mathbf{Q} , so does $\neg\neg X$.*
 - ▶ *If α_1 and α_2 have property \mathbf{Q} , so does α .*
 - ▶ *If β_1 and β_2 have property \mathbf{Q} , so does β .*

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Induction 18

*Every propositional formula has a property, **Q**, provided:*

- ▶ **Basis step:** *Every atomic formula and its negation has property **Q***
- ▶ **Induction steps:**
 - ▶ *If X has property **Q**, so does $\neg\neg X$.*
 - ▶ *If α_1 and α_2 have property **Q**, so does α .*
 - ▶ *If β_1 and β_2 have property **Q**, so does β .*

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Induction 18

*Every propositional formula has a property, **Q**, provided:*

- ▶ **Basis step:** *Every atomic formula and its negation has property **Q***
- ▶ **Induction steps:**
 - ▶ *If X has property **Q**, so does $\neg\neg X$.*
 - ▶ *If α_1 and α_2 have property **Q**, so does α .*
 - ▶ *If β_1 and β_2 have property **Q**, so does β .*

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Induction 18

*Every propositional formula has a property, **Q**, provided:*

- ▶ **Basis step:** *Every atomic formula and its negation has property **Q***
- ▶ **Induction steps:**
 - ▶ *If X has property **Q**, so does $\neg\neg X$.*
 - ▶ *If α_1 and α_2 have property **Q**, so does α .*
 - ▶ *If β_1 and β_2 have property **Q**, so does β .*

Proof: ... exercise ...

Theorem — (Modified) Principle of Structural Recursion 19

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas and their negations.*
- ▶ **Induction steps:**
 - ▶ *The value of f for $\neg\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for α is specified in terms of the values of f for α_1 and α_2 .*
 - ▶ *The value of f for β is specified in terms of the values of f for β_1 and β_2 .*

Theorem — (Modified) Principle of Structural Recursion 19

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas and their negations.*
- ▶ **Induction steps:**
 - ▶ *The value of f for $\neg\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for α is specified in terms of the values of f for α_1 and α_2 .*
 - ▶ *The value of f for β is specified in terms of the values of f for β_1 and β_2 .*

Theorem — (Modified) Principle of Structural Recursion 19

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas and their negations.*
- ▶ **Induction steps:**
 - ▶ *The value of f for $\neg\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for α is specified in terms of the values of f for α_1 and α_2 .*
 - ▶ *The value of f for β is specified in terms of the values of f for β_1 and β_2 .*

Theorem — (Modified) Principle of Structural Recursion 19

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas and their negations.*
- ▶ **Induction steps:**
 - ▶ *The value of f for $\neg\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for α is specified in terms of the values of f for α_1 and α_2 .*
 - ▶ *The value of f for β is specified in terms of the values of f for β_1 and β_2 .*

Theorem — (Modified) Principle of Structural Recursion 19

There is one, and only one, function f defined on the set \mathbf{P} of propositional formulas such that:

- ▶ **Basis step:** *The value of f is specified explicitly on atomic formulas and their negations.*
- ▶ **Induction steps:**
 - ▶ *The value of f for $\neg\neg X$ is specified in terms of the value of f for X .*
 - ▶ *The value of f for α is specified in terms of the values of f for α_1 and α_2 .*
 - ▶ *The value of f for β is specified in terms of the values of f for β_1 and β_2 .*

Example — Rank-function r

20

$r(A) = r(\neg A) = 0$ for all $A \in \mathbf{L}$. $r(\perp) = r(\top) = 0$.

$r(\neg\top) = r(\neg\perp) = 1$. $r(\neg\neg Z) = r(Z) + 1$.

$r(\alpha) = r(\alpha_1) + r(\alpha_2) + 1$. $r(\beta) = r(\beta_1) + r(\beta_2) + 1$.

Example — Depth-function h

21

$h(A) = h(\neg A) = 0$ for all $A \in \mathbf{L}$. $h(\perp) = h(\top) = 0$.

$h(\neg\top) = h(\neg\perp) = 1$. $h(\neg\neg Z) = h(Z) + 1$.

$h(\alpha) = \max\{h(\alpha_1), h(\alpha_2)\} + 1$. $h(\beta) = \max\{h(\beta_1), h(\beta_2)\} + 1$.

Exercise: Compute the degree, rank and depth of

1. $(P \supset Q) \supset (Q \downarrow \neg R)$
2. $(\top \supset P) \supset Q$
3. $Q \supset (\top \supset P)$.

Propositional Logic: Uniform Notation

Example — Rank-function r

20

$r(A) = r(\neg A) = 0$ for all $A \in \mathbf{L}$. $r(\perp) = r(\top) = 0$.

$r(\neg\top) = r(\neg\perp) = 1$. $r(\neg\neg Z) = r(Z) + 1$.

$r(\alpha) = r(\alpha_1) + r(\alpha_2) + 1$. $r(\beta) = r(\beta_1) + r(\beta_2) + 1$.

Example — Depth-function h

21

$h(A) = h(\neg A) = 0$ for all $A \in \mathbf{L}$. $h(\perp) = h(\top) = 0$.

$h(\neg\top) = h(\neg\perp) = 1$. $h(\neg\neg Z) = h(Z) + 1$.

$h(\alpha) = \max\{h(\alpha_1), h(\alpha_2)\} + 1$. $h(\beta) = \max\{h(\beta_1), h(\beta_2)\} + 1$.

Exercise: Compute the degree, rank and depth of

1. $(P \supset Q) \supset (Q \downarrow \neg R)$
2. $(\top \supset P) \supset Q$
3. $Q \supset (\top \supset P)$.

Propositional Logic: Uniform Notation

Example — Rank-function r

20

$r(A) = r(\neg A) = 0$ for all $A \in \mathbf{L}$. $r(\perp) = r(\top) = 0$.

$r(\neg\top) = r(\neg\perp) = 1$. $r(\neg\neg Z) = r(Z) + 1$.

$r(\alpha) = r(\alpha_1) + r(\alpha_2) + 1$. $r(\beta) = r(\beta_1) + r(\beta_2) + 1$.

Example — Depth-function h

21

$h(A) = h(\neg A) = 0$ for all $A \in \mathbf{L}$. $h(\perp) = h(\top) = 0$.

$h(\neg\top) = h(\neg\perp) = 1$. $h(\neg\neg Z) = h(Z) + 1$.

$h(\alpha) = \max\{h(\alpha_1), h(\alpha_2)\} + 1$. $h(\beta) = \max\{h(\beta_1), h(\beta_2)\} + 1$.

Exercise: Compute the degree, rank and depth of

1. $(P \supset Q) \supset (Q \downarrow \neg R)$
2. $(\top \supset P) \supset Q$
3. $Q \supset (\top \supset P)$.

Propositional Logic: Uniform Notation

Example — Rank-function r

20

$r(A) = r(\neg A) = 0$ for all $A \in \mathbf{L}$. $r(\perp) = r(\top) = 0$.

$r(\neg\top) = r(\neg\perp) = 1$. $r(\neg\neg Z) = r(Z) + 1$.

$r(\alpha) = r(\alpha_1) + r(\alpha_2) + 1$. $r(\beta) = r(\beta_1) + r(\beta_2) + 1$.

Example — Depth-function h

21

$h(A) = h(\neg A) = 0$ for all $A \in \mathbf{L}$. $h(\perp) = h(\top) = 0$.

$h(\neg\top) = h(\neg\perp) = 1$. $h(\neg\neg Z) = h(Z) + 1$.

$h(\alpha) = \max\{h(\alpha_1), h(\alpha_2)\} + 1$. $h(\beta) = \max\{h(\beta_1), h(\beta_2)\} + 1$.

Exercise: Compute the degree, rank and depth of

1. $(P \supset Q) \supset (Q \downarrow \neg R)$
2. $(\top \supset P) \supset Q$
3. $Q \supset (\top \supset P)$.

Propositional Logic: Uniform Notation

Example — Rank-function r

20

$r(A) = r(\neg A) = 0$ for all $A \in \mathbf{L}$. $r(\perp) = r(\top) = 0$.

$r(\neg\top) = r(\neg\perp) = 1$. $r(\neg\neg Z) = r(Z) + 1$.

$r(\alpha) = r(\alpha_1) + r(\alpha_2) + 1$. $r(\beta) = r(\beta_1) + r(\beta_2) + 1$.

Example — Depth-function h

21

$h(A) = h(\neg A) = 0$ for all $A \in \mathbf{L}$. $h(\perp) = h(\top) = 0$.

$h(\neg\top) = h(\neg\perp) = 1$. $h(\neg\neg Z) = h(Z) + 1$.

$h(\alpha) = \max\{h(\alpha_1), h(\alpha_2)\} + 1$. $h(\beta) = \max\{h(\beta_1), h(\beta_2)\} + 1$.

Exercise: Compute the degree, rank and depth of

1. $(P \supset Q) \supset (Q \downarrow \neg R)$
2. $(\top \supset P) \supset Q$
3. $Q \supset (\top \supset P)$.

Example — Rank-function r

20

$r(A) = r(\neg A) = 0$ for all $A \in \mathbf{L}$. $r(\perp) = r(\top) = 0$.

$r(\neg\top) = r(\neg\perp) = 1$. $r(\neg\neg Z) = r(Z) + 1$.

$r(\alpha) = r(\alpha_1) + r(\alpha_2) + 1$. $r(\beta) = r(\beta_1) + r(\beta_2) + 1$.

Example — Depth-function h

21

$h(A) = h(\neg A) = 0$ for all $A \in \mathbf{L}$. $h(\perp) = h(\top) = 0$.

$h(\neg\top) = h(\neg\perp) = 1$. $h(\neg\neg Z) = h(Z) + 1$.

$h(\alpha) = \max\{h(\alpha_1), h(\alpha_2)\} + 1$. $h(\beta) = \max\{h(\beta_1), h(\beta_2)\} + 1$.

Exercise: Compute the degree, rank and depth of

1. $(P \supset Q) \supset (Q \downarrow \neg R)$
2. $(\top \supset P) \supset Q$
3. $Q \supset (\top \supset P)$.

Example — Rank-function r

20

$r(A) = r(\neg A) = 0$ for all $A \in \mathbf{L}$. $r(\perp) = r(\top) = 0$.

$r(\neg\top) = r(\neg\perp) = 1$. $r(\neg\neg Z) = r(Z) + 1$.

$r(\alpha) = r(\alpha_1) + r(\alpha_2) + 1$. $r(\beta) = r(\beta_1) + r(\beta_2) + 1$.

Example — Depth-function h

21

$h(A) = h(\neg A) = 0$ for all $A \in \mathbf{L}$. $h(\perp) = h(\top) = 0$.

$h(\neg\top) = h(\neg\perp) = 1$. $h(\neg\neg Z) = h(Z) + 1$.

$h(\alpha) = \max\{h(\alpha_1), h(\alpha_2)\} + 1$. $h(\beta) = \max\{h(\beta_1), h(\beta_2)\} + 1$.

Exercise: Compute the degree, rank and depth of

1. $(P \supset Q) \supset (Q \downarrow \neg R)$
2. $(\top \supset P) \supset Q$
3. $Q \supset (\top \supset P)$.

Definition — Generalized Disjunction/Conjunction

22

We write

- ▶ $[X_1, X_2, \dots, X_n]$ for $X_1 \vee X_2 \vee \dots \vee X_n$
- ▶ $\langle X_1, X_2, \dots, X_n \rangle$ for $X_1 \wedge X_2 \wedge \dots \wedge X_n$

We have

- ▶ $v([X_1, X_2, \dots, X_n]) = t$ if for some $X_i (0 < i \leq n)$ we have $v(X_i) = t$; otherwise $v([X_1, X_2, \dots, X_n]) = f$
- ▶ $v(\langle X_1, X_2, \dots, X_n \rangle) = t$ if for every $X_i (0 < i \leq n)$ we have $v(X_i) = t$; otherwise $v(\langle X_1, X_2, \dots, X_n \rangle) = f$

Note: The following are tautologies:

- ▶ $\emptyset \equiv \perp$ and $\langle \rangle \equiv \top$
- ▶ $[X] \equiv X$ and $\langle X \rangle \equiv X$
- ▶ $[X, Y] \equiv (X \vee Y)$ and $\langle X, Y \rangle \equiv X \wedge Y$

Definition — Generalized Disjunction/Conjunction

22

We write

- ▶ $[X_1, X_2, \dots, X_n]$ for $X_1 \vee X_2 \vee \dots \vee X_n$
- ▶ $\langle X_1, X_2, \dots, X_n \rangle$ for $X_1 \wedge X_2 \wedge \dots \wedge X_n$

We have

- ▶ $v([X_1, X_2, \dots, X_n]) = t$ if for some $X_i (0 < i \leq n)$ we have $v(X_i) = t$; otherwise $v([X_1, X_2, \dots, X_n]) = f$
- ▶ $v(\langle X_1, X_2, \dots, X_n \rangle) = t$ if for every $X_i (0 < i \leq n)$ we have $v(X_i) = t$; otherwise $v(\langle X_1, X_2, \dots, X_n \rangle) = f$

Note: The following are tautologies:

- ▶ $\emptyset \equiv \perp$ and $\langle \rangle \equiv \top$
- ▶ $[X] \equiv X$ and $\langle X \rangle \equiv X$
- ▶ $[X, Y] \equiv (X \vee Y)$ and $\langle X, Y \rangle \equiv X \wedge Y$

Definition — Generalized Disjunction/Conjunction

22

We write

- ▶ $[X_1, X_2, \dots, X_n]$ for $X_1 \vee X_2 \vee \dots \vee X_n$
- ▶ $\langle X_1, X_2, \dots, X_n \rangle$ for $X_1 \wedge X_2 \wedge \dots \wedge X_n$

We have

- ▶ $v([X_1, X_2, \dots, X_n]) = t$ if for some $X_i (0 < i \leq n)$ we have $v(X_i) = t$; otherwise $v([X_1, X_2, \dots, X_n]) = f$
- ▶ $v(\langle X_1, X_2, \dots, X_n \rangle) = t$ if for every $X_i (0 < i \leq n)$ we have $v(X_i) = t$; otherwise $v(\langle X_1, X_2, \dots, X_n \rangle) = f$

Note: The following are tautologies:

- ▶ $[] \equiv \perp$ and $\langle \rangle \equiv \top$
- ▶ $[X] \equiv X$ and $\langle X \rangle \equiv X$
- ▶ $[X, Y] \equiv (X \vee Y)$ and $\langle X, Y \rangle \equiv X \wedge Y$

Propositional Logic: Normal Forms

Definition — Literal

23

Let $P \in \mathbf{L}$ be propositional letter. Then P and $\neg P$ are called *literals*. Moreover, \perp and \top are called *literals*.

Definition — Clause / Dual Clause

24

Let $X_i (0 < i \leq n)$ be literals. Then the disjunction $[X_1, X_2, \dots, X_n]$ is called a *clause* and the conjunction $\langle X_1, X_2, \dots, X_n \rangle$ is called a *dual clause*.

Example

25

... blackboard ...

Definition — Literal

23

Let $P \in \mathbf{L}$ be propositional letter. Then P and $\neg P$ are called *literals*. Moreover, \perp and \top are called *literals*.

Definition — Clause / Dual Clause

24

Let $X_i (0 < i \leq n)$ be literals. Then the disjunction $[X_1, X_2, \dots, X_n]$ is called a *clause* and the conjunction $\langle X_1, X_2, \dots, X_n \rangle$ is called a *dual clause*.

Example

25

... blackboard ...

Definition — Literal

23

Let $P \in \mathbf{L}$ be propositional letter. Then P and $\neg P$ are called *literals*. Moreover, \perp and \top are called *literals*.

Definition — Clause / Dual Clause

24

Let $X_i (0 < i \leq n)$ be literals. Then the disjunction $[X_1, X_2, \dots, X_n]$ is called a *clause* and the conjunction $\langle X_1, X_2, \dots, X_n \rangle$ is called a *dual clause*.

Example

25

... blackboard ...

Propositional Logic: Normal Forms

Definition — Conjunctive Normal Form

26

The formula $\langle C_1, C_2, \dots, C_n \rangle$ is in *conjunctive normal form* if all formulas $C_i (0 < i \leq n)$ are clauses. (Alternatively, we say $\langle C_1, C_2, \dots, C_n \rangle$ is in *clause form* or a *clause set*.)

Definition — Disjunctive Normal Form

27

The formula $[C_1, C_2, \dots, C_n]$ is in *disjunctive normal form* if all formulas $C_i (0 < i \leq n)$ are dual clauses. (Alternatively, we say $[C_1, C_2, \dots, C_n]$ is in *dual clause form* or a *dual clause set*.)

Example — Normal Forms

28

- ▶ $\langle [P, \neg Q], [\top, R, \neg S], [] \rangle$

... blackboard ...

Propositional Logic: Normal Forms

Definition — Conjunctive Normal Form

26

The formula $\langle C_1, C_2, \dots, C_n \rangle$ is in *conjunctive normal form* if all formulas $C_i (0 < i \leq n)$ are clauses. (Alternatively, we say $\langle C_1, C_2, \dots, C_n \rangle$ is in *clause form* or a *clause set*.)

Definition — Disjunctive Normal Form

27

The formula $[C_1, C_2, \dots, C_n]$ is in *disjunctive normal form* if all formulas $C_i (0 < i \leq n)$ are dual clauses. (Alternatively, we say $[C_1, C_2, \dots, C_n]$ is in *dual clause form* or a *dual clause set*.)

Example — Normal Forms

28

- ▶ $\langle [P, \neg Q], [\top, R, \neg S], [] \rangle$

... blackboard ...

Propositional Logic: Normal Forms

Definition — Conjunctive Normal Form

26

The formula $\langle C_1, C_2, \dots, C_n \rangle$ is in *conjunctive normal form* if all formulas $C_i (0 < i \leq n)$ are clauses. (Alternatively, we say $\langle C_1, C_2, \dots, C_n \rangle$ is in *clause form* or a *clause set*.)

Definition — Disjunctive Normal Form

27

The formula $[C_1, C_2, \dots, C_n]$ is in *disjunctive normal form* if all formulas $C_i (0 < i \leq n)$ are dual clauses. (Alternatively, we say $[C_1, C_2, \dots, C_n]$ is in *dual clause form* or a *dual clause set*.)

Example — Normal Forms

28

- ▶ $\langle [P, \neg Q], [\top, R, \neg S], [] \rangle$

... blackboard ...

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- if N is $\neg T$ replace N with \perp
- if N is $\neg \perp$ replace N with T
- if N is $\neg \neg Z$ replace N with Z
- if N is a β -formula, replace N with the two formula sequence β_1, β_2
- if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- if N is $\neg T$ replace N with \perp
- if N is $\neg \perp$ replace N with T
- if N is $\neg \neg Z$ replace N with Z
- if N is a β -formula, replace N with the two formula sequence β_1, β_2
- if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- if N is $\neg T$ replace N with \perp
- if N is $\neg \perp$ replace N with T
- if N is $\neg \neg Z$ replace N with Z
- if N is a β -formula, replace N with the two formula sequence β_1, β_2
- if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg T$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with T
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a β -formula, replace N with the two formula sequence β_1, β_2
- ▶ if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg T$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with T
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a β -formula, replace N with the two formula sequence β_1, β_2
- ▶ if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg\top$ replace N with \perp
- ▶ if N is $\neg\perp$ replace N with \top
- ▶ if N is $\neg\neg Z$ replace N with Z
- ▶ if N is a β -formula, replace N with the two formula sequence β_1, β_2
- ▶ if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg T$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with T
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a β -formula, replace N with the two formula sequence β_1, β_2
- ▶ if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg\top$ replace N with \perp
- ▶ if N is $\neg\perp$ replace N with \top
- ▶ if N is $\neg\neg Z$ replace N with Z
- ▶ if N is a β -formula, replace N with the two formula sequence β_1, β_2
- ▶ if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg\top$ replace N with \perp
- ▶ if N is $\neg\perp$ replace N with \top
- ▶ if N is $\neg\neg Z$ replace N with Z
- ▶ if N is a β -formula, replace N with the two formula sequence β_1, β_2
- ▶ if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Theorem — Normal Form (Clause Form)

29

There is an algorithm for converting each $X \in \mathbf{P}$ into clause form.

Algorithm TransformInClauseForm:

Input: $S = \langle D_1, \dots, D_n \rangle$ with $D_i = [X_{i_1}, \dots, X_{i_k}]$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $\langle [X] \rangle$)

Output: $S' = \langle C_1, \dots, C_m \rangle$ with $C_i = [Y_{i_1}, \dots, Y_{i_l}]$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg\top$ replace N with \perp
- ▶ if N is $\neg\perp$ replace N with \top
- ▶ if N is $\neg\neg Z$ replace N with Z
- ▶ if N is a β -formula, replace N with the two formula sequence β_1, β_2
- ▶ if N is an α -formula, replace D_i with two disjunctions, one like D_i but with α replaced by α_1 and one like D_i but with α replaced by α_2 .

Recurse on modified S .

Definition — Clause Set Reduction Rules

Input: $\langle D_1, \dots, D_n \rangle$

conjunction of disjunctions

$$\frac{\langle D_1, \dots, [\dots, \neg\neg Z, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, Z, \dots], \dots, D_n \rangle} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{\langle D_1, \dots, [\dots, \neg T, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, T, \dots], \dots, D_n \rangle} \quad \frac{\neg T}{T} \quad \text{analogous for } \frac{\neg \perp}{\top}$$

$$\frac{\langle D_1, \dots, [\dots, \beta, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \beta_1, \beta_2, \dots], \dots, D_n \rangle} \quad \frac{\beta}{\beta_1} \quad \frac{\beta}{\beta_2}$$

$$\frac{\langle D_1, \dots, [\dots, \alpha, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \alpha_1, \dots], [\dots, \alpha_2, \dots], \dots, D_n \rangle} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Definition — Clause Set Reduction Rules

Input: $\langle D_1, \dots, D_n \rangle$

conjunction of disjunctions

$$\frac{\langle D_1, \dots, [\dots, \neg\neg Z, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, Z, \dots], \dots, D_n \rangle} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{\langle D_1, \dots, [\dots, \neg T, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \top, \dots], \dots, D_n \rangle} \quad \frac{\neg T}{\top}$$

analogous for $\frac{\neg\perp}{\top}$

$$\frac{\langle D_1, \dots, [\dots, \beta, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \beta_1, \beta_2, \dots], \dots, D_n \rangle} \quad \frac{\beta}{\beta_1} \quad \frac{\beta}{\beta_2}$$

$$\frac{\langle D_1, \dots, [\dots, \alpha, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \alpha_1, \dots], [\dots, \alpha_2, \dots], \dots, D_n \rangle} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Definition — Clause Set Reduction Rules

Input: $\langle D_1, \dots, D_n \rangle$

conjunction of disjunctions

$$\frac{\langle D_1, \dots, [\dots, \neg\neg Z, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, Z, \dots], \dots, D_n \rangle} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{\langle D_1, \dots, [\dots, \neg\top, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \perp, \dots], \dots, D_n \rangle} \quad \frac{\neg\top}{\perp}$$

analogous for $\frac{\neg\perp}{\top}$

$$\frac{\langle D_1, \dots, [\dots, \beta, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \beta_1, \beta_2, \dots], \dots, D_n \rangle} \quad \frac{\beta}{\beta_1} \quad \frac{\beta}{\beta_2}$$

$$\frac{\langle D_1, \dots, [\dots, \alpha, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \alpha_1, \dots], [\dots, \alpha_2, \dots], \dots, D_n \rangle} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Definition — Clause Set Reduction Rules

Input: $\langle D_1, \dots, D_n \rangle$ conjunction of disjunctions

$$\frac{\langle D_1, \dots, [\dots, \neg\neg Z, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, Z, \dots], \dots, D_n \rangle} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{\langle D_1, \dots, [\dots, \neg T, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \perp, \dots], \dots, D_n \rangle} \quad \frac{\neg T}{\perp} \quad \text{analogous for } \frac{\neg \perp}{T}$$

$$\frac{\langle D_1, \dots, [\dots, \beta, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \beta_1, \beta_2, \dots], \dots, D_n \rangle} \quad \frac{\beta}{\beta_1} \quad \frac{\beta}{\beta_2}$$

$$\frac{\langle D_1, \dots, [\dots, \alpha, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \alpha_1, \dots], [\dots, \alpha_2, \dots], \dots, D_n \rangle} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Definition — Clause Set Reduction Rules

Input: $\langle D_1, \dots, D_n \rangle$ conjunction of disjunctions

$$\frac{\langle D_1, \dots, [\dots, \neg\neg Z, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, Z, \dots], \dots, D_n \rangle} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{\langle D_1, \dots, [\dots, \neg T, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \perp, \dots], \dots, D_n \rangle} \quad \frac{\neg T}{\perp} \quad \text{analogous for } \frac{\neg \perp}{T}$$

$$\frac{\langle D_1, \dots, [\dots, \beta, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \beta_1, \beta_2, \dots], \dots, D_n \rangle} \quad \frac{\beta}{\beta_1} \quad \frac{\beta}{\beta_2}$$

$$\frac{\langle D_1, \dots, [\dots, \alpha, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \alpha_1, \dots], [\dots, \alpha_2, \dots], \dots, D_n \rangle} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Definition — Clause Set Reduction Rules

Input: $\langle D_1, \dots, D_n \rangle$ conjunction of disjunctions

$$\frac{\langle D_1, \dots, [\dots, \neg\neg Z, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, Z, \dots], \dots, D_n \rangle} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{\langle D_1, \dots, [\dots, \neg\top, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \perp, \dots], \dots, D_n \rangle} \quad \frac{\neg\top}{\perp} \quad \text{analogous for } \frac{\neg\perp}{\top}$$

$$\frac{\langle D_1, \dots, [\dots, \beta, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \beta_1, \beta_2, \dots], \dots, D_n \rangle} \quad \frac{\beta}{\beta_1} \quad \frac{\beta}{\beta_2}$$

$$\frac{\langle D_1, \dots, [\dots, \alpha, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \alpha_1, \dots], [\dots, \alpha_2, \dots], \dots, D_n \rangle} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Lemma — Conjunctive Rewrite

31

If S is a conjunction of disjunctions, and one of the clause set reduction rules is applied to S , producing S^ then $S \equiv S^*$ is a tautology.*

Proof: Follows from $\neg T \equiv \perp$, $\neg \perp \equiv T$, $\neg\neg Z = Z$, $\beta \equiv \beta_1 \vee \beta_2$, $\alpha \equiv \alpha_1 \wedge \alpha_2$ and the Replacement Theorem.

Lemma — Conjunctive Rewrite

31

If S is a conjunction of disjunctions, and one of the clause set reduction rules is applied to S , producing S^ then $S \equiv S^*$ is a tautology.*

Proof: Follows from $\neg T \equiv \perp$, $\neg \perp \equiv T$, $\neg\neg Z = Z$, $\beta \equiv \beta_1 \vee \beta_2$, $\alpha \equiv \alpha_1 \wedge \alpha_2$ and the Replacement Theorem.

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- if N is $\neg T$ replace N with \perp
- if N is $\neg \perp$ replace N with T
- if N is $\neg \neg Z$ replace N with Z
- if N is a α -formula, replace N with the two formula sequence α_1, α_2
- if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- if N is $\neg T$ replace N with \perp
- if N is $\neg \perp$ replace N with T
- if N is $\neg \neg Z$ replace N with Z
- if N is a α -formula, replace N with the two formula sequence α_1, α_2
- if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg \top$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with \top
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a α -formula, replace N with the two formula sequence α_1, α_2
- ▶ if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg \top$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with \top
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a α -formula, replace N with the two formula sequence α_1, α_2
- ▶ if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg \top$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with \top
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a α -formula, replace N with the two formula sequence α_1, α_2
- ▶ if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg \top$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with \top
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a α -formula, replace N with the two formula sequence α_1, α_2
- ▶ if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg \top$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with \top
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a α -formula, replace N with the two formula sequence α_1, α_2
- ▶ if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg \top$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with \top
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a α -formula, replace N with the two formula sequence α_1, α_2
- ▶ if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Theorem — Normal Form (Dual Clause Form)

32

There is an algorithm for converting each formula $X \in \mathbf{P}$ into dual clause form.

Algorithm TransformInDualClauseForm:

Input: $S = [D_1, \dots, D_n]$ with $D_i = \langle X_{i_1}, \dots, X_{i_k} \rangle$ and $X_{i_1}, \dots, X_{i_k} \in \mathbf{P}$ (initially we start with $[\langle X \rangle]$)

Output: $[C_1, \dots, C_m]$ with $C_i = \langle Y_{i_1}, \dots, Y_{i_l} \rangle$ where Y_{i_1}, \dots, Y_{i_l} are literals.

If all X_{i_z} in S are literals (i.e. all D_i are dual clauses) then return S
else select D_i with non-literal $N = X_{i_z}$.

- ▶ if N is $\neg \top$ replace N with \perp
- ▶ if N is $\neg \perp$ replace N with \top
- ▶ if N is $\neg \neg Z$ replace N with Z
- ▶ if N is a α -formula, replace N with the two formula sequence α_1, α_2
- ▶ if N is an β -formula, replace D_i with two disjunctions, one like D_i but with β replaced by β_1 and one like D_i but with β replaced by β_2 .

Recurse on modified S .

Definition — Dual Clause Set Reduction Rules

Input: $[D_1, \dots, D_n]$

disjunction of conjunctions

$$\frac{[D_1, \dots, \langle \dots, \neg\neg Z, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, Z, \dots \rangle, \dots, D_n]} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{[D_1, \dots, \langle \dots, \neg\top, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \perp, \dots \rangle, \dots, D_n]} \quad \frac{\neg\top}{\perp}$$

analogous for $\frac{\neg\perp}{\top}$

$$\frac{[D_1, \dots, \langle \dots, \alpha, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \alpha_1, \alpha_2, \dots \rangle, \dots, D_n]} \quad \frac{\frac{\alpha}{\alpha_1}}{\alpha_2}$$

$$\frac{[D_1, \dots, \langle \dots, \beta, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \beta_1, \dots \rangle, \langle \dots, \beta_2, \dots \rangle, \dots, D_n]} \quad \frac{\beta}{\beta_1 \mid \beta_2}$$

Definition — Dual Clause Set Reduction Rules

Input: $[D_1, \dots, D_n]$

disjunction of conjunctions

$$\frac{[D_1, \dots, \langle \dots, \neg\neg Z, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, Z, \dots \rangle, \dots, D_n]} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{[D_1, \dots, \langle \dots, \neg\top, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \perp, \dots \rangle, \dots, D_n]} \quad \frac{\neg\top}{\perp} \quad \text{analogous for } \frac{\neg\perp}{\top}$$

$$\frac{[D_1, \dots, \langle \dots, \alpha, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \alpha_1, \alpha_2, \dots \rangle, \dots, D_n]} \quad \frac{\frac{\alpha}{\alpha_1}}{\alpha_2}$$

$$\frac{[D_1, \dots, \langle \dots, \beta, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \beta_1, \dots \rangle, \langle \dots, \beta_2, \dots \rangle, \dots, D_n]} \quad \frac{\beta}{\beta_1 \mid \beta_2}$$

Lemma — Disjunctive Rewrite

34

If S is a disjunction of conjunctions, and one of the dual clause set reduction rules is applied to S , producing S^ then $S \equiv S^*$ is a tautology.*

Proof: Follows from $\neg T \equiv \perp$, $\neg \perp \equiv T$, $\neg\neg Z = Z$, $\beta \equiv \beta_1 \vee \beta_2$, $\alpha \equiv \alpha_1 \wedge \alpha_2$ and the Replacement Theorem.

Lemma — Disjunctive Rewrite

34

If S is a disjunction of conjunctions, and one of the dual clause set reduction rules is applied to S , producing S^ then $S \equiv S^*$ is a tautology.*

Proof: Follows from $\neg T \equiv \perp$, $\neg \perp \equiv T$, $\neg\neg Z = Z$, $\beta \equiv \beta_1 \vee \beta_2$, $\alpha \equiv \alpha_1 \wedge \alpha_2$ and the Replacement Theorem.

Semantic Tableaux & Resolution

- ▶ Refutation systems:
To prove X , we assume $\neg X$ and produce a contradiction
- ▶ We subsequently expand $\neg X$ (in the sense of the rules seen before) and reveal its internal logical structure
- ▶ Resolution works on conjunctive normal form, i.e. clause sets
- ▶ Tableaux works on disjunctive normal form, i.e. dual clause sets; they can be conveniently be displayed as trees

Sets of formulas: $\langle X_1, \dots, X_n \rangle$ was introduced as a conjunctive list of formulas. In the following we often use the more general notion $\{X_1, \dots, X_n\}$ of conjunctive (finite) sets of formulas. That is we implicitly treat idempotency, associativity and commutativity of \wedge . This added generality eases some proofs later.

Semantic Tableaux & Resolution

- ▶ Refutation systems:
To prove X , we assume $\neg X$ and produce a contradiction
- ▶ We subsequently expand $\neg X$ (in the sense of the rules seen before) and reveal its internal logical structure
- ▶ Resolution works on conjunctive normal form, i.e. clause sets
- ▶ Tableaux works on disjunctive normal form, i.e. dual clause sets; they can be conveniently be displayed as trees

Sets of formulas: $\langle X_1, \dots, X_n \rangle$ was introduced as a conjunctive list of formulas. In the following we often use the more general notion $\{X_1, \dots, X_n\}$ of conjunctive (finite) sets of formulas. That is we implicitly treat idempotency, associativity and commutativity of \wedge . This added generality eases some proofs later.

Semantic Tableaux & Resolution

- ▶ Refutation systems:
 - To prove X , we assume $\neg X$ and produce a contradiction
- ▶ We subsequently expand $\neg X$ (in the sense of the rules seen before) and reveal its internal logical structure
- ▶ Resolution works on conjunctive normal form, i.e. clause sets
- ▶ Tableaux works on disjunctive normal form, i.e. dual clause sets; they can be conveniently be displayed as trees

Sets of formulas: $\langle X_1, \dots, X_n \rangle$ was introduced as a conjunctive list of formulas. In the following we often use the more general notion $\{X_1, \dots, X_n\}$ of conjunctive (finite) sets of formulas. That is we implicitly treat idempotency, associativity and commutativity of \wedge . This added generality eases some proofs later.

Semantic Tableaux & Resolution

- ▶ Refutation systems:
To prove X , we assume $\neg X$ and produce a contradiction
- ▶ We subsequently expand $\neg X$ (in the sense of the rules seen before) and reveal its internal logical structure
- ▶ Resolution works on conjunctive normal form, i.e. clause sets
- ▶ Tableaux works on disjunctive normal form, i.e. dual clause sets; they can be conveniently be displayed as trees

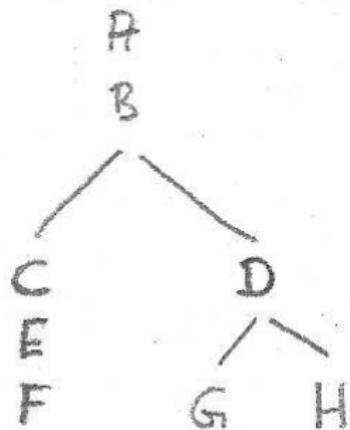
Sets of formulas: $\langle X_1, \dots, X_n \rangle$ was introduced as a conjunctive list of formulas. In the following we often use the more general notion $\{X_1, \dots, X_n\}$ of conjunctive (finite) sets of formulas. That is we implicitly treat idempotency, associativity and commutativity of \wedge . This added generality eases some proofs later.

Semantic Tableaux & Resolution

- ▶ Refutation systems:
To prove X , we assume $\neg X$ and produce a contradiction
- ▶ We subsequently expand $\neg X$ (in the sense of the rules seen before) and reveal its internal logical structure
- ▶ Resolution works on conjunctive normal form, i.e. clause sets
- ▶ Tableaux works on disjunctive normal form, i.e. dual clause sets; they can be conveniently be displayed as trees

Sets of formulas: $\langle X_1, \dots, X_n \rangle$ was introduced as a conjunctive **list** of formulas. In the following we often use the more general notion $\{X_1, \dots, X_n\}$ of conjunctive (finite) sets of formulas. That is we implicitly treat idempotency, associativity and commutativity of \wedge . This added generality eases some proofs later.

Tree Representation



$\langle A, B, \neg \langle \neg C, \neg E, \neg F \rangle, \neg D, \neg \langle \neg G, \neg H \rangle \rangle$

$$A \wedge B \wedge (\neg C \wedge \neg E \wedge \neg F) \vee \neg D \wedge \neg (G \vee H)$$

Remember: Tableau works of disjunction of conjunctions

Input: $[D_1, \dots, D_n]$

$$\frac{[D_1, \dots, \langle \dots, \neg\neg Z, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, Z, \dots \rangle, \dots, D_n]} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{[D_1, \dots, \langle \dots, \neg\top, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \perp, \dots \rangle, \dots, D_n]} \quad \frac{\neg\top}{\perp}$$

analogous for $\frac{\neg\perp}{\top}$

$$\frac{[D_1, \dots, \langle \dots, \alpha, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \alpha_1, \alpha_2, \dots \rangle, \dots, D_n]} \quad \frac{\alpha}{\alpha_1} \quad \frac{}{\alpha_2}$$

$$\frac{[D_1, \dots, \langle \dots, \beta, \dots \rangle, \dots, D_n]}{[D_1, \dots, \langle \dots, \beta_1, \dots \rangle, \langle \dots, \beta_2, \dots \rangle, \dots, D_n]} \quad \frac{\beta}{\beta_1 \mid \beta_2}$$

Definition — Tableau Expansion Rules

35

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\alpha}{\alpha_1 \quad \alpha_2} \quad \frac{\beta}{\beta_1 \mid \beta_2}$$

Definition — Tableau Expansion

36

The one-branch tree $\begin{array}{c} A_1 \\ \vdots \\ A_n \end{array}$ is a *tableau* for $\{A_1, \dots, A_n\}$

If T is a tableau for $\{A_1, \dots, A_n\}$ and T^* results from T by the application of a Tableau Expansion Rule, then T^* is a *tableau (expansion)* for $\{A_1, \dots, A_n\}$

Definition — Tableau Expansion Rules

35

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\alpha_1}{\alpha_1} \quad \frac{\beta}{\beta_1 \mid \beta_2}$$

Definition — Tableau Expansion

36

 A_1

The one-branch tree \vdash is a *tableau* for $\{A_1, \dots, A_n\}$

 A_n

If T is a tableau for $\{A_1, \dots, A_n\}$ and T^* results from T by the application of a Tableau Expansion Rule, then T^* is a *tableau expansion* for $\{A_1, \dots, A_n\}$

Definition — Tableau Expansion Rules

35

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\alpha}{\alpha_1 \quad \alpha_2} \quad \frac{\beta}{\beta_1 \mid \beta_2}$$

Definition — Tableau Expansion

36

 A_1

The one-branch tree \vdash is a *tableau* for $\{A_1, \dots, A_n\}$

 A_n

If T is a tableau for $\{A_1, \dots, A_n\}$ and T^* results from T by the application of a Tableau Expansion Rule, then T^* is a *tableau (expansion)* for $\{A_1, \dots, A_n\}$

Definition — Closed Branch and Closed Tableau

37

A branch θ of a tableau is called *closed* if X and $\neg X$ occur on θ for some formula $X \in \mathbf{P}$, or if \perp occurs on θ .

A branch θ of a tableau is called *atomically closed* if A and $\neg A$ occur on θ for some atomic formula A , or if \perp occurs on θ .

A *tableau is (atomically) closed* if every branch is (atomically) closed.

Definition — Tableau Proof

38

A *tableaux proof* of formula $X \in \mathbf{P}$ is a closed tableau expansion for $\{\neg X\}$. X is a *theorem* of the tableau system if X has a tableau proof. We write: $\vdash_{pt} X$.

Definition — Closed Branch and Closed Tableau

37

A branch θ of a tableau is called *closed* if X and $\neg X$ occur on θ for some formula $X \in \mathbf{P}$, or if \perp occurs on θ .

A branch θ of a tableau is called *atomically closed* if A and $\neg A$ occur on θ for some atomic formula A , or if \perp occurs on θ .

A *tableau is (atomically) closed* if every branch is (atomically) closed.

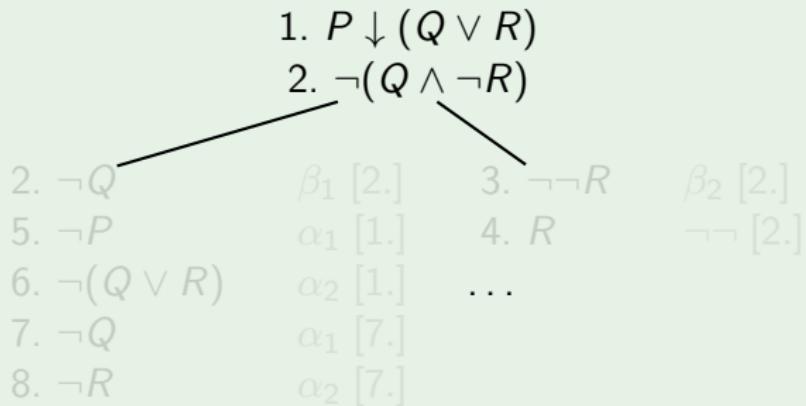
Definition — Tableau Proof

38

A *tableaux proof* of formula $X \in \mathbf{P}$ is a closed tableau expansion for $\{\neg X\}$. X is a *theorem* of the tableau system if X has a tableau proof. We write: $\vdash_{pt} X$.

Example — Tableau for $\{P \downarrow (Q \vee R), \neg(Q \wedge \neg R)\}$

39

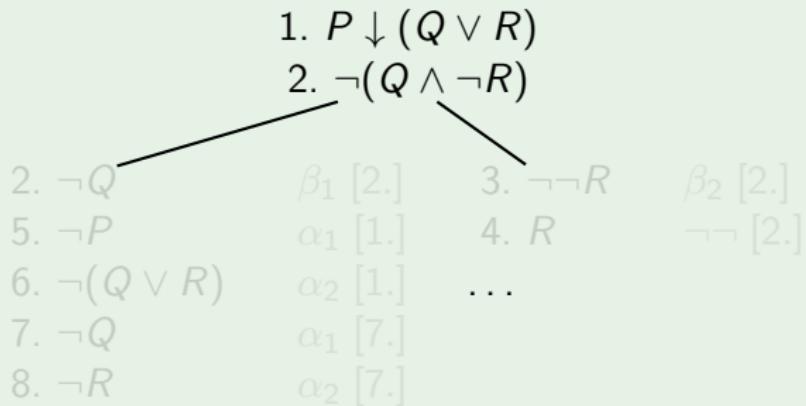
Example — Tableau Proof for
 $(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$

40

... on blackboard ...

Example — Tableau for $\{P \downarrow (Q \vee R), \neg(Q \wedge \neg R)\}$

39

Example — Tableau Proof for
 $(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$

40

... on blackboard ...

Example — Tableau for $\{P \downarrow (Q \vee R), \neg(Q \wedge \neg R)\}$

39

	1. $P \downarrow (Q \vee R)$			
	2. $\neg(Q \wedge \neg R)$			
2. $\neg Q$	$\beta_1 [2.]$	3. $\neg\neg R$	$\beta_2 [2.]$	
5. $\neg P$	$\alpha_1 [1.]$	4. R	$\neg\neg [2.]$	
6. $\neg(Q \vee R)$	$\alpha_2 [1.]$...		
7. $\neg Q$	$\alpha_1 [7.]$			
8. $\neg R$	$\alpha_2 [7.]$			

Example — Tableau Proof for
 $(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$

40

... on blackboard ...

Example — Tableau for $\{P \downarrow (Q \vee R), \neg(Q \wedge \neg R)\}$

39

	1. $P \downarrow (Q \vee R)$			
	2. $\neg(Q \wedge \neg R)$			
2. $\neg Q$	$\beta_1 [2.]$	3. $\neg\neg R$	$\beta_2 [2.]$	
5. $\neg P$	$\alpha_1 [1.]$	4. R	$\neg\neg [2.]$	
6. $\neg(Q \vee R)$	$\alpha_2 [1.]$...		
7. $\neg Q$	$\alpha_1 [7.]$			
8. $\neg R$	$\alpha_2 [7.]$			

Example — Tableau Proof for
 $(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$

40

... on blackboard ...

Example — Tableau for $\{P \downarrow (Q \vee R), \neg(Q \wedge \neg R)\}$

39

	1. $P \downarrow (Q \vee R)$			
	2. $\neg(Q \wedge \neg R)$			
2. $\neg Q$	$\beta_1 [2.]$	3. $\neg\neg R$	$\beta_2 [2.]$	
5. $\neg P$	$\alpha_1 [1.]$	4. R	$\neg\neg [2.]$	
6. $\neg(Q \vee R)$	$\alpha_2 [1.]$...		
7. $\neg Q$	$\alpha_1 [7.]$			
8. $\neg R$	$\alpha_2 [7.]$			

Example — Tableau Proof for
 $(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$

40

... on blackboard ...

Example — Tableau for $\{P \downarrow (Q \vee R), \neg(Q \wedge \neg R)\}$

39

	1. $P \downarrow (Q \vee R)$			
	2. $\neg(Q \wedge \neg R)$			
2. $\neg Q$	$\beta_1 [2.]$	3. $\neg\neg R$	$\beta_2 [2.]$	
5. $\neg P$	$\alpha_1 [1.]$	4. R	$\neg\neg [2.]$	
6. $\neg(Q \vee R)$	$\alpha_2 [1.]$...		
7. $\neg Q$	$\alpha_1 [7.]$			
8. $\neg R$	$\alpha_2 [7.]$			

Example — Tableau Proof for
 $(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$

40

... on blackboard ...

Example — Tableau for $\{P \downarrow (Q \vee R), \neg(Q \wedge \neg R)\}$

39

	1. $P \downarrow (Q \vee R)$			
	2. $\neg(Q \wedge \neg R)$			
2. $\neg Q$	$\beta_1 [2.]$	3. $\neg\neg R$	$\beta_2 [2.]$	
5. $\neg P$	$\alpha_1 [1.]$	4. R	$\neg\neg [2.]$	
6. $\neg(Q \vee R)$	$\alpha_2 [1.]$...		
7. $\neg Q$	$\alpha_1 [7.]$			
8. $\neg R$	$\alpha_2 [7.]$			

Example — Tableau Proof for
 $(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$

40

... on blackboard ...

Some Remarks:

- ▶ Soundness (anything provable is a tautology) and Completeness (any tautology is provable): soon!
- ▶ Tableau proof can be shorter than truth table verifications and the tableau method easily extends to first-order logic.
- ▶ The tableau rules are applied non-deterministic — we have the choice where to apply them (implementations will need to make this concrete)
- ▶ Important: In propositional tableau we never have to use/process a formula more than once on any branch.
- ▶ If we allow reuse of formulas, then Completeness can be easily shown by a very general method: Model Existence Theorem (Abstract Consistency Method)
- ▶ If we disallow reuse of formulas (strictness), then Completeness is more difficult to show; other techniques are needed.

Some Remarks:

- ▶ Soundness (anything provable is a tautology) and Completeness (any tautology is provable): soon!
- ▶ Tableau proof can be shorter than truth table verifications and the tableau method easily extends to first-order logic.
- ▶ The tableau rules are applied non-deterministic — we have the choice where to apply them (implementations will need to make this concrete)
- ▶ Important: In propositional tableau we never have to use/process a formula more than once on any branch.
- ▶ If we allow reuse of formulas, then Completeness can be easily shown by a very general method: Model Existence Theorem (Abstract Consistency Method)
- ▶ If we disallow reuse of formulas (strictness), then Completeness is more difficult to show; other techniques are needed.

Some Remarks:

- ▶ Soundness (anything provable is a tautology) and Completeness (any tautology is provable): soon!
- ▶ Tableau proof can be shorter than truth table verifications and the tableau method easily extends to first-order logic.
- ▶ The tableau rules are applied non-deterministic — we have the choice where to apply them (implementations will need to make this concrete)
- ▶ Important: In propositional tableau we never have to use/process a formula more than once on any branch.
- ▶ If we allow reuse of formulas, then Completeness can be easily shown by a very general method: Model Existence Theorem (Abstract Consistency Method)
- ▶ If we disallow reuse of formulas (strictness), then Completeness is more difficult to show; other techniques are needed.

Some Remarks:

- ▶ Soundness (anything provable is a tautology) and Completeness (any tautology is provable): soon!
- ▶ Tableau proof can be shorter than truth table verifications and the tableau method easily extends to first-order logic.
- ▶ The tableau rules are applied non-deterministic — we have the choice where to apply them (implementations will need to make this concrete)
- ▶ Important: In propositional tableau we never have to use/process a formula more than once on any branch.
- ▶ If we allow reuse of formulas, then Completeness can be easily shown by a very general method: Model Existence Theorem (Abstract Consistency Method)
- ▶ If we disallow reuse of formulas (strictness), then Completeness is more difficult to show; other techniques are needed.

Some Remarks:

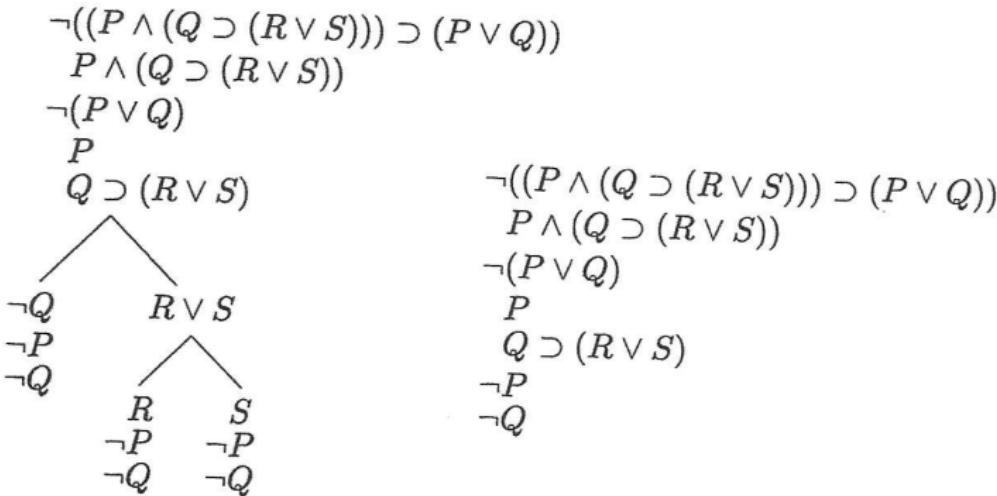
- ▶ Soundness (anything provable is a tautology) and Completeness (any tautology is provable): soon!
- ▶ Tableau proof can be shorter than truth table verifications and the tableau method easily extends to first-order logic.
- ▶ The tableau rules are applied non-deterministic — we have the choice where to apply them (implementations will need to make this concrete)
- ▶ Important: In propositional tableau we never have to use/process a formula more than once on any branch.
- ▶ If we allow reuse of formulas, then Completeness can be easily shown by a very general method: Model Existence Theorem (Abstract Consistency Method)
- ▶ If we disallow reuse of formulas (strictness), then Completeness is more difficult to show; other techniques are needed.

Some Remarks:

- ▶ Soundness (anything provable is a tautology) and Completeness (any tautology is provable): soon!
- ▶ Tableau proof can be shorter than truth table verifications and the tableau method easily extends to first-order logic.
- ▶ The tableau rules are applied non-deterministic — we have the choice where to apply them (implementations will need to make this concrete)
- ▶ Important: In propositional tableau we never have to use/process a formula more than once on any branch.
- ▶ If we allow reuse of formulas, then Completeness can be easily shown by a very general method: Model Existence Theorem (Abstract Consistency Method)
- ▶ If we disallow reuse of formulas (strictness), then Completeness is more difficult to show; other techniques are needed.

Expansion heuristics do matter:

$$\neg((P \wedge (Q \supset (R \vee S))) \supset (P \vee Q))$$



Definition — Strict Tableau

41

A tableau is *strict*, if no formula has had a Tableau Expansion Rule applied to it twice on the same branch.

Expansion heuristics do matter:

$$\neg((P \wedge (Q \supset (R \vee S))) \supset (P \vee Q))$$

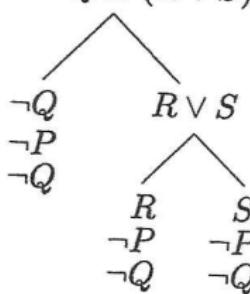
$$\neg((P \wedge (Q \supset (R \vee S))) \supset (P \vee Q))$$

$$P \wedge (Q \supset (R \vee S))$$

$$\neg(P \vee Q)$$

$$P$$

$$Q \supset (R \vee S)$$



$$\neg((P \wedge (Q \supset (R \vee S))) \supset (P \vee Q))$$

$$P \wedge (Q \supset (R \vee S))$$

$$\neg(P \vee Q)$$

$$P$$

$$Q \supset (R \vee S)$$

$$\neg P$$

$$\neg Q$$

Definition — Strict Tableau

41

A tableau is *strict*, if no formula has had a Tableau Expansion Rule applied to it twice on the same branch.

Remember: Resolution works on conjunction of disjunctions

Input: $\langle D_1, \dots, D_n \rangle$ where the D_i are clauses (disjunctions)

$$\frac{\langle D_1, \dots, [\dots, \neg\neg Z, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, Z, \dots], \dots, D_n \rangle} \quad \frac{\neg\neg Z}{Z}$$

$$\frac{\langle D_1, \dots, [\dots, \neg\top, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \perp, \dots], \dots, D_n \rangle} \quad \frac{\neg\top}{\perp}$$

analogous for $\frac{\neg\perp}{\top}$

$$\frac{\langle D_1, \dots, [\dots, \beta, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \beta_1, \beta_2, \dots], \dots, D_n \rangle} \quad \frac{\begin{matrix} \beta \\ \beta_1 \\ \beta_2 \end{matrix}}{\beta_1 \mid \beta_2}$$

$$\frac{\langle D_1, \dots, [\dots, \alpha, \dots], \dots, D_n \rangle}{\langle D_1, \dots, [\dots, \alpha_1, \dots], [\dots, \alpha_2, \dots], \dots, D_n \rangle} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Definition — Resolution Expansion Rules

42

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\beta}{\begin{array}{c} \beta_1 \\ \beta_2 \end{array}} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Example — $\{P \downarrow (Q \wedge R), \neg(Q \vee (P \supset Q))\}$

43

1.	$[P \downarrow (Q \wedge R)]$	
2.	$[\neg(Q \vee (P \supset Q))]$	
3.	$[\neg P]$	$\alpha_1 [1.]$
4.	$[\neg(Q \wedge R)]$	$\alpha_2 [1.]$
5.	$[\neg Q, \neg R]$	$\beta [4.]$
6.	$[\neg Q]$	$\alpha_1 [2.]$
7.	$[\neg(P \supset Q)]$	$\alpha_2 [2.]$
8.	$[P]$	$\alpha_1 [7.]$
9.	$[\neg Q]$	$\alpha_2 [7.]$

Definition — Resolution Expansion Rules

42

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\beta}{\begin{array}{c} \beta_1 \\ \beta_2 \end{array}} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Example — $\{P \downarrow (Q \wedge R), \neg(Q \vee (P \supset Q))\}$

43

1. $[P \downarrow (Q \wedge R)]$
2. $[\neg(Q \vee (P \supset Q))]$
3. $[\neg P]$
4. $[\neg(Q \wedge R)]$
5. $[\neg Q, \neg R]$
6. $[\neg Q]$
7. $[\neg(P \supset Q)]$
8. $[P]$
9. $[\neg Q]$

α_1 [1.]
 α_2 [1.]
 β [4.]
 α_1 [2.]
 α_2 [2.]
 α_1 [7.]
 α_2 [7.]

Definition — Resolution Expansion Rules

42

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\beta}{\begin{array}{c} \beta_1 \\ \beta_2 \end{array}} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Example — $\{P \downarrow (Q \wedge R), \neg(Q \vee (P \supset Q))\}$

43

- | | | | |
|----|--------------------------------|------------|------|
| 1. | $[P \downarrow (Q \wedge R)]$ | | |
| 2. | $[\neg(Q \vee (P \supset Q))]$ | | |
| 3. | $[\neg P]$ | α_1 | [1.] |
| 4. | $[\neg(Q \wedge R)]$ | α_2 | [1.] |
| 5. | $[\neg Q, \neg R]$ | β | [4.] |
| 6. | $[\neg Q]$ | α_1 | [2.] |
| 7. | $[\neg(P \supset Q)]$ | α_2 | [2.] |
| 8. | $[P]$ | α_1 | [7.] |
| 9. | $[\neg Q]$ | α_2 | [7.] |

Definition — Resolution Expansion Rules

42

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\beta}{\begin{array}{c} \beta_1 \\ \beta_2 \end{array}} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Example — $\{P \downarrow (Q \wedge R), \neg(Q \vee (P \supset Q))\}$

43

- | | | | |
|----|--------------------------------|------------|------|
| 1. | $[P \downarrow (Q \wedge R)]$ | | |
| 2. | $[\neg(Q \vee (P \supset Q))]$ | | |
| 3. | $[\neg P]$ | α_1 | [1.] |
| 4. | $[\neg(Q \wedge R)]$ | α_2 | [1.] |
| 5. | $[\neg Q, \neg R]$ | β | [4.] |
| 6. | $[\neg Q]$ | α_1 | [2.] |
| 7. | $[\neg(P \supset Q)]$ | α_2 | [2.] |
| 8. | $[P]$ | α_1 | [7.] |
| 9. | $[\neg Q]$ | α_2 | [7.] |

Definition — Resolution Expansion Rules

42

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\beta}{\begin{array}{c} \beta_1 \\ \beta_2 \end{array}} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Example — $\{P \downarrow (Q \wedge R), \neg(Q \vee (P \supset Q))\}$

43

- | | | | |
|----|--------------------------------|------------|------|
| 1. | $[P \downarrow (Q \wedge R)]$ | | |
| 2. | $[\neg(Q \vee (P \supset Q))]$ | | |
| 3. | $[\neg P]$ | α_1 | [1.] |
| 4. | $[\neg(Q \wedge R)]$ | α_2 | [1.] |
| 5. | $[\neg Q, \neg R]$ | β | [4.] |
| 6. | $[\neg Q]$ | α_1 | [2.] |
| 7. | $[\neg(P \supset Q)]$ | α_2 | [2.] |
| 8. | $[P]$ | α_1 | [7.] |
| 9. | $[\neg Q]$ | α_2 | [7.] |

Definition — Resolution Expansion Rules

42

$$\frac{\neg\neg Z}{Z} \quad \frac{\neg T}{\perp} \quad \frac{\neg\perp}{T} \quad \frac{\beta}{\begin{array}{c} \beta_1 \\ \beta_2 \end{array}} \quad \frac{\alpha}{\alpha_1 \mid \alpha_2}$$

Example — $\{P \downarrow (Q \wedge R), \neg(Q \vee (P \supset Q))\}$

43

- | | | | |
|----|--------------------------------|------------|------|
| 1. | $[P \downarrow (Q \wedge R)]$ | | |
| 2. | $[\neg(Q \vee (P \supset Q))]$ | | |
| 3. | $[\neg P]$ | α_1 | [1.] |
| 4. | $[\neg(Q \wedge R)]$ | α_2 | [1.] |
| 5. | $[\neg Q, \neg R]$ | β | [4.] |
| 6. | $[\neg Q]$ | α_1 | [2.] |
| 7. | $[\neg(P \supset Q)]$ | α_2 | [2.] |
| 8. | $[P]$ | α_1 | [7.] |
| 9. | $[\neg Q]$ | α_2 | [7.] |

Definition — Strict Resolution

44

We call a sequence of Clause Set Reduction Rule applications *strict* if every disjunction has at most one rule applied to it.

Remark: Just as with Tableaux, completeness of the non-strict version of Resolution will be easy and elegant; strict completeness requires more involved techniques. Strictness matters for implementation.

Remark: Modern resolution theorem provers use more advanced clause normalization techniques in order to compute smaller/better suited clause sets.

Definition — Strict Resolution

44

We call a sequence of Clause Set Reduction Rule applications *strict* if every disjunction has at most one rule applied to it.

Remark: Just as with Tableaux, completeness of the non-strict version of Resolution will be easy and elegant; strict completeness requires more involved techniques. Strictness matters for implementation.

Remark: Modern resolution theorem provers use more advanced clause normalization techniques in order to compute smaller/better suited clause sets.

Definition — Strict Resolution

44

We call a sequence of Clause Set Reduction Rule applications *strict* if every disjunction has at most one rule applied to it.

Remark: Just as with Tableaux, completeness of the non-strict version of Resolution will be easy and elegant; strict completeness requires more involved techniques. Strictness matters for implementation.

Remark: Modern resolution theorem provers use more advanced clause normalization techniques in order to compute smaller/better suited clause sets.

Definition — Resolution

45

Let D_1 and D_2 be disjunctions, with X occurring as a member of D_1 and $\neg X$ as a member of D_2 . Let D be the result of the following:

- ▶ Deleting all occurrences of X from D_1
- ▶ Deleting all occurrences of $\neg X$ from D_2
- ▶ Combining the resulting disjunctions into a big disjunction D

D is the result of *resolving* D_1 and D_2 on X . D is called the *resolvent* of D_1 and D_2 , with X being the formula *resolved on*. We also allow a trivial special case of resolution: if F is a disjunction with occurrences of \perp , and D is the result of deleting all occurrences of \perp from F , then D is called the *trivial resolvent* of F .

Example — Resolution

46

... blackboard ...

Definition — Resolution

45

Let D_1 and D_2 be disjunctions, with X occurring as a member of D_1 and $\neg X$ as a member of D_2 . Let D be the result of the following:

- ▶ Deleting all occurrences of X from D_1
- ▶ Deleting all occurrences of $\neg X$ from D_2
- ▶ Combining the resulting disjunctions into a big disjunction D

D is the result of *resolving* D_1 and D_2 on X . D is called the *resolvent* of D_1 and D_2 , with X being the formula *resolved on*. We also allow a trivial special case of resolution: if F is a disjunction with occurrences of \perp , and D is the result of deleting all occurrences of \perp from F , then D is called the *trivial resolvent* of F .

Example — Resolution

46

... blackboard ...

Definition — Resolution

45

Let D_1 and D_2 be disjunctions, with X occurring as a member of D_1 and $\neg X$ as a member of D_2 . Let D be the result of the following:

- ▶ Deleting all occurrences of X from D_1
- ▶ Deleting all occurrences of $\neg X$ from D_2
- ▶ Combining the resulting disjunctions into a big disjunction D

D is the result of *resolving* D_1 and D_2 on X . D is called the *resolvent* of D_1 and D_2 , with X being the formula *resolved on*. We also allow a trivial special case of resolution: if F is a disjunction with occurrences of \perp , and D is the result of deleting all occurrences of \perp from F , then D is called the *trivial resolvent* of F .

Example — Resolution

46

... blackboard ...

Definition — Resolution

45

Let D_1 and D_2 be disjunctions, with X occurring as a member of D_1 and $\neg X$ as a member of D_2 . Let D be the result of the following:

- ▶ Deleting all occurrences of X from D_1
- ▶ Deleting all occurrences of $\neg X$ from D_2
- ▶ Combining the resulting disjunctions into a big disjunction D

D is the result of *resolving* D_1 and D_2 on X . D is called the *resolvent* of D_1 and D_2 , with X being the formula *resolved on*. We also allow a trivial special case of resolution: if F is a disjunction with occurrences of \perp , and D is the result of deleting all occurrences of \perp from F , then D is called the *trivial resolvent* of F .

Example — Resolution

46

... blackboard ...

Definition — Resolution Rule

47

D follows from the disjunctions D_1 and D_2 by the *Resolution Rule* if D is the result of resolving D_1 and D_2 on some formula X . If X is atomic, we call this an *atomic application of the Resolution Rule*.

Remark: There is no analog of the strictness for the Resolution Rule. We may have to use disjunctions more than once to ensure completeness.

Definition — Resolution Rule

47

D follows from the disjunctions D_1 and D_2 by the *Resolution Rule* if D is the result of resolving D_1 and D_2 on some formula X . If X is atomic, we call this an *atomic application of the Resolution Rule*.

Remark: There is no analog of the strictness for the Resolution Rule. We may have to use disjunctions more than once to ensure completeness.

Definition — Resolution Expansion

48

[A_1]

: is a *Resolution Expansion* for $\{A_1, \dots, A_n\}$

[A_n]

If S is a Resolution Expansion for $\{A_1, \dots, A_n\}$ and D results from some line or lines of S by the application of a Clause Set Reduction Rule or the Resolution Rule, then S with D added as a new line is also a *Resolution Expansion* for $\{A_1, \dots, A_n\}$.

Definition — Closed

49

We call a Resolution Expansion containing the empty clause \emptyset *closed*.

Note: if $\emptyset \in S$ then for any valuation v we have $v(S) = \text{f}$ (since $v\emptyset = \text{f}$ and S is a conjunction).

Definition — Resolution Expansion

48

[A_1]

: is a *Resolution Expansion* for $\{A_1, \dots, A_n\}$

[A_n]

If S is a Resolution Expansion for $\{A_1, \dots, A_n\}$ and D results from some line or lines of S by the application of a Clause Set Reduction Rule or the Resolution Rule, then S with D added as a new line is also a *Resolution Expansion* for $\{A_1, \dots, A_n\}$.

Definition — Closed

49

We call a Resolution Expansion containing the empty clause \perp *closed*.

Note: if $\perp \in S$ then for any valuation v we have $v(S) = \text{f}$ (since $v\perp = \text{f}$ and S is a conjunction).

Definition — Resolution Expansion

48

[A_1]

: is a *Resolution Expansion* for $\{A_1, \dots, A_n\}$

[A_n]

If S is a Resolution Expansion for $\{A_1, \dots, A_n\}$ and D results from some line or lines of S by the application of a Clause Set Reduction Rule or the Resolution Rule, then S with D added as a new line is also a *Resolution Expansion* for $\{A_1, \dots, A_n\}$.

Definition — Closed

49

We call a Resolution Expansion containing the empty clause [] *closed*.

Note: if $[] \in S$ then for any valuation v we have $v(S) = \text{f}$ (since $v[] = \text{f}$ and S is a conjunction).

Definition — Resolution Expansion

48

[A_1]

: is a *Resolution Expansion* for $\{A_1, \dots, A_n\}$

[A_n]

If S is a Resolution Expansion for $\{A_1, \dots, A_n\}$ and D results from some line or lines of S by the application of a Clause Set Reduction Rule or the Resolution Rule, then S with D added as a new line is also a *Resolution Expansion* for $\{A_1, \dots, A_n\}$.

Definition — Closed

49

We call a Resolution Expansion containing the empty clause [] *closed*.

Note: if $[] \in S$ then for any valuation v we have $v(S) = \text{f}$ (since $v[] = \text{f}$ and S is a conjunction).

Definition — Resolution Proof

50

A *resolution proof* of formula $X \in \mathbf{P}$ is a closed resolution for $\{\neg X\}$. X is a *theorem* of the resolution system if X has a resolution proof. We write: $\vdash_{pr} X$.

Example — Resolution Proof

51

$((P \wedge Q) \vee (R \supset S)) \supset ((P \vee (R \supset S)) \wedge (Q \vee (R \supset S)))$
... on blackboard ...

Definition — Resolution Proof

50

A *resolution proof* of formula $X \in \mathbf{P}$ is a closed resolution for $\{\neg X\}$. X is a *theorem* of the resolution system if X has a resolution proof. We write: $\vdash_{pr} X$.

Example — Resolution Proof

51

$((P \wedge Q) \vee (R \supset S)) \supset ((P \vee (R \supset S)) \wedge (Q \vee (R \supset S)))$
... on blackboard ...

- ▶ How about writing a tableaux or resolution theorem prover yourself?
- ▶ Let's discuss this this afternoon in the practical sessions.

Definition — Soundness

52

A proof system for propositional logic is called *sound* if it can prove only tautologies.

Remarks

- ▶ A sound proof system (resp. sound theorem prover) does not produce incorrect results.
- ▶ If we prove soundness for the basic unrestricted proof systems then this implies soundness also for the systems in which restrictions (e.g. strictness) are applied. Hence, soundness is typically proved for unrestricted systems.
- ▶ Soundness of the tableau system is given here; resolution is similar (exercise).

Definition — Soundness

52

A proof system for propositional logic is called *sound* if it can prove only tautologies.

Remarks

- ▶ A sound proof system (resp. sound theorem prover) does not produce incorrect results.
- ▶ If we prove soundness for the basic unrestricted proof systems then this implies soundness also for the systems in which restrictions (e.g. strictness) are applied. Hence, soundness is typically proved for unrestricted systems.
- ▶ Soundness of the tableau system is given here; resolution is similar (exercise).

Definition — Soundness

52

A proof system for propositional logic is called *sound* if it can prove only tautologies.

Remarks

- ▶ A sound proof system (resp. sound theorem prover) does not produce incorrect results.
- ▶ If we prove soundness for the basic unrestricted proof systems then this implies soundness also for the systems in which restrictions (e.g. strictness) are applied. Hence, soundness is typically proved for unrestricted systems.
- ▶ Soundness of the tableau system is given here; resolution is similar (exercise).

Definition — Soundness

52

A proof system for propositional logic is called *sound* if it can prove only tautologies.

Remarks

- ▶ A sound proof system (resp. sound theorem prover) does not produce incorrect results.
- ▶ If we prove soundness for the basic unrestricted proof systems then this implies soundness also for the systems in which restrictions (e.g. strictness) are applied. Hence, soundness is typically proved for unrestricted systems.
- ▶ Soundness of the tableau system is given here; resolution is similar (exercise).

Idea: Define what it means for a tableau to be satisfiable. Then show that satisfiability is a loop invariant. This easily implies soundness.

Definition — Satisfiability (Tableaux)

53

A (conjunctive) set S of propositional formulas is *satisfiable* if some Boolean valuation v maps every member of S to \top .

A tableau branch is *satisfiable* if the set of propositional formulas on it is satisfiable.

A tableau T is *satisfiable* if at least one branch of T is satisfiable.

Idea: Define what it means for a tableau to be satisfiable. Then show that satisfiability is a loop invariant. This easily implies soundness.

Definition — Satisfiability (Tableaux)

53

A (conjunctive) set S of propositional formulas is *satisfiable* if some Boolean valuation v maps every member of S to \top .

A tableau branch is *satisfiable* if the set of propositional formulas on it is satisfiable.

A tableau T is *satisfiable* if at least one branch of T is satisfiable.

Idea: Define what it means for a tableau to be satisfiable. Then show that satisfiability is a loop invariant. This easily implies soundness.

Definition — Satisfiability (Tableaux)

53

A (conjunctive) set S of propositional formulas is *satisfiable* if some Boolean valuation v maps every member of S to \top .

A tableau branch is *satisfiable* if the set of propositional formulas on it is satisfiable.

A tableau T is *satisfiable* if at least one branch of T is satisfiable.

Idea: Define what it means for a tableau to be satisfiable. Then show that satisfiability is a loop invariant. This easily implies soundness.

Definition — Satisfiability (Tableaux)

53

A (conjunctive) set S of propositional formulas is *satisfiable* if some Boolean valuation v maps every member of S to \top .

A tableau branch is *satisfiable* if the set of propositional formulas on it is satisfiable.

A tableau T is *satisfiable* if at least one branch of T is satisfiable.

Proposition

54

Any application of a Tableau Expansion Rule to a satisfiable tableau yields another satisfiable tableau.

Proof: (next slide)

Proposition

54

Any application of a Tableau Expansion Rule to a satisfiable tableau yields another satisfiable tableau.

Proof: (next slide)

Propositional Logic: Calculi — Tableaux Soundness

Proof: Assume tableau T is satisfiable, i.e. T has at least one satisfiable branch τ . Moreover, let tableau T^* be obtained from T by application of a Tableau Expansion Rule r to branch θ in T . We show by case distinction that T^* is satisfiable.

- ▶ $\tau \neq \theta$: τ is also a branch of T^* , hence T^* satisfiable.
- ▶ $\tau = \theta$: θ is satisfiable, i.e. $v(X) = t$ for all $X \in \theta$. Let $X \in \theta$ be the formula that r was applied to:
 - ▶ $X = \neg T$ and $X = \neg 1$; trivial
 - ▶ $X = \neg\neg Z$; $v(\neg\neg Z) = t$; Z is added to T to obtain T^* ; $v(Z) = t$ since $\neg\neg Z \equiv Z$ is a tautology and hence $v(\neg\neg Z) = v(Z)$
 - ▶ $X = \alpha$; $v(\alpha) = t$; α_1 and α_2 were added to T to obtain T^* ; $v(\alpha_1) = v(\alpha_2) = t$ since $\alpha \equiv (\alpha_1 \wedge \alpha_2)$ is a tautology and hence $v(\alpha) = v(\alpha_1 \wedge \alpha_2) = v(\alpha_1) \wedge v(\alpha_2)$
 - ▶ $X = \beta$; $v(\alpha) = t$; left and right children β_1 and β_2 were added to the last node of T to obtain T^* ; since $v(\beta) = v(\alpha_1 \vee \beta_2) = v(\alpha_1) \vee v(\beta_2)$ we must have $v(\beta_1) = t$ or $v(\beta_2) = t$; hence either the left-hand branch (containing β_1) or the right-hand branch (containing β_2) is satisfiable

Propositional Logic: Calculi — Tableaux Soundness

Proof: Assume tableau T is satisfiable, i.e. T has at least one satisfiable branch τ . Moreover, let tableau T^* be obtained from T by application of a Tableau Expansion Rule r to branch θ in T . We show by case distinction that T^* is satisfiable.

- ▶ $\tau \neq \theta$: τ is also a branch of T^* , hence T^* satisfiable.
- ▶ $\tau = \theta$: θ is satisfiable, i.e. $v(X) = \text{t}$ for all $X \in \theta$. Let $X \in \theta$ be the formula that r was applied to:
 - ▶ $X = \neg T$ and $X = \neg \perp$: trivial
 - ▶ $X = \neg \neg Z$: $v(\neg \neg Z) = \text{t}$; Z is added to T to obtain T^* ; $v(Z) = \text{t}$ since $\neg \neg Z \equiv Z$ is a tautology and hence $v(\neg \neg Z) = v(Z)$
 - ▶ $X = \alpha$: $v(\alpha) = \text{t}$; α_1 and α_2 were added to T to obtain T^* ; $v(\alpha_1) = v(\alpha_2) = \text{t}$ since $\alpha \equiv (\alpha_1 \wedge \alpha_2)$ is a tautology and hence $v(\alpha) = v(\alpha_1 \wedge \alpha_2) = v(\alpha_1) \wedge v(\alpha_2)$
 - ▶ $X = \beta$: $v(\alpha) = \text{t}$; left and right children β_1 and β_2 were added to the last node of T to obtain T^* ; since $v(\beta) = v(\alpha_1 \vee \beta_2) = v(\alpha_1) \vee v(\beta_2)$ we must have $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; hence either the left-hand branch (containing β_1) or the right-hand branch (containing β_2) is satisfiable

Proof: Assume tableau T is satisfiable, i.e. T has at least one satisfiable branch τ . Moreover, let tableau T^* be obtained from T by application of a Tableau Expansion Rule r to branch θ in T . We show by case distinction that T^* is satisfiable.

- ▶ $\tau \neq \theta$: τ is also a branch of T^* , hence T^* satisfiable.
- ▶ $\tau = \theta$: θ is satisfiable, i.e. $v(X) = \text{t}$ for all $X \in \theta$. Let $X \in \theta$ be the formula that r was applied to:
 - ▶ $X = \neg T$ and $X = \neg \perp$: trivial
 - ▶ $X = \neg \neg Z$: $v(\neg \neg Z) = \text{t}$; Z is added to T to obtain T^* ; $v(Z) = \text{t}$ since $\neg \neg Z \equiv Z$ is a tautology and hence $v(\neg \neg Z) = v(Z)$
 - ▶ $X = \alpha$: $v(\alpha) = \text{t}$; α_1 and α_2 were added to T to obtain T^* ; $v(\alpha_1) = v(\alpha_2) = \text{t}$ since $\alpha \equiv (\alpha_1 \wedge \alpha_2)$ is a tautology and hence $v(\alpha) = v(\alpha_1 \wedge \alpha_2) = v(\alpha_1) \wedge v(\alpha_2)$
 - ▶ $X = \beta$: $v(\alpha) = \text{t}$; left and right children β_1 and β_2 were added to the last node of T to obtain T^* ; since $v(\beta) = v(\alpha_1 \vee \beta_2) = v(\alpha_1) \vee v(\beta_2)$ we must have $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; hence either the left-hand branch (containing β_1) or the right-hand branch (containing β_2) is satisfiable

Propositional Logic: Calculi — Tableaux Soundness

Proof: Assume tableau T is satisfiable, i.e. T has at least one satisfiable branch τ . Moreover, let tableau T^* be obtained from T by application of a Tableau Expansion Rule r to branch θ in T . We show by case distinction that T^* is satisfiable.

- ▶ $\tau \neq \theta$: τ is also a branch of T^* , hence T^* satisfiable.
- ▶ $\tau = \theta$: θ is satisfiable, i.e. $v(X) = \text{t}$ for all $X \in \theta$. Let $X \in \theta$ be the formula that r was applied to:
 - ▶ $X = \neg T$ and $X = \neg \perp$: trivial
 - ▶ $X = \neg\neg Z$: $v(\neg\neg Z) = \text{t}$; Z is added to T to obtain T^* ; $v(Z) = \text{t}$ since $\neg\neg Z \equiv Z$ is a tautology and hence $v(\neg\neg Z) = v(Z)$
 - ▶ $X = \alpha$: $v(\alpha) = \text{t}$; α_1 and α_2 were added to T to obtain T^* ; $v(\alpha_1) = v(\alpha_2) = \text{t}$ since $\alpha \equiv (\alpha_1 \wedge \alpha_2)$ is a tautology and hence $v(\alpha) = v(\alpha_1 \wedge \alpha_2) = v(\alpha_1) \wedge v(\alpha_2)$
 - ▶ $X = \beta$: $v(\alpha) = \text{t}$; left and right children β_1 and β_2 were added to the last node of T to obtain T^* ; since $v(\beta) = v(\alpha_1 \vee \beta_2) = v(\alpha_1) \vee v(\beta_2)$ we must have $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; hence either the left-hand branch (containing β_1) or the right-hand branch (containing β_2) is satisfiable

Proof: Assume tableau T is satisfiable, i.e. T has at least one satisfiable branch τ . Moreover, let tableau T^* be obtained from T by application of a Tableau Expansion Rule r to branch θ in T . We show by case distinction that T^* is satisfiable.

- ▶ $\tau \neq \theta$: τ is also a branch of T^* , hence T^* satisfiable.
- ▶ $\tau = \theta$: θ is satisfiable, i.e. $v(X) = \text{t}$ for all $X \in \theta$. Let $X \in \theta$ be the formula that r was applied to:
 - ▶ $X = \neg T$ and $X = \neg \perp$: trivial
 - ▶ $X = \neg\neg Z$: $v(\neg\neg Z) = \text{t}$; Z is added to T to obtain T^* ; $v(Z) = \text{t}$ since $\neg\neg Z \equiv Z$ is a tautology and hence $v(\neg\neg Z) = v(Z)$
 - ▶ $X = \alpha$: $v(\alpha) = \text{t}$; α_1 and α_2 were added to T to obtain T^* ; $v(\alpha_1) = v(\alpha_2) = \text{t}$ since $\alpha \equiv (\alpha_1 \wedge \alpha_2)$ is a tautology and hence $v(\alpha) = v(\alpha_1 \wedge \alpha_2) = v(\alpha_1) \wedge v(\alpha_2)$
 - ▶ $X = \beta$: $v(\alpha) = \text{t}$; left and right children β_1 and β_2 were added to the last node of T to obtain T^* ; since $v(\beta) = v(\alpha_1 \vee \beta_2) = v(\alpha_1) \vee v(\beta_2)$ we must have $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; hence either the left-hand branch (containing β_1) or the right-hand branch (containing β_2) is satisfiable

Propositional Logic: Calculi — Tableaux Soundness

Proof: Assume tableau T is satisfiable, i.e. T has at least one satisfiable branch τ . Moreover, let tableau T^* be obtained from T by application of a Tableau Expansion Rule r to branch θ in T . We show by case distinction that T^* is satisfiable.

- ▶ $\tau \neq \theta$: τ is also a branch of T^* , hence T^* satisfiable.
- ▶ $\tau = \theta$: θ is satisfiable, i.e. $v(X) = \text{t}$ for all $X \in \theta$. Let $X \in \theta$ be the formula that r was applied to:
 - ▶ $X = \neg T$ and $X = \neg \perp$: trivial
 - ▶ $X = \neg\neg Z$: $v(\neg\neg Z) = \text{t}$; Z is added to T to obtain T^* ; $v(Z) = \text{t}$ since $\neg\neg Z \equiv Z$ is a tautology and hence $v(\neg\neg Z) = v(Z)$
 - ▶ $X = \alpha$: $v(\alpha) = \text{t}$; α_1 and α_2 were added to T to obtain T^* ; $v(\alpha_1) = v(\alpha_2) = \text{t}$ since $\alpha \equiv (\alpha_1 \wedge \alpha_2)$ is a tautology and hence $v(\alpha) = v(\alpha_1 \wedge \alpha_2) = v(\alpha_1) \wedge v(\alpha_2)$
 - ▶ $X = \beta$: $v(\alpha) = \text{t}$; left and right children β_1 and β_2 were added to the last node of T to obtain T^* ; since $v(\beta) = v(\alpha_1 \vee \beta_2) = v(\alpha_1) \vee v(\beta_2)$ we must have $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; hence either the left-hand branch (containing β_1) or the right-hand branch (containing β_2) is satisfiable

Proposition

55

If there is a closed tableau for a formula set S , then S is not satisfiable.

Proof: Given a closed tableau T^* for S . Assume S is satisfiable. Since S is satisfiable, the initial tableau T for S is satisfiable. By the previous lemma all subsequent tableaux, including T^* , are satisfiable; contradiction to the definition of closed tableau.

Theorem — Propositional Tableau Soundness

56

If X has a tableau proof, then X is a tautology.

Proof: A tableau proof for X is a closed tableau for $\{\neg X\}$. By the above lemma $\{\neg X\}$ is not satisfiable, i.e. $v(\neg X) = \text{f}$; hence $v(X) = \text{t}$, i.e. X is a tautology.

Proposition

55

If there is a closed tableau for a formula set S , then S is not satisfiable.

Proof: Given a closed tableau T^* for S . Assume S is satisfiable. Since S is satisfiable, the initial tableau T for S is satisfiable. By the previous lemma all subsequent tableaux, including T^* , are satisfiable; contradiction to the definition of closed tableau.

Theorem — Propositional Tableau Soundness

56

If X has a tableau proof, then X is a tautology.

Proof: A tableau proof for X is a closed tableau for $\{\neg X\}$. By the above lemma $\{\neg X\}$ is not satisfiable, i.e. $v(\neg X) = \text{f}$; hence $v(X) = \text{t}$, i.e. X is a tautology.

Proposition

55

If there is a closed tableau for a formula set S , then S is not satisfiable.

Proof: Given a closed tableau T^* for S . Assume S is satisfiable. Since S is satisfiable, the initial tableau T for S is satisfiable. By the previous lemma all subsequent tableaux, including T^* , are satisfiable; contradiction to the definition of closed tableau.

Theorem — Propositional Tableau Soundness

56

If X has a tableau proof, then X is a tautology.

Proof: A tableau proof for X is a closed tableau for $\{\neg X\}$. By the above lemma $\{\neg X\}$ is not satisfiable, i.e. $v(\neg X) = \text{f}$; hence $v(X) = \text{t}$, i.e. X is a tautology.

Proposition

55

If there is a closed tableau for a formula set S , then S is not satisfiable.

Proof: Given a closed tableau T^* for S . Assume S is satisfiable. Since S is satisfiable, the initial tableau T for S is satisfiable. By the previous lemma all subsequent tableaux, including T^* , are satisfiable; contradiction to the definition of closed tableau.

Theorem — Propositional Tableau Soundness

56

If X has a tableau proof, then X is a tautology.

Proof: A tableau proof for X is a closed tableau for $\{\neg X\}$. By the above lemma $\{\neg X\}$ is not satisfiable, i.e. $v(\neg X) = \text{f}$; hence $v(X) = \text{t}$, i.e. X is a tautology.

Definition — Satisfiability (Resolution)

57

A resolution expansion is *satisfiable* if some Boolean valuation ν maps every line of it to t .

Proposition

58

Any application of a Clause Set Reduction Rule or the Resolution Rule to a satisfiable Resolution Expansion yields another satisfiable Resolution Expansion.

Proof: ... exercise ...

Definition — Satisfiability (Resolution)

57

A resolution expansion is *satisfiable* if some Boolean valuation ν maps every line of it to t .

Proposition

58

Any application of a Clause Set Reduction Rule or the Resolution Rule to a satisfiable Resolution Expansion yields another satisfiable Resolution Expansion.

Proof: ... exercise ...

Proposition

59

If there is a closed Resolution Expansion for a set S , then S is not satisfiable.

Proof: ... exercise ...

Theorem

60

If $X \in \mathbf{P}$ has a resolution proof, then X is a tautology.

Proof: ... exercise ...

Proposition

59

If there is a closed Resolution Expansion for a set S , then S is not satisfiable.

Proof: ... exercise ...

Theorem

60

If $X \in \mathbf{P}$ has a resolution proof, then X is a tautology.

Proof: ... exercise ...

- ▶ How to prove completeness? (Completeness: Given that I have a true statement, can my calculus/proof system indeed show that it is true?)
- ▶ (Refutation) completeness can be proved rather easily for resolution or tableau in propositional logic as given here.
- ▶ For first-order and especially higher-order logic completeness proofs become increasingly difficult.
- ▶ Therefore we will introduce a proof tool that uniformly supports completeness proofs (and many other things): abstract consistency method.
- ▶ This proof tool is based on a strong theorem which connects syntax and semantics: model existence theorem.

- ▶ How to prove completeness? (Completeness: Given that I have a true statement, can my calculus/proof system indeed show that it is true?)
- ▶ (Refutation) completeness can be proved rather easily for resolution or tableau in propositional logic as given here.
- ▶ For first-order and especially higher-order logic completeness proofs become increasingly difficult.
- ▶ Therefore we will introduce a **proof tool** that uniformly supports completeness proofs (and many other things): **abstract consistency method**.
- ▶ This proof tool is based on a **strong theorem** which connects syntax and semantics: **model existence theorem**.

- ▶ How to prove completeness? (Completeness: Given that I have a true statement, can my calculus/proof system indeed show that it is true?)
- ▶ (Refutation) completeness can be proved rather easily for resolution or tableau in propositional logic as given here.
- ▶ For first-order and especially higher-order logic completeness proofs become increasingly difficult.
- ▶ Therefore we will introduce a **proof tool** that uniformly supports completeness proofs (and many other things): **abstract consistency method**.
- ▶ This proof tool is based on a **strong theorem** which connects syntax and semantics: **model existence theorem**.

- ▶ How to prove completeness? (Completeness: Given that I have a true statement, can my calculus/proof system indeed show that it is true?)
- ▶ (Refutation) completeness can be proved rather easily for resolution or tableau in propositional logic as given here.
- ▶ For first-order and especially higher-order logic completeness proofs become increasingly difficult.
- ▶ Therefore we will introduce a **proof tool** that uniformly supports completeness proofs (and many other things): **abstract consistency method**.
- ▶ This proof tool is based on a **strong theorem** which connects syntax and semantics: **model existence theorem**.

Propositional Logic: Calculi — Completeness

- ▶ How to prove completeness? (Completeness: Given that I have a true statement, can my calculus/proof system indeed show that it is true?)
- ▶ (Refutation) completeness can be proved rather easily for resolution or tableau in propositional logic as given here.
- ▶ For first-order and especially higher-order logic completeness proofs become increasingly difficult.
- ▶ Therefore we will introduce a **proof tool** that uniformly supports completeness proofs (and many other things): **abstract consistency method**.
- ▶ This proof tool is based on a **strong theorem** which connects syntax and semantics: **model existence theorem**.

Abstract Consistency Technique

- ▶ Technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [Hintikka55,Smullyan63,Smullyan68]. It is well explained in Fitting's textbook [Fitting96].
- ▶ The technique has been (partly) extended to higher-order logic by Peter Andrews' in [Andrews71]; Peter Andrews only achieves a generalization for his rather weak semantical v -complexes (i.e., a very weak notion of semantics for higher-order logic) and not, for instance for Henkin Semantics. This extension is well explained in Peter Andrews's textbook [Andrews02].
- ▶ The technique has been extended to Henkin semantics (and a whole landscape of notions of semantics for higher-order logic) in [BenzmüllerBrownKohlhase04] (problem still: saturation).
- ▶ (Technique is rather an overkill for propositional logic).

Abstract Consistency Technique

- ▶ Technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [Hintikka55,Smullyan63,Smullyan68]. It is well explained in Fitting's textbook [Fitting96].
- ▶ The technique has been (partly) extended to higher-order logic by Peter Andrews' in [Andrews71]; Peter Andrews only achieves a generalization for his rather weak semantical v -complexes (i.e., a very weak notion of semantics for higher-order logic) and not, for instance for Henkin Semantics. This extension is well explained in Peter Andrews's textbook [Andrews02].
- ▶ The technique has been extended to Henkin semantics (and a whole landscape of notions of semantics for higher-order logic) in [BenzmüllerBrownKohlhase04] (problem still: saturation).
- ▶ (Technique is rather an overkill for propositional logic).

Abstract Consistency Technique

- ▶ Technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [Hintikka55,Smullyan63,Smullyan68]. It is well explained in Fitting's textbook [Fitting96].
- ▶ The technique has been (partly) extended to higher-order logic by Peter Andrews' in [Andrews71]; Peter Andrews only achieves a generalization for his rather weak semantical v -complexes (i.e., a very weak notion of semantics for higher-order logic) and not, for instance for Henkin Semantics. This extension is well explained in Peter Andrews's textbook [Andrews02].
- ▶ The technique has been extended to Henkin semantics (and a whole landscape of notions of semantics for higher-order logic) in [BenzmüllerBrownKohlhase04] (problem still: saturation).
- ▶ (Technique is rather an overkill for propositional logic).

Abstract Consistency Technique

- ▶ Technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [Hintikka55,Smullyan63,Smullyan68]. It is well explained in Fitting's textbook [Fitting96].
- ▶ The technique has been (partly) extended to higher-order logic by Peter Andrews' in [Andrews71]; Peter Andrews only achieves a generalization for his rather weak semantical v -complexes (i.e., a very weak notion of semantics for higher-order logic) and not, for instance for Henkin Semantics. This extension is well explained in Peter Andrews's textbook [Andrews02].
- ▶ The technique has been extended to Henkin semantics (and a whole landscape of notions of semantics for higher-order logic) in [BenzmüllerBrownKohlhase04] (problem still: saturation).
- ▶ (Technique is rather an overkill for propositional logic).

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above we can show the completeness of a particular calculus S (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Propositional Logic: Abstract Consistency

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above we can show the completeness of a particular calculus S (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Propositional Logic: Abstract Consistency

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above *we can show the completeness of a particular calculus S* (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above *we can show the completeness of a particular calculus S* (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above *we can show the completeness of a particular calculus S* (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above *we can show the completeness of a particular calculus S* (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above *we can show the completeness of a particular calculus S* (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Abstract Consistency Technique

- ▶ A *model existence theorem* for a logical system L is a theorem of the form:
If a set of formulas Φ of L is a member of an abstract consistency class, then there exists a model for Φ .
- ▶ Given a model existence theorem as described above *we can show the completeness of a particular calculus S* (i.e., the derivability relation \vdash_S) by proving that the class Γ of sets of formulas Φ that are S -consistent (i.e., cannot be refuted in S) is an abstract consistency class. Then the model existence theorem tells us that S -consistent sets of sentences are satisfiable. Now we assume that a sentence A is true, so $\neg A$ is not satisfiable and is therefore C -inconsistent. Hence, $\neg A$ is refutable in C . This shows refutation completeness of C . (For many calculi C , this also shows A is provable, thus establishing completeness of C .)

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Abstract Consistency Method: Supports elegant Completeness Proofs

- ▶ very elegant and powerful proof techniques
- ▶ generalizes well for first-order and higher-order logic
- ▶ overkill for propositional proof procedure: much simpler proofs are possible

Main Steps:

1. Hintikka Sets, Hintikka Prop: connection between Syntax and Semantics
2. Abstract Consistency Property, Model Existence Theorem: abstract 'standard' completeness argument
3. Apply the technique to the Tableau and Resolution Systems

Note: Steps 1. and 2. are proved once and for all and can be reused; only 3. needs to be done for each proof system.

Definition — Propositional Hintikka Set

61

A set $H \subset \mathbf{P}$ of propositional formulas is called a *propositional Hintikka set*, provided that:

1. For all propositional letters $A \in \mathbf{L}$, not both $A \in H$ and $\neg A \in H$
2. $\perp \notin H$ and $\top \notin H$
3. if $\neg\neg Z \in H$ then $Z \in H$
4. if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$
5. if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

Hintikka sets are also called *downward saturated*.

Definition — Propositional Hintikka Set

61

A set $H \subset \mathbf{P}$ of propositional formulas is called a *propositional Hintikka set*, provided that:

1. For all propositional letters $A \in \mathbf{L}$, not both $A \in H$ and $\neg A \in H$
2. $\perp \notin H$ and $\top \notin H$
3. if $\neg\neg Z \in H$ then $Z \in H$
4. if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$
5. if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

Hintikka sets are also called *downward saturated*.

Definition — Propositional Hintikka Set

61

A set $H \subset \mathbf{P}$ of propositional formulas is called a *propositional Hintikka set*, provided that:

1. For all propositional letters $A \in \mathbf{L}$, not both $A \in H$ and $\neg A \in H$
2. $\perp \notin H$ and $\top \notin H$
3. if $\neg\neg Z \in H$ then $Z \in H$
4. if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$
5. if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

Hintikka sets are also called *downward saturated*.

Definition — Propositional Hintikka Set

61

A set $H \subset \mathbf{P}$ of propositional formulas is called a *propositional Hintikka set*, provided that:

1. For all propositional letters $A \in \mathbf{L}$, not both $A \in H$ and $\neg A \in H$
2. $\perp \notin H$ and $\top \notin H$
3. if $\neg\neg Z \in H$ then $Z \in H$
4. if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$
5. if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

Hintikka sets are also called *downward saturated*.

Definition — Propositional Hintikka Set

61

A set $H \subset \mathbf{P}$ of propositional formulas is called a *propositional Hintikka set*, provided that:

1. For all propositional letters $A \in \mathbf{L}$, not both $A \in H$ and $\neg A \in H$
2. $\perp \notin H$ and $\top \notin H$
3. if $\neg\neg Z \in H$ then $Z \in H$
4. if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$
5. if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

Hintikka sets are also called *downward saturated*.

Definition — Propositional Hintikka Set

61

A set $H \subset \mathbf{P}$ of propositional formulas is called a *propositional Hintikka set*, provided that:

1. For all propositional letters $A \in \mathbf{L}$, not both $A \in H$ and $\neg A \in H$
2. $\perp \notin H$ and $\top \notin H$
3. if $\neg\neg Z \in H$ then $Z \in H$
4. if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$
5. if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

Hintikka sets are also called *downward saturated*.

Definition — Propositional Hintikka Set

61

A set $H \subset \mathbf{P}$ of propositional formulas is called a *propositional Hintikka set*, provided that:

1. For all propositional letters $A \in \mathbf{L}$, not both $A \in H$ and $\neg A \in H$
2. $\perp \notin H$ and $\top \notin H$
3. if $\neg\neg Z \in H$ then $Z \in H$
4. if $\alpha \in H$ then $\alpha_1 \in H$ and $\alpha_2 \in H$
5. if $\beta \in H$ then $\beta_1 \in H$ or $\beta_2 \in H$

Hintikka sets are also called *downward saturated*.

Example — Propositional Hintikka Sets

62

The following are propositional Hintikka sets

- ▶ \emptyset
- ▶ the set of propositional letters L
- ▶ $\{P \wedge (\neg Q \supset R), P, (\neg Q \supset R), \neg\neg Q, Q\}$

Example — Propositional Hintikka Sets

62

The following are propositional Hintikka sets

- ▶ \emptyset
- ▶ the set of propositional letters L
- ▶ $\{P \wedge (\neg Q \supset R), P, (\neg Q \supset R), \neg\neg Q, Q\}$

Example — Propositional Hintikka Sets

62

The following are propositional Hintikka sets

- ▶ \emptyset
- ▶ the set of propositional letters L
- ▶ $\{P \wedge (\neg Q \supset R), P, (\neg Q \supset R), \neg\neg Q, Q\}$

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
$$f(A) = \text{t} \text{ if } A \in H; f(A) = \text{f} \text{ if } \neg A \in H; f(A) = \text{f} \text{ if } A \notin H$$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in P$ we have: either $X \notin H$ or else $v(X) = \text{t}$

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
$$f(A) = \text{t} \text{ if } A \in H; f(A) = \text{f} \text{ if } \neg A \in H; f(A) = \text{f} \text{ if } A \notin H$$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in P$ we have: either $X \notin H$ or else $v(X) = \text{t}$

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ↳ base case

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathcal{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ↳ base case

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathcal{P}$
we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ◀ base case

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$

- ▶ base case
 - ▶ $X = A$ for $A \in \mathbf{L}$: by construction of v resp. f
 - ▶ $X = \neg A$ for $A \in \mathbf{L}$: if $\neg A \in H$ then $v(A) = \text{f}$ by construction of f , then $v(\neg A) = \text{t}$ since v is a Boolean valuation
 - ▶ $X = \perp$ and $X = \neg \top$: $X \notin H$ by definition of Hintikka sets, nothing to show
 - ▶ $X = \neg \perp$: $v(\perp) = \text{f}$ and $v(\neg \perp) = \neg v(\perp) = \text{t}$ by def. of v
 - ▶ $X = \top$: $v(\top) = \text{t}$ by definition of v

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ▶ base case
 - ▶ $X = A$ for $A \in \mathbf{L}$: by construction of v resp. f
 - ▶ $X = \neg A$ for $A \in \mathbf{L}$: if $\neg A \in H$ then $v(A) = \text{f}$ by construction of f ; then $v(\neg A) = \text{t}$ since v is a Boolean valuation
 - ▶ $X = \perp$ and $X = \neg \top$: $X \notin H$ by definition of Hintikka sets, nothing to show
 - ▶ $X = \neg \perp$: $v(\perp) = \text{f}$ and $v(\neg \perp) = \neg v(\perp) = \text{t}$ by def. of v
 - ▶ $X = \top$: $v(\top) = \text{t}$ by definition of v

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ▶ base case
 - ▶ $X = A$ for $A \in \mathbf{L}$: by construction of v resp. f
 - ▶ $X = \neg A$ for $A \in \mathbf{L}$: if $\neg A \in H$ then $v(\neg A) = \text{f}$ by construction of f ; then $v(\neg A) = \text{t}$ since v is a Boolean valuation
 - ▶ $X = \perp$ and $X = \neg \top$: $X \notin H$ by definition of Hintikka sets, nothing to show
 - ▶ $X = \neg \perp$: $v(\perp) = \text{f}$ and $v(\neg \perp) = \neg v(\perp) = \text{t}$ by def. of v
 - ▶ $X = \top$: $v(\top) = \text{t}$ by definition of v

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ▶ base case
 - ▶ $X = A$ for $A \in \mathbf{L}$: by construction of v resp. f
 - ▶ $X = \neg A$ for $A \in \mathbf{L}$: if $\neg A \in H$ then $v(A) = \text{f}$ by construction of f ; then $v(\neg A) = \text{t}$ since v is a Boolean valuation
 - ▶ $X = \perp$ and $X = \neg \top$: $X \notin H$ by definition of Hintikka sets, nothing to show
 - ▶ $X = \neg \perp$: $v(\perp) = \text{f}$ and $v(\neg \perp) = \neg v(\perp) = \text{t}$ by def. of v
 - ▶ $X = \top$: $v(\top) = \text{t}$ by definition of v

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ▶ base case
 - ▶ $X = A$ for $A \in \mathbf{L}$: by construction of v resp. f
 - ▶ $X = \neg A$ for $A \in \mathbf{L}$: if $\neg A \in H$ then $v(A) = \text{f}$ by construction of f ; then $v(\neg A) = \text{t}$ since v is a Boolean valuation
 - ▶ $X = \perp$ and $X = \neg \top$: $X \notin H$ by definition of Hintikka sets, nothing to show
 - ▶ $X = \neg \perp$: $v(\perp) = \text{f}$ and $v(\neg \perp) = \neg v(\perp) = \text{t}$ by def. of v
 - ▶ $X = \top$: $v(\top) = \text{t}$ by definition of v

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ▶ base case
 - ▶ $X = A$ for $A \in \mathbf{L}$: by construction of v resp. f
 - ▶ $X = \neg A$ for $A \in \mathbf{L}$: if $\neg A \in H$ then $v(\neg A) = \text{f}$ by construction of f ; then $v(\neg A) = \text{t}$ since v is a Boolean valuation
 - ▶ $X = \perp$ and $X = \neg \top$: $X \notin H$ by definition of Hintikka sets, nothing to show
 - ▶ $X = \neg \perp$: $v(\perp) = \text{f}$ and $v(\neg \perp) = \neg v(\perp) = \text{t}$ by def. of v
 - ▶ $X = \top$: $v(\top) = \text{t}$ by definition of v

Proposition — Hintikka's Lemma

63

Every propositional Hintikka set H is satisfiable

Proof: We produce a valuation v so that $v(Z) = \text{t}$ for all $Z \in H$:

- ▶ let f be the following mapping:
 $f(A) = \text{t}$ if $A \in H$; $f(A) = \text{f}$ if $\neg A \in H$; $f(A) = \text{f}$ if $A \notin H$
- ▶ f is well-defined by condition 1 of Hintikka sets
- ▶ let v be the unique Boolean valuation extending f
- ▶ we prove by (modified) structural induction that for all $X \in \mathbf{P}$ we have: either $X \notin H$ or else $v(X) = \text{t}$
 - ▶ base case
 - ▶ $X = A$ for $A \in \mathbf{L}$: by construction of v resp. f
 - ▶ $X = \neg A$ for $A \in \mathbf{L}$: if $\neg A \in H$ then $v(A) = \text{f}$ by construction of f ; then $v(\neg A) = \text{t}$ since v is a Boolean valuation
 - ▶ $X = \perp$ and $X = \neg \top$: $X \notin H$ by definition of Hintikka sets, nothing to show
 - ▶ $X = \neg \perp$: $v(\perp) = \text{f}$ and $v(\neg \perp) = \neg v(\perp) = \text{t}$ by def. of v
 - ▶ $X = \top$: $v(\top) = \text{t}$ by definition of v

Lemma — Hintikka's Lemma

64

Every propositional Hintikka set H is satisfiable.

Proof (continued):

► step case

- ▶ $X = \neg\neg Z$: assume $\neg\neg Z \in H$ (otherwise nothing to show); then $Z \in H$ since H is Hintikka set; by induction hypothesis $v(Z) = \text{t}$; then $v(\neg\neg Z) = \text{t}$ since v is Boolean valuation
- ▶ $X = \alpha$: assume $X \in H$ (otherwise nothing to show); then $\alpha_1 \in H$ and $\alpha_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\alpha_1) = v(\alpha_2) = \text{t}$; then $v(\alpha) = \text{t}$ by Uniform Notation Lemma 1.
- ▶ $X = \beta$: assume $X \in H$ (otherwise nothing to show); then $\beta_1 \in H$ or $\beta_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; then $v(\beta) = \text{t}$ by Uniform Notation Lemma 1.

Lemma — Hintikka's Lemma

64

Every propositional Hintikka set H is satisfiable.

Proof (continued):

- ▶ step case
 - ▶ $X = \neg\neg Z$: assume $\neg\neg Z \in H$ (otherwise nothing to show); then $Z \in H$ since H is Hintikka set; by induction hypothesis $v(Z) = \text{t}$; then $v(\neg\neg Z) = \text{t}$ since v is Boolean valuation
 - ▶ $X = \alpha$: assume $X \in H$ (otherwise nothing to show); then $\alpha_1 \in H$ and $\alpha_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\alpha_1) = v(\alpha_2) = \text{t}$; then $v(\alpha) = \text{t}$ by Uniform Notation Lemma 1.
 - ▶ $X = \beta$: assume $X \in H$ (otherwise nothing to show); then $\beta_1 \in H$ or $\beta_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; then $v(\beta) = \text{t}$ by Uniform Notation Lemma 1.

Lemma — Hintikka's Lemma

64

Every propositional Hintikka set H is satisfiable.

Proof (continued):

- ▶ step case
 - ▶ $X = \neg\neg Z$: assume $\neg\neg Z \in H$ (otherwise nothing to show); then $Z \in H$ since H is Hintikka set; by induction hypothesis $v(Z) = \text{t}$; then $v(\neg\neg Z) = \text{t}$ since v is Boolean valuation
 - ▶ $X = \alpha$: assume $X \in H$ (otherwise nothing to show); then $\alpha_1 \in H$ and $\alpha_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\alpha_1) = v(\alpha_2) = \text{t}$; then $v(\alpha) = \text{t}$ by Uniform Notation Lemma 1.
 - ▶ $X = \beta$: assume $X \in H$ (otherwise nothing to show); then $\beta_1 \in H$ or $\beta_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; then $v(\beta) = \text{t}$ by Uniform Notation Lemma 1.

Lemma — Hintikka's Lemma

64

Every propositional Hintikka set H is satisfiable.

Proof (continued):

- ▶ step case
 - ▶ $X = \neg\neg Z$: assume $\neg\neg Z \in H$ (otherwise nothing to show); then $Z \in H$ since H is Hintikka set; by induction hypothesis $v(Z) = \text{t}$; then $v(\neg\neg Z) = \text{t}$ since v is Boolean valuation
 - ▶ $X = \alpha$: assume $X \in H$ (otherwise nothing to show); then $\alpha_1 \in H$ and $\alpha_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\alpha_1) = v(\alpha_2) = \text{t}$; then $v(\alpha) = \text{t}$ by Uniform Notation Lemma 1.
 - ▶ $X = \beta$: assume $X \in H$ (otherwise nothing to show); then $\beta_1 \in H$ or $\beta_2 \in H$ since H is Hintikka set; by induction hypothesis $v(\beta_1) = \text{t}$ or $v(\beta_2) = \text{t}$; then $v(\beta) = \text{t}$ by Uniform Notation Lemma 1.

Proposition — Consistency

65

If H is a propositional Hintikka set, then for all $X \in \mathbf{P}$ not both $X \in H$ and $\neg X \in H$.

Proof: follows from previous lemma

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Model Existence Theorem / Abstract Consistency Property

Idea:

- ▶ powerful technique; employs/extends Hintikka sets
- ▶ applicable to prove completeness for various calculi
- ▶ applicable to prove various other meta theorems
- ▶ scales well for first-order logic and higher-order logic
- ▶ notion of abstract consistency is used to construct Boolean valuations
- ▶ model existence theorem proved only once and for all
- ▶ the actual completeness prove for a proof system S is the relatively simple: verify that S meets the abstract consistency conditions
- ▶ model existence theorem and the machinery illustrated below then automatically applies and we get completeness for 'free'

Note: We will consider now collections of sets of formulas; an abstract consistency property \mathcal{C} is defined as such a collection of sets of formulas. If a set of formulas $S \subseteq \mathbf{P}$ is in \mathcal{C} then we say that S is \mathcal{C} -consistent.

Definition — Propositional Consistency Property

66

Let \mathcal{C} be a collection of sets of formulas. \mathcal{C} is a *propositional consistency property* if for each $S \in \mathcal{C}$:

∇_c : for any $A \in \mathbf{L}$, not both $A \in S$ and $\neg A \in S$

∇_T : $\perp \notin S$ and $\neg T \notin S$

$\nabla_{\neg\neg}$: if $\neg\neg Z \in S$ then $S \cup \{Z\} \in \mathcal{C}$

∇_α : if $\alpha \in S$ then $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$

∇_β : if $\beta \in S$ then $S \cup \{\beta_1\} \in \mathcal{C}$ or $S \cup \{\beta_2\} \in \mathcal{C}$

Note: We will consider now collections of sets of formulas; an abstract consistency property \mathcal{C} is defined as such a collection of sets of formulas. If a set of formulas $S \subseteq \mathbf{P}$ is in \mathcal{C} then we say that S is \mathcal{C} -consistent.

Definition — Propositional Consistency Property

66

Let \mathcal{C} be a collection of sets of formulas. \mathcal{C} is a *propositional consistency property* if for each $S \in \mathcal{C}$:

∇_{\exists} : for any $A \in \mathbf{L}$, not both $A \in S$ and $\neg A \in S$

∇_{\top} : $\perp \notin S$ and $\neg \top \notin S$

$\nabla_{\neg\neg}$: if $\neg\neg Z \in S$ then $S \cup \{Z\} \in \mathcal{C}$

∇_α : if $\alpha \in S$ then $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$

∇_β : if $\beta \in S$ then $S \cup \{\beta_1\} \in \mathcal{C}$ or $S \cup \{\beta_2\} \in \mathcal{C}$

Note: We will consider now collections of sets of formulas; an abstract consistency property \mathcal{C} is defined as such a collection of sets of formulas. If a set of formulas $S \subseteq \mathbf{P}$ is in \mathcal{C} then we say that S is \mathcal{C} -consistent.

Definition — Propositional Consistency Property

66

Let \mathcal{C} be a collection of sets of formulas. \mathcal{C} is a *propositional consistency property* if for each $S \in \mathcal{C}$:

∇_c : for any $A \in \mathbf{L}$, not both $A \in S$ and $\neg A \in S$

∇_{\perp} : $\perp \notin S$ and $\top \notin S$

$\nabla_{\neg\neg}$: if $\neg\neg Z \in S$ then $S \cup \{Z\} \in \mathcal{C}$

∇_α : if $\alpha \in S$ then $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$

∇_β : if $\beta \in S$ then $S \cup \{\beta_1\} \in \mathcal{C}$ or $S \cup \{\beta_2\} \in \mathcal{C}$

Note: We will consider now collections of sets of formulas; an abstract consistency property \mathcal{C} is defined as such a collection of sets of formulas. If a set of formulas $S \subseteq \mathbf{P}$ is in \mathcal{C} then we say that S is \mathcal{C} -consistent.

Definition — Propositional Consistency Property

66

Let \mathcal{C} be a collection of sets of formulas. \mathcal{C} is a *propositional consistency property* if for each $S \in \mathcal{C}$:

∇_c : for any $A \in \mathbf{L}$, not both $A \in S$ and $\neg A \in S$

∇_{\perp} : $\perp \notin S$ and $\top \notin S$

$\nabla_{\neg\neg}$: if $\neg\neg Z \in S$ then $S \cup \{Z\} \in \mathcal{C}$

∇_α : if $\alpha \in S$ then $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$

∇_β : if $\beta \in S$ then $S \cup \{\beta_1\} \in \mathcal{C}$ or $S \cup \{\beta_2\} \in \mathcal{C}$

Note: We will consider now collections of sets of formulas; an abstract consistency property \mathcal{C} is defined as such a collection of sets of formulas. If a set of formulas $S \subseteq \mathbf{P}$ is in \mathcal{C} then we say that S is \mathcal{C} -consistent.

Definition — Propositional Consistency Property

66

Let \mathcal{C} be a collection of sets of formulas. \mathcal{C} is a *propositional consistency property* if for each $S \in \mathcal{C}$:

∇_c : for any $A \in \mathbf{L}$, not both $A \in S$ and $\neg A \in S$

∇_{\perp} : $\perp \notin S$ and $\top \notin S$

$\nabla_{\neg\neg}$: if $\neg\neg Z \in S$ then $S \cup \{Z\} \in \mathcal{C}$

∇_α : if $\alpha \in S$ then $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$

∇_β : if $\beta \in S$ then $S \cup \{\beta_1\} \in \mathcal{C}$ or $S \cup \{\beta_2\} \in \mathcal{C}$

Note: We will consider now collections of sets of formulas; an abstract consistency property \mathcal{C} is defined as such a collection of sets of formulas. If a set of formulas $S \subseteq \mathbf{P}$ is in \mathcal{C} then we say that S is \mathcal{C} -consistent.

Definition — Propositional Consistency Property

66

Let \mathcal{C} be a collection of sets of formulas. \mathcal{C} is a *propositional consistency property* if for each $S \in \mathcal{C}$:

∇_c : for any $A \in \mathbf{L}$, not both $A \in S$ and $\neg A \in S$

∇_{\perp} : $\perp \notin S$ and $\top \notin S$

$\nabla_{\neg\neg}$: if $\neg\neg Z \in S$ then $S \cup \{Z\} \in \mathcal{C}$

∇_α : if $\alpha \in S$ then $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$

∇_β : if $\beta \in S$ then $S \cup \{\beta_1\} \in \mathcal{C}$ or $S \cup \{\beta_2\} \in \mathcal{C}$

Note: We will consider now collections of sets of formulas; an abstract consistency property \mathcal{C} is defined as such a collection of sets of formulas. If a set of formulas $S \subseteq \mathbf{P}$ is in \mathcal{C} then we say that S is \mathcal{C} -consistent.

Definition — Propositional Consistency Property

66

Let \mathcal{C} be a collection of sets of formulas. \mathcal{C} is a *propositional consistency property* if for each $S \in \mathcal{C}$:

∇_c : for any $A \in \mathbf{L}$, not both $A \in S$ and $\neg A \in S$

∇_{\perp} : $\perp \notin S$ and $\top \notin S$

$\nabla_{\neg\neg}$: if $\neg\neg Z \in S$ then $S \cup \{Z\} \in \mathcal{C}$

∇_α : if $\alpha \in S$ then $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$

∇_β : if $\beta \in S$ then $S \cup \{\beta_1\} \in \mathcal{C}$ or $S \cup \{\beta_2\} \in \mathcal{C}$

Example — Abstract Consistency Class

67

- ▶ Is this an abstract consistency class?

$\{\{\neg(\neg A \vee B), \neg\neg A, C\}, \{\neg(\neg A \vee B)\}, \{\neg\neg A\}, \{\}\}$

Definition — Subset Closed

68

Let \mathcal{C} be a propositional consistency property. \mathcal{C} is *subset closed* if for all $S \in \mathcal{C}$ holds: if $S' \subseteq S$ then $S' \in \mathcal{C}$.

Lemma

69

Every propositional consistency property \mathcal{C} can be extended to one that is subset closed.

Proof: ... exercise; consider $\mathcal{C}' = \{S' \mid S' \subseteq S \text{ and } S \in \mathcal{C}\} \dots$

Definition — Subset Closed

68

Let \mathcal{C} be a propositional consistency property. \mathcal{C} is *subset closed* if for all $S \in \mathcal{C}$ holds: if $S' \subseteq S$ then $S' \in \mathcal{C}$.

Lemma

69

Every propositional consistency property \mathcal{C} can be extended to one that is subset closed.

Proof: ... exercise; consider $\mathcal{C}' = \{S' \mid S' \subseteq S \text{ and } S \in \mathcal{C}\} \dots$

Definition — Subset Closed

68

Let \mathcal{C} be a propositional consistency property. \mathcal{C} is *subset closed* if for all $S \in \mathcal{C}$ holds: if $S' \subseteq S$ then $S' \in \mathcal{C}$.

Lemma

69

Every propositional consistency property \mathcal{C} can be extended to one that is subset closed.

Proof: ... exercise; consider $\mathcal{C}' = \{S' \mid S' \subseteq S \text{ and } S \in \mathcal{C}\} \dots$

Example — Subset Closed

70

- ▶ not closed under subsets:
 $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- ▶ closed under subsets:
 $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- ▶ Can you turn the example from the previous slide into a subset closed abstract consistency class?

Example — Subset Closed

70

- ▶ not closed under subsets:
 $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- ▶ closed under subsets:
 $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- ▶ Can you turn the example from the previous slide into a subset closed abstract consistency class?

Example — Subset Closed

70

- ▶ not closed under subsets:
 $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- ▶ closed under subsets:
 $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- ▶ Can you turn the example from the previous slide into a subset closed abstract consistency class?

Definition — Finite Character

71

Let \mathcal{C} be an propositional consistency property. \mathcal{C} is of *finite character* if we have: $S \in \mathcal{C}$ if and only if every finite subset S' of S is in \mathcal{C} .

Lemma

72

Every propositional consistency property \mathcal{C} of finite character is subset closed.

Proof: Let $T \in \mathcal{C}$ and $S \subseteq T$. We have to show that $S \in \mathcal{C}$. Every finite subset A of S is also a finite subset of T . Since \mathcal{C} is of finite character and $T \in \mathcal{C}$ we get that all $A \in \mathcal{C}$. Thus, $S \in \mathcal{C}$ since \mathcal{C} is of finite character.

Definition — Finite Character

71

Let \mathcal{C} be an propositional consistency property. \mathcal{C} is of *finite character* if we have: $S \in \mathcal{C}$ if and only if every finite subset S' of S is in \mathcal{C} .

Lemma

72

Every propositional consistency property \mathcal{C} of finite character is subset closed.

Proof: Let $T \in \mathcal{C}$ and $S \subseteq T$. We have to show that $S \in \mathcal{C}$. Every finite subset A of S is also a finite subset of T . Since \mathcal{C} is of finite character and $T \in \mathcal{C}$ we get that all $A \in \mathcal{C}$. Thus, $S \in \mathcal{C}$ since \mathcal{C} is of finite character.

Definition — Finite Character

71

Let \mathcal{C} be an propositional consistency property. \mathcal{C} is of *finite character* if we have: $S \in \mathcal{C}$ if and only if every finite subset S' of S is in \mathcal{C} .

Lemma

72

Every propositional consistency property \mathcal{C} of finite character is subset closed.

Proof: Let $T \in \mathcal{C}$ and $S \subseteq T$. We have to show that $S \in \mathcal{C}$. Every finite subset A of S is also a finite subset of T . Since \mathcal{C} is of finite character and $T \in \mathcal{C}$ we get that all $A \in \mathcal{C}$. Thus, $S \in \mathcal{C}$ since \mathcal{C} is of finite character.

Definition — Finite Character

71

Let \mathcal{C} be an propositional consistency property. \mathcal{C} is of *finite character* if we have: $S \in \mathcal{C}$ if and only if every finite subset S' of S is in \mathcal{C} .

Lemma

72

Every propositional consistency property \mathcal{C} of finite character is subset closed.

Proof: Let $T \in \mathcal{C}$ and $S \subseteq T$. We have to show that $S \in \mathcal{C}$. Every finite subset A of S is also a finite subset of T . Since \mathcal{C} is of finite character and $T \in \mathcal{C}$ we get that all $A \in \mathcal{C}$. Thus, $S \in \mathcal{C}$ since \mathcal{C} is of finite character.

Definition — Finite Character

71

Let \mathcal{C} be an propositional consistency property. \mathcal{C} is of *finite character* if we have: $S \in \mathcal{C}$ if and only if every finite subset S' of S is in \mathcal{C} .

Lemma

72

Every propositional consistency property \mathcal{C} of finite character is subset closed.

Proof: Let $T \in \mathcal{C}$ and $S \subseteq T$. We have to show that $S \in \mathcal{C}$. Every finite subset A of S is also a finite subset of T . Since \mathcal{C} is of finite character and $T \in \mathcal{C}$ we get that all $A \in \mathcal{C}$. Thus, $S \in \mathcal{C}$ since \mathcal{C} is of finite character.

Definition — Finite Character

71

Let \mathcal{C} be an propositional consistency property. \mathcal{C} is of *finite character* if we have: $S \in \mathcal{C}$ if and only if every finite subset S' of S is in \mathcal{C} .

Lemma

72

Every propositional consistency property \mathcal{C} of finite character is subset closed.

Proof: Let $T \in \mathcal{C}$ and $S \subseteq T$. We have to show that $S \in \mathcal{C}$. Every finite subset A of S is also a finite subset of T . Since \mathcal{C} is of finite character and $T \in \mathcal{C}$ we get that all $A \in \mathcal{C}$. Thus, $S \in \mathcal{C}$ since \mathcal{C} is of finite character.

Example — Finite Character

73

We define two classes of sets

- ▶ $C := \{\phi \mid \phi \text{ is finite subset of } \mathbb{N}\}$
- ▶ $D := 2^{\mathbb{N}}$

C is closed under subsets but **not** of finite character.

D is closed under subsets **and** of finite character.

Lemma

74

Every subset closed propositional consistency property \mathcal{C} can be extended to one that is of finite character.

Proof: ... exercise; consider

$\mathcal{C}' = \{S \mid \text{all finite subsets of } S \text{ are in } \mathcal{C}\} \dots$

Example — Finite Character

73

We define two classes of sets

- ▶ $C := \{\phi \mid \phi \text{ is finite subset of } \mathbb{N}\}$
- ▶ $D := 2^{\mathbb{N}}$

C is closed under subsets but **not** of finite character.

D is closed under subsets **and** of finite character.

Lemma

74

Every subset closed propositional consistency property \mathcal{C} can be extended to one that is of finite character.

Proof: ... exercise; consider

$\mathcal{C}' = \{S \mid \text{all finite subsets of } S \text{ are in } \mathcal{C}\} \dots$

Example — Finite Character

73

We define two classes of sets

- ▶ $C := \{\phi \mid \phi \text{ is finite subset of } \mathbb{N}\}$
- ▶ $D := 2^{\mathbb{N}}$

C is closed under subsets but **not** of finite character.

D is closed under subsets **and** of finite character.

Lemma

74

Every subset closed propositional consistency property \mathcal{C} can be extended to one that is of finite character.

Proof: . . . exercise; consider

$\mathcal{C}' = \{S \mid \text{all finite subsets of } S \text{ are in } \mathcal{C}\} \dots$

Lemma — Limits

75

Let \mathcal{C} be a propositional consistency property of finite character. Moreover, let S_1, S_2, S_3, \dots be a sequence of members of \mathcal{C} , such that $S_1 \subseteq S_2 \subseteq \dots \subseteq S_n$. Then $\bigcup_i S_i \in \mathcal{C}$.

Proof: It suffices to show that each finite subset $\{A_1, \dots, A_k\}$ of $\bigcup_i S_i$ is in \mathcal{C} . For each $A_i (1 \leq i \leq k)$ we have $A_i \in S_{n_i}$ for some smallest integer n_i . Let $N = \max\{n_1, \dots, n_k\}$. It must hold that $A_i \in S_N$. Since $S_N \in \mathcal{C}$ and since \mathcal{C} is subset closed we know that $\{A_1, \dots, A_k\} \in \mathcal{C}$.

Lemma — Limits

75

Let \mathcal{C} be a propositional consistency property of finite character. Moreover, let S_1, S_2, S_3, \dots be a sequence of members of \mathcal{C} , such that $S_1 \subseteq S_2 \subseteq \dots \subseteq S_n$. Then $\bigcup_i S_i \in \mathcal{C}$.

Proof: It suffices to show that each finite subset $\{A_1, \dots, A_k\}$ of $\bigcup_i S_i$ is in \mathcal{C} . For each $A_i (1 \leq i \leq k)$ we have $A_i \in S_{n_i}$ for some smallest integer n_i . Let $N = \max\{n_1, \dots, n_k\}$. It must hold that $A_i \in S_N$. Since $S_N \in \mathcal{C}$ and since \mathcal{C} is subset closed we know that $\{A_1, \dots, A_k\} \in \mathcal{C}$.

Lemma — Limits

75

Let \mathcal{C} be a propositional consistency property of finite character. Moreover, let S_1, S_2, S_3, \dots be a sequence of members of \mathcal{C} , such that $S_1 \subseteq S_2 \subseteq \dots \subseteq S_n$. Then $\bigcup_i S_i \in \mathcal{C}$.

Proof: It suffices to show that each finite subset $\{A_1, \dots, A_k\}$ of $\bigcup_i S_i$ is in \mathcal{C} . For each $A_i (1 \leq i \leq k)$ we have $A_i \in S_{n_i}$ for some smallest integer n_i . Let $N = \max\{n_1, \dots, n_k\}$. It must hold that $A_i \in S_N$. Since $S_N \in \mathcal{C}$ and since \mathcal{C} is subset closed we know that $\{A_1, \dots, A_k\} \in \mathcal{C}$.

Lemma — Limits

75

Let \mathcal{C} be a propositional consistency property of finite character. Moreover, let S_1, S_2, S_3, \dots be a sequence of members of \mathcal{C} , such that $S_1 \subseteq S_2 \subseteq \dots \subseteq S_n$. Then $\bigcup_i S_i \in \mathcal{C}$.

Proof: It suffices to show that each finite subset $\{A_1, \dots, A_k\}$ of $\bigcup_i S_i$ is in \mathcal{C} . For each $A_i (1 \leq i \leq k)$ we have $A_i \in S_{n_i}$ for some smallest integer n_i . Let $N = \max\{n_1, \dots, n_k\}$. It must hold that $A_i \in S_N$. Since $S_N \in \mathcal{C}$ and since \mathcal{C} is subset closed we know that $\{A_1, \dots, A_k\} \in \mathcal{C}$.

Lemma — Limits

75

Let \mathcal{C} be a propositional consistency property of finite character. Moreover, let S_1, S_2, S_3, \dots be a sequence of members of \mathcal{C} , such that $S_1 \subseteq S_2 \subseteq \dots \subseteq S_n$. Then $\bigcup_i S_i \in \mathcal{C}$.

Proof: It suffices to show that each finite subset $\{A_1, \dots, A_k\}$ of $\bigcup_i S_i$ is in \mathcal{C} . For each $A_i (1 \leq i \leq k)$ we have $A_i \in S_{n_i}$ for some smallest integer n_i . Let $N = \max\{n_1, \dots, n_k\}$. It must hold that $A_i \in S_N$. Since $S_N \in \mathcal{C}$ and since \mathcal{C} is subset closed we know that $\{A_1, \dots, A_k\} \in \mathcal{C}$.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Theorem — Propositional Model Existence

76

Let \mathcal{C} be a propositional consistency property. If $S \in \mathcal{C}$ then S is satisfiable.

Proof: By our previous Lemmata we may assume that \mathcal{C} is of finite character (if \mathcal{C} is not, we may simply extend it into \mathcal{C}' of finite character). Let X_1, \dots, X_n, \dots be an enumeration of the propositional formulas in some fixed order (note that we did require that \mathbf{L} is countable). We now consider the following sequence S_1, S_2, S_3, \dots of members of \mathcal{C}

$$S_1 = S$$

$$S_{n+1} = \begin{cases} S_n \cup \{X_n\} & \text{if } S_n \cup \{X_n\} \in \mathcal{C} \\ S_n & \text{otherwise} \end{cases}$$

Note that for all S_n holds: $S_n \in \mathcal{C}$ and $S_n \subseteq S_{n+1}$. Hence $H = \bigcup_n S_n$ (which extends S) is in \mathcal{C} by our previous Lemma.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Proof (continued):

It is easy to check that H is maximal in \mathcal{C} , i.e. if $H \subseteq K$ for some $K \in \mathcal{C}$ then $H = K$.

It is also easy to check that H is a Hintikka set: We exemplary consider case α (the others are similar): Suppose $\alpha \in H$; we need to show that $\alpha_1, \alpha_2 \in H$. Since $\alpha \in H$ we have $H \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ by ∇_α . $H \cup \{\alpha_1, \alpha_2\}$ extends H which is maximal in \mathcal{C} , hence it must be identical to H . Thus, $\alpha_1, \alpha_2 \in H$.

Since H is an Hintikka set, we can apply the Hintikka Lemma. It follows that H is satisfiable. Hence, $S \subseteq H$ is satisfiable.

Theorem — Compactness Theorem

77

Let $S \subseteq \mathbf{P}$ be a set of propositional formulas. S is satisfiable if every finite subset of S is satisfiable.

Proof: Assume every finite subset of S is satisfiable. Define \mathcal{C} as follows:

$$\mathcal{C} = \{W \mid \text{every finite subset of } W \text{ is satisfiable}\}$$

Trivially $S \in \mathcal{C}$. Hence, if we can show that \mathcal{C} is a propositional consistency property then we are done: it immediately follows by the Model Existence Theorem that S is satisfiable.

To prove that \mathcal{C} is a propositional consistency property we need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

Theorem — Compactness Theorem

77

Let $S \subseteq \mathbf{P}$ be a set of propositional formulas. S is satisfiable if every finite subset of S is satisfiable.

Proof: Assume every finite subset of S is satisfiable. Define \mathcal{C} as follows:

$$\mathcal{C} = \{W \mid \text{every finite subset of } W \text{ is satisfiable}\}$$

Trivially $S \in \mathcal{C}$. Hence, if we can show that \mathcal{C} is a propositional consistency property then we are done: it immediately follows by the Model Existence Theorem that S is satisfiable.

To prove that \mathcal{C} is a propositional consistency property we need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

Theorem — Compactness Theorem

77

Let $S \subseteq \mathbf{P}$ be a set of propositional formulas. S is satisfiable if every finite subset of S is satisfiable.

Proof: Assume every finite subset of S is satisfiable. Define \mathcal{C} as follows:

$$\mathcal{C} = \{W \mid \text{every finite subset of } W \text{ is satisfiable}\}$$

Trivially $S \in \mathcal{C}$. Hence, if we can show that \mathcal{C} is a propositional consistency property then we are done: it immediately follows by the Model Existence Theorem that S is satisfiable.

To prove that \mathcal{C} is a propositional consistency property we need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

Theorem — Compactness Theorem

77

Let $S \subseteq \mathbf{P}$ be a set of propositional formulas. S is satisfiable if every finite subset of S is satisfiable.

Proof: Assume every finite subset of S is satisfiable. Define \mathcal{C} as follows:

$$\mathcal{C} = \{W \mid \text{every finite subset of } W \text{ is satisfiable}\}$$

Trivially $S \in \mathcal{C}$. Hence, if we can show that \mathcal{C} is a propositional consistency property then we are done: it immediately follows by the Model Existence Theorem that S is satisfiable.

To prove that \mathcal{C} is a propositional consistency property we need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

Theorem — Compactness Theorem

77

Let $S \subseteq \mathbf{P}$ be a set of propositional formulas. S is satisfiable if every finite subset of S is satisfiable.

Proof: Assume every finite subset of S is satisfiable. Define \mathcal{C} as follows:

$$\mathcal{C} = \{W \mid \text{every finite subset of } W \text{ is satisfiable}\}$$

Trivially $S \in \mathcal{C}$. Hence, if we can show that \mathcal{C} is a propositional consistency property then we are done: it immediately follows by the Model Existence Theorem that S is satisfiable.

To prove that \mathcal{C} is a propositional consistency property we need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

Theorem — Compactness Theorem

77

Let $S \subseteq \mathbf{P}$ be a set of propositional formulas. S is satisfiable if every finite subset of S is satisfiable.

Proof: Assume every finite subset of S is satisfiable. Define \mathcal{C} as follows:

$$\mathcal{C} = \{W \mid \text{every finite subset of } W \text{ is satisfiable}\}$$

Trivially $S \in \mathcal{C}$. Hence, if we can show that \mathcal{C} is a propositional consistency property then we are done: it immediately follows by the Model Existence Theorem that S is satisfiable.

To prove that \mathcal{C} is a propositional consistency property we need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$.

Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

$\nabla_\perp, \nabla_{\neg\neg}, \nabla_\beta$: similar ... exercise ...

Proof (continued):

∇_c : Assume $W \in \mathcal{C}$. Suppose for some $A \in \mathbf{L}$ both $A \in W$ and $\neg A \in W$. $\{A, \neg A\}$ is a finite subset of W . This set is, however, not satisfiable. This contradicts the assumption $W \in \mathcal{C}$. Hence, we can not have both $A \in W$ and $\neg A \in W$.

∇_α : Assume $W \in \mathcal{C}$ and $\alpha \in W$. We show that every finite subset of $W \cup \{\alpha_1, \alpha_2\}$ is satisfiable and hence that $W \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$. Let W' be a finite subset of W . Case 1: $\alpha_1, \alpha_2 \notin W'$. Then W' is satisfiable since $W \in \mathcal{C}$. Case 2: $W' = W_o \cup \{\alpha_1, \alpha_2\}$, where W_o is finite. Then $W_o \cup \{\alpha\}$ is a finite subset of W , and hence satisfiable. Hence there is some v such that $v(\alpha) = \text{t}$. By the Uniform Notation Lemma 1 we then know $v(\alpha_1) = v(\alpha_2) = \text{t}$. Hence, $W_o \cup \{\alpha, \alpha_1, \alpha_2\}$ is satisfiable. Hence, $W_o \cup \{\alpha_1, \alpha_2\}$ is satisfiable. Case 3: $W' = W_o \cup \{\alpha_1\}$ similar. Case 4: $W' = W_o \cup \{\alpha_2\}$ similar.

∇_\perp , $\nabla_{\neg\neg}$, ∇_β : similar ... exercise ...

Definition

78

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *tableau consistent* if there is no closed tableau for S .

Lemma

79

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} | S \text{ is tableau consistent}\}$ of all tableau consistent sets is a Propositional Consistency Property.

Proof: We need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

∇_c : Assume $A, \neg A \in S$. Then there is a closed tableau for S . This contradicts assumption $S \in \mathcal{C}$.

∇_{\perp} : similar ... exercise ...

Definition

78

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *tableau consistent* if there is no closed tableau for S .

Lemma

79

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} \mid S \text{ is tableau consistent}\}$ of all tableau consistent sets is a *Propositional Consistency Property*.

Proof: We need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

∇_c : Assume $A, \neg A \in S$. Then there is a closed tableau for S . This contradicts assumption $S \in \mathcal{C}$.

∇_{\perp} : similar ... exercise ...

Definition

78

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *tableau consistent* if there is no closed tableau for S .

Lemma

79

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} \mid S \text{ is tableau consistent}\}$ of all tableau consistent sets is a *Propositional Consistency Property*.

Proof: We need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

∇_c : Assume $A, \neg A \in S$. Then there is a closed tableau for S . This contradicts assumption $S \in \mathcal{C}$.

∇_{\perp} : similar ... exercise ...

Definition

78

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *tableau consistent* if there is no closed tableau for S .

Lemma

79

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} \mid S \text{ is tableau consistent}\}$ of all tableau consistent sets is a *Propositional Consistency Property*.

Proof: We need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

∇_c : Assume $A, \neg A \in S$. Then there is a closed tableau for S . This contradicts assumption $S \in \mathcal{C}$.

∇_{\perp} : similar ... exercise ...

Definition

78

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *tableau consistent* if there is no closed tableau for S .

Lemma

79

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} \mid S \text{ is tableau consistent}\}$ of all tableau consistent sets is a *Propositional Consistency Property*.

Proof: We need to verify the conditions $\nabla_c, \nabla_{\perp}, \nabla_{\neg\neg}, \nabla_{\alpha}, \nabla_{\beta}$.

∇_c : Assume $A, \neg A \in S$. Then there is a closed tableau for S . This contradicts assumption $S \in \mathcal{C}$.

∇_{\perp} : similar . . . exercise . . .

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

α

X_1

:

X_n

α_1

α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

α

X_1

:

X_n

α_1

α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

α

X_1

:

X_n

α_1

α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

 α X_1 \vdots X_n α_1 α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

α

X_1

:

X_n

α_1

α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

 α X_1 \vdots X_n α_1 α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

 α X_1 \vdots X_n α_1 α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Proof (continued):

∇_α : Assume $\alpha \in S$. Instead of showing $S \in \mathcal{C}$ implies $S \cup \{\alpha_1, \alpha_2\} \in \mathcal{C}$ we show the contrapositive: assume $S \cup \{\alpha_1, \alpha_2\} \notin \mathcal{C}$, then show $S \notin \mathcal{C}$.

If $S \cup \{\alpha_1, \alpha_2\}$ is not tableau consistent then there is a closed tableau for it. Let's say $S = \{\alpha, X_1, \dots, X_n\}$. Then there must be closed tableau for $S \cup \{\alpha_1, \alpha_2\}$ of the following form:

 α X_1 \vdots X_n α_1 α_2

'rest of the closed tableau'

To show that S is not tableau consistent is easy: simply use the α -rule to produce α_1 and α_2 and reuse the 'rest of the closed tableau'.

Theorem — Completeness for Propositional Tableau

80

If $X \in \mathbf{P}$ is a tautology, then X has a tableau proof.

Proof: (we show the contrapositive)

Suppose X does not have tableau proof. This means that there is no closed tableau for $\{\neg X\}$, i.e. $\{\neg X\}$ is tableau consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Tableau 80

If $X \in \mathbf{P}$ is a tautology, then X has a tableau proof.

Proof: (we show the contrapositive)

Suppose X does not have tableau proof. This means that there is no closed tableau for $\{\neg X\}$, i.e. $\{\neg X\}$ is tableau consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Tableau

80

If $X \in \mathbf{P}$ is a tautology, then X has a tableau proof.

Proof: (we show the contrapositive)

Suppose X does not have tableau proof. This means that there is no closed tableau for $\{\neg X\}$, i.e. $\{\neg X\}$ is tableau consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Tableau 80

If $X \in \mathbf{P}$ is a tautology, then X has a tableau proof.

Proof: (we show the contrapositive)

Suppose X does not have tableau proof. This means that there is no closed tableau for $\{\neg X\}$, i.e. $\{\neg X\}$ is tableau consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Tableau

80

If $X \in \mathbf{P}$ is a tautology, then X has a tableau proof.

Proof: (we show the contrapositive)

Suppose X does not have tableau proof. This means that there is no closed tableau for $\{\neg X\}$, i.e. $\{\neg X\}$ is tableau consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Tableau

80

If $X \in \mathbf{P}$ is a tautology, then X has a tableau proof.

Proof: (we show the contrapositive)

Suppose X does not have tableau proof. This means that there is no closed tableau for $\{\neg X\}$, i.e. $\{\neg X\}$ is tableau consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Some remarks:

- ▶ note how the technique reduces the completeness argument to some relatively easy to check local, syntactic criteria: the ∇_* -conditions.
- ▶ technique can easily be adapted for 'atomically closed tableau'
- ▶ technique does not easily extend for 'strict versions of tableau and resolution'

Definition — Resolution Derivation

81

Let S be a set of disjunctions. A *resolution derivation* from S is a sequence of disjunctions, each of which is a member of S , or comes from an earlier term in the sequence by one of the Clause Set Reduction Rules or comes from earlier terms by the Resolution Rule.

We say a disjunction D is *resolution derivable* from S if D is the last line of a resolution derivation from S .

Remark: This definition slightly generalizes the notion of resolution expansion: it allows us to start with any family of generalized disjunctions $\{[A_1], \dots, [A_n]\}$ (instead of starting from $\{A_1, \dots, A_n\}$ as in resolution expansions).

Definition — Resolution Derivation

81

Let S be a set of disjunctions. A *resolution derivation* from S is a sequence of disjunctions, each of which is a member of S , or comes from an earlier term in the sequence by one of the Clause Set Reduction Rules or comes from earlier terms by the Resolution Rule.

We say a disjunction D is *resolution derivable* from S if D is the last line of a resolution derivation from S .

Remark: This definition slightly generalizes the notion of resolution expansion: it allows us to start with any family of generalized disjunctions $\{[A_1], \dots, [A_n]\}$ (instead of starting from $\{A_1, \dots, A_n\}$ as in resolution expansions).

Definition — Resolution Derivation

81

Let S be a set of disjunctions. A *resolution derivation* from S is a sequence of disjunctions, each of which is a member of S , or comes from an earlier term in the sequence by one of the Clause Set Reduction Rules or comes from earlier terms by the Resolution Rule.

We say a disjunction D is *resolution derivable* from S if D is the last line of a resolution derivation from S .

Remark: This definition slightly generalizes the notion of resolution expansion: it allows us to start with any family of generalized disjunctions $\{[A_1], \dots, [A_n]\}$ (instead of starting from $\{A_1, \dots, A_n\}$ as in resolution expansions).

Definition — X -enlargement

82

Let $X, A_1, \dots, A_n \in \mathbf{P}$. $[X, A_1, \dots, A_n]$ and $[A_1, \dots, A_n]$ are both called **X -enlargements** of $[A_1, \dots, A_n]$.

If S is a set of disjunctions and S^* is the result of replacing each member of S by an X -enlargement, then S^* is called an **X -enlargement** of S .

Example — X -enlargement

83

$\{[A_1, A_2, X], [B_1, B_2, B_3, X]\}$ and $\{[A_1, A_2], [B_1, B_2, B_3, X]\}$ are both X -enlargements of $\{[A_1, A_2], [B_1, B_2, B_3]\}$.

Definition — X -enlargement

82

Let $X, A_1, \dots, A_n \in \mathbf{P}$. $[X, A_1, \dots, A_n]$ and $[A_1, \dots, A_n]$ are both called *X -enlargements* of $[A_1, \dots, A_n]$.

If S is a set of disjunctions and S^* is the result of replacing each member of S by an X -enlargement, then S^* is called an *X -enlargement* of S .

Example — X -enlargement

83

$\{[A_1, A_2, X], [B_1, B_2, B_3, X]\}$ and $\{[A_1, A_2], [B_1, B_2, B_3, X]\}$ are both X -enlargements of $\{[A_1, A_2], [B_1, B_2, B_3]\}$.

Definition — X -enlargement

82

Let $X, A_1, \dots, A_n \in \mathbf{P}$. $[X, A_1, \dots, A_n]$ and $[A_1, \dots, A_n]$ are both called *X -enlargements* of $[A_1, \dots, A_n]$.

If S is a set of disjunctions and S^* is the result of replacing each member of S by an X -enlargement, then S^* is called an *X -enlargement* of S .

Example — X -enlargement

83

$\{[A_1, A_2, X], [B_1, B_2, B_3, X]\}$ and $\{[A_1, A_2], [B_1, B_2, B_3, X]\}$ are both X -enlargements of $\{[A_1, A_2], [B_1, B_2, B_3]\}$.

Lemma

84

Suppose S_1 and S_2 are sets of disjunctions, and S_2 is an X -enlargement of S_1 . If the disjunction D_1 is resolution derivable from S_1 , then there is an X -enlargement D_2 of D_1 that is resolution derivable from S_2 .

Proof: ... exercise; carry along X in the derivation ...

Lemma

84

Suppose S_1 and S_2 are sets of disjunctions, and S_2 is an X -enlargement of S_1 . If the disjunction D_1 is resolution derivable from S_1 , then there is an X -enlargement D_2 of D_1 that is resolution derivable from S_2 .

Proof: . . . exercise; carry along X in the derivation . . .

Definition — Resolution Consistent

85

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *resolution consistent* if there is no closed resolution expansion for S .

Lemma

86

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} | S \text{ is resolution consistent}\}$ of all resolution consistent sets is a Propositional Consistency Property.

Proof: ... exercise ...

Definition — Resolution Consistent

85

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *resolution consistent* if there is no closed resolution expansion for S .

Lemma

86

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} | S \text{ is resolution consistent}\}$ of all resolution consistent sets is a Propositional Consistency Property.

Proof: ... exercise ...

Definition — Resolution Consistent

85

A finite set $S \subseteq \mathbf{P}$ of propositional formulas is called *resolution consistent* if there is no closed resolution expansion for S .

Lemma

86

The collection $\mathcal{C} = \{S \subseteq \mathbf{P} | S \text{ is resolution consistent}\}$ of all resolution consistent sets is a Propositional Consistency Property.

Proof: . . . exercise . . .

Theorem — Completeness for Propositional Resolution 87

If $X \in \mathbf{P}$ is a tautology, then X has a resolution proof.

Proof: (we show the contrapositive)

Suppose X does not have resolution proof. This means that there is no closed resolution expansion for $\{\neg X\}$, i.e. $\{\neg X\}$ is resolution consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Resolution 87

If $X \in \mathbf{P}$ is a tautology, then X has a resolution proof.

Proof: (we show the contrapositive)

Suppose X does not have resolution proof. This means that there is no closed resolution expansion for $\{\neg X\}$, i.e. $\{\neg X\}$ is resolution consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Resolution 87

If $X \in \mathbf{P}$ is a tautology, then X has a resolution proof.

Proof: (we show the contrapositive)

Suppose X does not have resolution proof. This means that there is no closed resolution expansion for $\{\neg X\}$, i.e. $\{\neg X\}$ is resolution consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Resolution 87

If $X \in \mathbf{P}$ is a tautology, then X has a resolution proof.

Proof: (we show the contrapositive)

Suppose X does not have resolution proof. This means that there is no closed resolution expansion for $\{\neg X\}$, i.e. $\{\neg X\}$ is resolution consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Resolution 87

If $X \in \mathbf{P}$ is a tautology, then X has a resolution proof.

Proof: (we show the contrapositive)

Suppose X does not have resolution proof. This means that there is no closed resolution expansion for $\{\neg X\}$, i.e. $\{\neg X\}$ is resolution consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Theorem — Completeness for Propositional Resolution 87

If $X \in \mathbf{P}$ is a tautology, then X has a resolution proof.

Proof: (we show the contrapositive)

Suppose X does not have resolution proof. This means that there is no closed resolution expansion for $\{\neg X\}$, i.e. $\{\neg X\}$ is resolution consistent. Hence, $\{\neg X\}$ is satisfiable by the previous Lemma and the Propositional Model Existence Theorem. Hence, X can not be a tautology.

Propositional Logic: Logical Consequence

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \mathbf{t}$ for every Boolean valuation that maps every member of
 S to \mathbf{t} . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \text{t}$ for every Boolean valuation that maps every member of
 S to t . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \mathbf{t}$ for every Boolean valuation that maps every member of
 S to \mathbf{t} . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \mathbf{t}$ for every Boolean valuation that maps every member of
 S to \mathbf{t} . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \mathbf{t}$ for every Boolean valuation that maps every member of
 S to \mathbf{t} . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \mathbf{t}$ for every Boolean valuation that maps every member of
 S to \mathbf{t} . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \mathbf{t}$ for every Boolean valuation that maps every member of
 S to \mathbf{t} . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Motivation: So far we addressed the question: Is X a tautology
In practice more relevant: Does X follow from a (possibly infinite)
set of axioms S ?

Definition — Propositional Consequence

88

$X \in \mathbf{P}$ is called a *propositional consequence* of a set $S \subseteq \mathbf{P}$ if
 $v(X) = \mathbf{t}$ for every Boolean valuation that maps every member of
 S to \mathbf{t} . We write $S \models_p X$.

Note: X is a tautology is the special case $\emptyset \models_p X$

Proposition

89

1. If $A, \neg A \in S$, then for any X we have $S \models_p X$.
2. If $S \models_p X$ and $S \subseteq S^*$, then $S^* \models_p X$
3. $S \models_p X$ if and only if $S \cup \{\neg X\}$ is not satisfiable.

Proof: easy; exercise

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Theorem

90

$S \models_p X$ if and only if there is a finite subset S_0 of S such that $S_0 \models_p X$.

Proof:

\Leftarrow : By (2.) of the above Lemma.

\Rightarrow : Suppose $S \models_p X$. By (3.) of the above Lemma we have $S \cup \{\neg X\}$ is not satisfiable. By Propositional Compactness some finite subset $S_0 \subset S \cup \{\neg X\}$ is not satisfiable. Hence $S_0 \cup \{\neg X\}$ is not satisfiable (any extension of S_0 must be unsatisfiable). By (3.) of the above Lemma we have $S_0 \models X$.

Question: How to prove $S \models X$ problems?

Definition — S -Introduction Rule for Tableau

91

Given a set of axioms S . *Any member of S can be added to the end of any branch.*

We write $S \vdash_{pt} X$ if there is a closed propositional tableau for $\{\neg X\}$ when allowing the S -Introduction Rule for Tableau to be applied with the axioms in S .

Definition — S -Introduction Rule for Resolution

92

Given a set of axioms S . *[Y] can be added as a line to a resolution expansion for any $Y \in S$.*

We write $S \vdash_{pr} X$ if there is a closed propositional resolution expansion for $\{\neg X\}$ when allowing the S -Introduction Rule for Resolution to be applied with the axioms in S .

Question: How to prove $S \models X$ problems?

Definition — S -Introduction Rule for Tableau

91

Given a set of axioms S . *Any member of S can be added to the end of any branch.*

We write $S \vdash_{pt} X$ if there is a closed propositional tableau for $\{\neg X\}$ when allowing the S -Introduction Rule for Tableau to be applied with the axioms in S .

Definition — S -Introduction Rule for Resolution

92

Given a set of axioms S . *[Y] can be added as a line to a resolution expansion for any $Y \in S$.*

We write $S \vdash_{pr} X$ if there is a closed propositional resolution expansion for $\{\neg X\}$ when allowing the S -Introduction Rule for Resolution to be applied with the axioms in S .

Question: How to prove $S \models X$ problems?

Definition — S -Introduction Rule for Tableau

91

Given a set of axioms S . *Any member of S can be added to the end of any branch.*

We write $S \vdash_{pt} X$ if there is a closed propositional tableau for $\{\neg X\}$ when allowing the S -Introduction Rule for Tableau to be applied with the axioms in S .

Definition — S -Introduction Rule for Resolution

92

Given a set of axioms S . *[Y] can be added as a line to a resolution expansion for any $Y \in S$.*

We write $S \vdash_{pr} X$ if there is a closed propositional resolution expansion for $\{\neg X\}$ when allowing the S -Introduction Rule for Resolution to be applied with the axioms in S .

Theorem — Strong Soundness and Completeness

93

For any set $S \subseteq \mathbf{P}$ and any $X \in \mathbf{P}$ we have

$S \models_p X$ if and only if $S \vdash_{pt} X$ if and only if $S \vdash_{pr} X$.

Proof: ... analogous to before; but: we need to redo the work with appropriately modified notions taking S into account; see the 'recipe' in the Fitting book ...

Theorem

94

$\langle A_1, \dots, A_n \rangle \supset X$ is a tautology if and only if $\{A_1, \dots, A_n\} \models_p X$.

Proof: ... exercise ...

Theorem — Strong Soundness and Completeness

93

For any set $S \subseteq \mathbf{P}$ and any $X \in \mathbf{P}$ we have

$S \models_p X$ if and only if $S \vdash_{pt} X$ if and only if $S \vdash_{pr} X$.

Proof: ... analogous to before; but: we need to redo the work with appropriately modified notions taking S into account; see the 'recipe' in the Fitting book ...

Theorem

94

$\langle A_1, \dots, A_n \rangle \supset X$ is a tautology if and only if $\{A_1, \dots, A_n\} \models_p X$.

Proof: ... exercise ...

Theorem — Strong Soundness and Completeness

93

For any set $S \subseteq \mathbf{P}$ and any $X \in \mathbf{P}$ we have

$S \models_p X$ if and only if $S \vdash_{pt} X$ if and only if $S \vdash_{pr} X$.

Proof: ... analogous to before; but: we need to redo the work with appropriately modified notions taking S into account; see the 'recipe' in the Fitting book ...

Theorem

94

$\langle A_1, \dots, A_n \rangle \supset X$ is a tautology if and only if $\{A_1, \dots, A_n\} \models_p X$.

Proof: ... exercise ...

Theorem — Strong Soundness and Completeness

93

For any set $S \subseteq \mathbf{P}$ and any $X \in \mathbf{P}$ we have

$S \models_p X$ if and only if $S \vdash_{pt} X$ if and only if $S \vdash_{pr} X$.

Proof: ... analogous to before; but: we need to redo the work with appropriately modified notions taking S into account; see the 'recipe' in the Fitting book ...

Theorem

94

$\langle A_1, \dots, A_n \rangle \supset X$ is a tautology if and only if $\{A_1, \dots, A_n\} \models_p X$.

Proof: ... exercise ...