



# Automatic Theorem Proving with Vampire

Martin Suda

Czech Technical University in Prague, Czech Republic

VTSA 2025, Liege, Belgium

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

- *first-order logic* as a formal language to describe the world

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

- *first-order logic* as a formal language to describe the world
- employs algorithms for logical *deduction*:  
deriving new facts that follow from facts previously established

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

- *first-order logic* as a formal language to describe the world
- employs algorithms for logical *deduction*:  
deriving new facts that follow from facts previously established
- provides a *proof* as the ultimate, indisputable witness

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

- *first-order logic* as a formal language to describe the world
- employs algorithms for logical *deduction*:  
deriving new facts that follow from facts previously established
- provides a *proof* as the ultimate, indisputable witness

## **Applications**

- new discoveries in mathematics

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

- *first-order logic* as a formal language to describe the world
- employs algorithms for logical *deduction*:  
deriving new facts that follow from facts previously established
- provides a *proof* as the ultimate, indisputable witness

## Applications

- new discoveries in mathematics
- hardware and software verification

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

- *first-order logic* as a formal language to describe the world
- employs algorithms for logical *deduction*:  
deriving new facts that follow from facts previously established
- provides a *proof* as the ultimate, indisputable witness

## Applications

- new discoveries in mathematics
- hardware and software verification
- "hammer" support in proof assistants (ITPs)

# Automatic Theorem Proving

One of the classical disciplines of *symbolic artificial intelligence*

- *first-order logic* as a formal language to describe the world
- employs algorithms for logical *deduction*:  
deriving new facts that follow from facts previously established
- provides a *proof* as the ultimate, indisputable witness

## Applications

- new discoveries in mathematics
- hardware and software verification
- "hammer" support in proof assistants (ITPs)
- cryptography and security, natural language processing
- ...

## **Vampire**

- State-of-the-art ATP for first-order logic and beyond  
(arithmetic reasoning, finite models, answer extraction, ...)

# Introducing Our Workhorse

## **Vampire**

- State-of-the-art ATP for first-order logic and beyond (arithmetic reasoning, finite models, answer extraction, ...)
- refutational / superposition-based / saturation-based

# Introducing Our Workhorse

## Vampire

- State-of-the-art ATP for first-order logic and beyond (arithmetic reasoning, finite models, answer extraction, ...)
- refutational / superposition-based / saturation-based
- regular CASC competition winner



Developed jointly with other groups: Manchester, Vienna, Southampton

# What an Automatic Theorem Prover is Expected to Do

## Input:

- a set of **axioms**  $A_1, \dots, A_n$  (first order formulas)
- a **conjecture**  $G$  (first-order formula)

# What an Automatic Theorem Prover is Expected to Do

## Input:

- a set of **axioms**  $A_1, \dots, A_n$  (first order formulas)
- a **conjecture**  $G$  (first-order formula)

## Output:

- if  $A_1, \dots, A_n \models G$ , return “yes” and

# What an Automatic Theorem Prover is Expected to Do

## Input:

- a set of **axioms**  $A_1, \dots, A_n$  (first order formulas)
- a **conjecture**  $G$  (first-order formula)

## Output:

- if  $A_1, \dots, A_n \models G$ , return “yes” and a **proof** of the fact (more on this later)

# What an Automatic Theorem Prover is Expected to Do

## Input:

- a set of **axioms**  $A_1, \dots, A_n$  (first order formulas)
- a **conjecture**  $G$  (first-order formula)

## Output:

- if  $A_1, \dots, A_n \models G$ , return “yes” and a **proof** of the fact (more on this later)
- otherwise, report “no”

# What an Automatic Theorem Prover is Expected to Do

## Input:

- a set of **axioms**  $A_1, \dots, A_n$  (first order formulas)
- a **conjecture**  $G$  (first-order formula)

## Output:

- if  $A_1, \dots, A_n \models G$ , return “yes” and a **proof** of the fact (more on this later)
- otherwise, report “no” (ideally)

# Saturation-based Theorem Proving

## The basic pieces of a modern ATP:

- refutational (i.e., proof by contradiction):  
instead of  $A_1, \dots, A_n \models G$  we show  $A_1, \dots, A_n, \neg G \models \perp$

# Saturation-based Theorem Proving

## The basic pieces of a modern ATP:

- refutational (i.e., proof by contradiction):  
instead of  $A_1, \dots, A_n \models G$  we show  $A_1, \dots, A_n, \neg G \models \perp$
- $A_1, \dots, A_n, \neg G \xrightarrow{\text{preprocessing \& clausification}} C_1, \dots, C_m$

# Saturation-based Theorem Proving

## The basic pieces of a modern ATP:

- refutational (i.e., proof by contradiction):  
instead of  $A_1, \dots, A_n \models G$  we show  $A_1, \dots, A_n, \neg G \models \perp$
- $A_1, \dots, A_n, \neg G \xrightarrow{\text{preprocessing \& clausification}} C_1, \dots, C_m$
- logical calculus *Calc* (resolution and superposition):  
derives new *conclusions* from already established *premises*

# Saturation-based Theorem Proving

## The basic pieces of a modern ATP:

- refutational (i.e., proof by contradiction):  
instead of  $A_1, \dots, A_n \models G$  we show  $A_1, \dots, A_n, \neg G \models \perp$
- $A_1, \dots, A_n, \neg G \xrightarrow{\text{preprocessing \& clausification}} C_1, \dots, C_m$
- logical calculus *Calc* (resolution and superposition):  
derives new *conclusions* from already established *premises*
- search for the empty clause  $\perp$  using *Calc*

# Saturation-based Theorem Proving

## The basic pieces of a modern ATP:

- refutational (i.e., proof by contradiction):  
instead of  $A_1, \dots, A_n \models G$  we show  $A_1, \dots, A_n, \neg G \models \perp$
- $A_1, \dots, A_n, \neg G \xrightarrow{\text{preprocessing \& clausification}} C_1, \dots, C_m$
- logical calculus *Calc* (resolution and superposition):  
derives new *conclusions* from already established *premises*
- search for the empty clause  $\perp$  using *Calc*

## Saturation:

- systematically compute a *closure* of  $C_1, \dots, C_m$  under *Calc*

# Saturation-based Theorem Proving

## The basic pieces of a modern ATP:

- refutational (i.e., proof by contradiction):  
instead of  $A_1, \dots, A_n \models G$  we show  $A_1, \dots, A_n, \neg G \models \perp$
- $A_1, \dots, A_n, \neg G \xrightarrow{\text{preprocessing \& clausification}} C_1, \dots, C_m$
- logical calculus *Calc* (resolution and superposition):  
derives new *conclusions* from already established *premises*
- search for the empty clause  $\perp$  using *Calc*

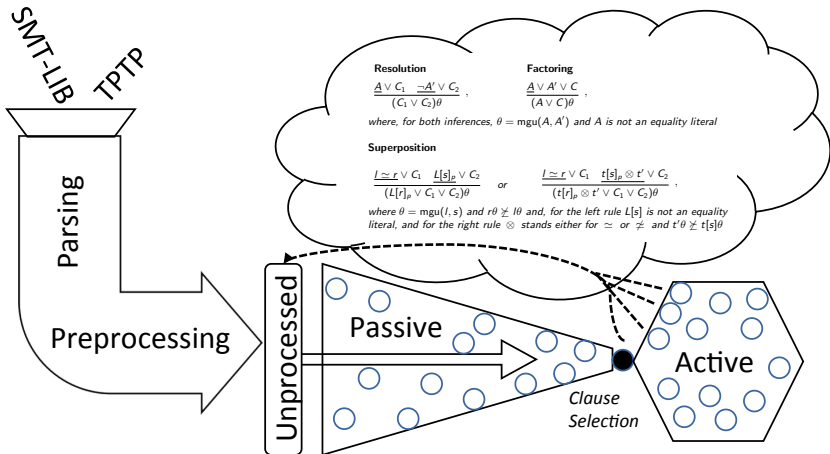
## Saturation:

- systematically compute a *closure* of  $C_1, \dots, C_m$  under *Calc*

## Given-clause algorithms:

- a particular way of organizing saturation
- maintain two sets of clauses: *Passive*, *Active*
- the next given clause: via a *clause selection* heuristic

# Saturation-based Theorem Proving in One Slide



# Obtaining Vampire

**GitHub repo:** <https://github.com/vprover/vampire>

- If you want to compile yourself
- Start from `Readme.md` and please ask if you get stuck

# Obtaining Vampire

**GitHub repo:** <https://github.com/vprover/vampire>

- If you want to compile yourself
- Start from `Readme.md` and please ask if you get stuck

**Precompiled binaries:** <https://tinyurl.com/vtsa2025vampires>

- (X46, ARM64) × (Linux, MacOS, Windows)
- hopefully, this will work for you

# Obtaining Vampire

**GitHub repo:** <https://github.com/vprover/vampire>

- If you want to compile yourself
- Start from `Readme.md` and please ask if you get stuck

**Precompiled binaries:** <https://tinyurl.com/vtsa2025vampires>

- (X46, ARM64) × (Linux, MacOS, Windows)
- hopefully, this will work for you

**TPTP:** <https://www.tptp.org/>

- Input language standard, problem library, infrastructure support
- SystemOnTPTP

# Outline

First-Order Logic as a Formal Language

Clause Normal Form in First-Order Logic

A Static View: Inferences, Soundness, and Completeness

A Dynamic View: Saturation

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.
- Function symbols of arity 0 are called **constants**.

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.
- Function symbols of arity 0 are called **constants**.
- What would predicates of arity 0 be?

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.
- Function symbols of arity 0 are called **constants**.
- What would predicates of arity 0 be?
- Countable set of **variables**  $\mathcal{V}$ , disjoint from  $\Sigma$ .

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.
- Function symbols of arity 0 are called **constants**.
- What would predicates of arity 0 be?
- Countable set of **variables**  $\mathcal{V}$ , disjoint from  $\Sigma$ .
- (Let's not talk about sorts to keep things simple for the moment.)

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.
- Function symbols of arity 0 are called **constants**.
- What would predicates of arity 0 be?
- Countable set of **variables**  $\mathcal{V}$ , disjoint from  $\Sigma$ .
- (Let's not talk about sorts to keep things simple for the moment.)

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.
- Function symbols of arity 0 are called **constants**.
- What would predicates of arity 0 be?
- Countable set of **variables**  $\mathcal{V}$ , disjoint from  $\Sigma$ .
- (Let's not talk about sorts to keep things simple for the moment.)

## Example (The Language of Groups)

$$\Sigma^G = (\Sigma_P^G, \Sigma_F^G) \quad \Sigma_F^G = \{+/2, -/1, 0/0\},$$

# First-Order Formula Syntax I

## Language

- **Signature**  $\Sigma = (\Sigma_P, \Sigma_F)$  of predicate and function **symbols**.
- Every **predicate**  $p \in \Sigma_P$  and **function**  $f \in \Sigma_F$  has a fixed **arity**.
- We always have a binary (arity=2) predicate  $\approx$  for equality.
- Function symbols of arity 0 are called **constants**.
- What would predicates of arity 0 be?
- Countable set of **variables**  $\mathcal{V}$ , disjoint from  $\Sigma$ .
- (Let's not talk about sorts to keep things simple for the moment.)

## Example (The Language of Groups)

$$\Sigma^G = (\Sigma_P^G, \Sigma_F^G) \quad \Sigma_F^G = \{+/2, -/1, 0/0\}, \quad (\Sigma_P^G = \{\approx\})$$

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x))$

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x)) \rightsquigarrow 0, x, 0 + x, -(0 + x)$

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x)) \rightsquigarrow 0, x, 0 + x, -(0 + x)$

## Formulas (inductively)

- if  $p/k \in \Sigma_P$  and  $t_1, \dots, t_k$  are terms, then  $p(t_1, \dots, t_k)$  is an (atomic) formula

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x)) \rightsquigarrow 0, x, 0 + x, -(0 + x)$

## Formulas (inductively)

- if  $p/k \in \Sigma_P$  and  $t_1, \dots, t_k$  are terms, then  $p(t_1, \dots, t_k)$  is an (**atomic**) formula
- if  $\varphi_1, \varphi_2$  are formulas, then so are  $\neg\varphi_1, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \dots$

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x)) \rightsquigarrow 0, x, 0 + x, -(0 + x)$

## Formulas (inductively)

- if  $p/k \in \Sigma_P$  and  $t_1, \dots, t_k$  are terms, then  $p(t_1, \dots, t_k)$  is an (**atomic**) formula
- if  $\varphi_1, \varphi_2$  are formulas, then so are  $\neg\varphi_1, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \dots$
- if  $x \in \mathcal{V}$  is a variable and  $\varphi$  a formula, then so are  $\forall x.\varphi$  and  $\exists x.\varphi$

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x)) \rightsquigarrow 0, x, 0 + x, -(0 + x)$

## Formulas (inductively)

- if  $p/k \in \Sigma_P$  and  $t_1, \dots, t_k$  are terms, then  $p(t_1, \dots, t_k)$  is an (**atomic**) formula
- if  $\varphi_1, \varphi_2$  are formulas, then so are  $\neg\varphi_1, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \dots$
- if  $x \in \mathcal{V}$  is a variable and  $\varphi$  a formula, then so are  $\forall x.\varphi$  and  $\exists x.\varphi$

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x)) \rightsquigarrow 0, x, 0 + x, -(0 + x)$

## Formulas (inductively)

- if  $p/k \in \Sigma_P$  and  $t_1, \dots, t_k$  are terms, then  $p(t_1, \dots, t_k)$  is an (**atomic**) formula
- if  $\varphi_1, \varphi_2$  are formulas, then so are  $\neg\varphi_1, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \dots$
- if  $x \in \mathcal{V}$  is a variable and  $\varphi$  a formula, then so are  $\forall x.\varphi$  and  $\exists x.\varphi$

Ex:  $\forall x.(0 + x) \approx x,$

# First-Order Formula Syntax II

## Terms (inductively)

- every variable  $x \in \mathcal{V}$  is a term
- if  $f/k \in \Sigma_F$  and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term
- (there are no other terms)

Ex:  $0, x, +(0, x), -(+(0, x)) \rightsquigarrow 0, x, 0 + x, -(0 + x)$

## Formulas (inductively)

- if  $p/k \in \Sigma_P$  and  $t_1, \dots, t_k$  are terms, then  $p(t_1, \dots, t_k)$  is an (atomic) formula
- if  $\varphi_1, \varphi_2$  are formulas, then so are  $\neg\varphi_1, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \dots$
- if  $x \in \mathcal{V}$  is a variable and  $\varphi$  a formula, then so are  $\forall x.\varphi$  and  $\exists x.\varphi$

Ex:  $\forall x.(0 + x) \approx x, \quad \forall x, y, z.(x + y) + z \approx x + (y + z)$

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

Let  $x \in \mathcal{V}$  be a variable

- a (sub-)term  $x$  of a formula  $\varphi$  is called an **occurrence** of  $x$  in  $\varphi$

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

Let  $x \in \mathcal{V}$  be a variable

- a (sub-)term  $x$  of a formula  $\varphi$  is called an **occurrence** of  $x$  in  $\varphi$
- an occurrence is **bound** if it is part of a formula  $\varphi$  in  $\forall x.\varphi$  or  $\exists x.\varphi$

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

Let  $x \in \mathcal{V}$  be a variable

- a (sub-)term  $x$  of a formula  $\varphi$  is called an **occurrence** of  $x$  in  $\varphi$
- an occurrence is **bound** if it is part of a formula  $\varphi$  in  $\forall x.\varphi$  or  $\exists x.\varphi$
- otherwise, the occurrence is **free**

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

Let  $x \in \mathcal{V}$  be a variable

- a (sub-)term  $x$  of a formula  $\varphi$  is called an **occurrence** of  $x$  in  $\varphi$
- an occurrence is **bound** if it is part of a formula  $\varphi$  in  $\forall x.\varphi$  or  $\exists x.\varphi$
- otherwise, the occurrence is **free**

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

Let  $x \in \mathcal{V}$  be a variable

- a (sub-)term  $x$  of a formula  $\varphi$  is called an **occurrence** of  $x$  in  $\varphi$
- an occurrence is **bound** if it is part of a formula  $\varphi$  in  $\forall x.\varphi$  or  $\exists x.\varphi$
- otherwise, the occurrence is **free**
  
- formula is called a **sentence** if no variable occurs free in it

# First-Order Formula Syntax III

## Commonly Used Notions

- formula is **quantifier-free** if it does not contain any quantifiers
- formula is **ground** if it does not contain any variables

Let  $x \in \mathcal{V}$  be a variable

- a (sub-)term  $x$  of a formula  $\varphi$  is called an **occurrence** of  $x$  in  $\varphi$
- an occurrence is **bound** if it is part of a formula  $\varphi$  in  $\forall x.\varphi$  or  $\exists x.\varphi$
- otherwise, the occurrence is **free**
  
- formula is called a **sentence** if no variable occurs free in it

Ex:  $\forall y.(x \neq 0 \rightarrow (\forall x.x + 0 \approx y))$

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .
  - specifically, for a constant  $c$ , we have " $c^{\mathcal{I}} \in \mathcal{D}$ "

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .
  - specifically, for a constant  $c$ , we have " $c^{\mathcal{I}} \in \mathcal{D}$ "
  - a  $n$ -ary function  $p^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{B}$ , for each  $p \in \Sigma_P$  or arity  $n$ .

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .
  - specifically, for a constant  $c$ , we have " $c^{\mathcal{I}} \in \mathcal{D}$ "
  - a  $n$ -ary function  $p^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{B}$ , for each  $p \in \Sigma_P$  or arity  $n$ .
  - specifically, for a predicate symbol  $v$  of arity 0, we have  $v^{\mathcal{I}} \in \mathcal{B}$

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .
  - specifically, for a constant  $c$ , we have “ $c^{\mathcal{I}} \in \mathcal{D}$ ”
  - a  $n$ -ary function  $p^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{B}$ , for each  $p \in \Sigma_P$  or arity  $n$ .
  - specifically, for a predicate symbol  $v$  of arity 0, we have  $v^{\mathcal{I}} \in \mathcal{B}$
  - equality ( $\approx$ ) is always interpreted as the identity relation!

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .
  - specifically, for a constant  $c$ , we have “ $c^{\mathcal{I}} \in \mathcal{D}$ ”
  - a  $n$ -ary function  $p^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{B}$ , for each  $p \in \Sigma_P$  or arity  $n$ .
  - specifically, for a predicate symbol  $v$  of arity 0, we have  $v^{\mathcal{I}} \in \mathcal{B}$
  - equality ( $\approx$ ) is always interpreted as the identity relation!

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .
  - specifically, for a constant  $c$ , we have “ $c^{\mathcal{I}} \in \mathcal{D}$ ”
  - a  $n$ -ary function  $p^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{B}$ , for each  $p \in \Sigma_P$  or arity  $n$ .
  - specifically, for a predicate symbol  $v$  of arity 0, we have  $v^{\mathcal{I}} \in \mathcal{B}$
  - equality ( $\approx$ ) is always interpreted as the identity relation!

## Example (Interpretation for our $\Sigma^G$ )

$$\mathcal{I} = (\mathcal{D} = \{o, \star\}, \mathcal{A} = \{+^{\mathcal{I}} : \{(o, o) \mapsto o, (o, \star) \mapsto \star, \dots\}, \\ -^{\mathcal{I}} : \{o \mapsto o, \star \mapsto \star\}, 0^{\mathcal{I}} : \{() \mapsto o\}\})$$

# First-Order Formula Semantics I

## Structures

- $\Sigma$ -interpretation (or structure):  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$
- $\mathcal{D}$  is the universe or domain, a non-empty set
- recall:  $\mathcal{B} = \{0, 1\}$
- $\mathcal{A}$  is the assignment:
  - a  $n$ -ary function  $f^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{D}$ , for each  $f \in \Sigma_F$  or arity  $n$ .
  - specifically, for a constant  $c$ , we have " $c^{\mathcal{I}} \in \mathcal{D}$ "
  - a  $n$ -ary function  $p^{\mathcal{I}} : \mathcal{D}^n \rightarrow \mathcal{B}$ , for each  $p \in \Sigma_P$  or arity  $n$ .
  - specifically, for a predicate symbol  $v$  of arity 0, we have  $v^{\mathcal{I}} \in \mathcal{B}$
  - equality ( $\approx$ ) is always interpreted as the identity relation!

## Example (Interpretation for our $\Sigma^G$ )

$$\mathcal{I} = (\mathcal{D} = \{o, \star\}, \mathcal{A} = \{+^{\mathcal{I}} : \{(o, o) \mapsto o, (o, \star) \mapsto \star, \dots\}, \\ -^{\mathcal{I}} : \{o \mapsto o, \star \mapsto \star\}, 0^{\mathcal{I}} : \{() \mapsto o\}\})$$

Q: How many assignments on  $\{o, \star\}$  for  $\Sigma^G$  are there?

## First-Order Formula Semantics II

Actually, we also need the assignment to interpret variables:

- for every  $x \in \mathcal{V}$ , we will also have  $x^{\mathcal{I}} \in \mathcal{D}$

## First-Order Formula Semantics II

Actually, we also need the assignment to interpret variables:

- for every  $x \in \mathcal{V}$ , we will also have  $x^{\mathcal{I}} \in \mathcal{D}$
- however, we only really need to know  $x^{\mathcal{I}}$  for variables freely occurring in our formulas (and there should be none such)

## First-Order Formula Semantics II

Actually, we also need the assignment to interpret variables:

- for every  $x \in \mathcal{V}$ , we will also have  $x^{\mathcal{I}} \in \mathcal{D}$
- however, we only really need to know  $x^{\mathcal{I}}$  for variables freely occurring in our formulas (and there should be none such)
- wait till next slide to understand what it all means

## First-Order Formula Semantics II

Actually, we also need the assignment to interpret variables:

- for every  $x \in \mathcal{V}$ , we will also have  $x^{\mathcal{I}} \in \mathcal{D}$
- however, we only really need to know  $x^{\mathcal{I}}$  for variables freely occurring in our formulas (and there should be none such)
- wait till next slide to understand what it all means

# First-Order Formula Semantics II

Actually, we also need the assignment to interpret variables:

- for every  $x \in \mathcal{V}$ , we will also have  $x^{\mathcal{I}} \in \mathcal{D}$
- however, we only really need to know  $x^{\mathcal{I}}$  for variables freely occurring in our formulas (and there should be none such)
- wait till next slide to understand what it all means

## Notation

Let  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$  be a  $\Sigma$ -interpretation,  $x \in \mathcal{V}$  a variable, and  $\mathbf{v} \in \mathcal{D}$ , then  $\mathcal{I}[x \mapsto \mathbf{v}]$  is the interpretation which is the same as  $\mathcal{I}$  except that it evaluates  $x$  to  $\mathbf{v}$ , i.e.  $x^{\mathcal{I}[x \mapsto \mathbf{v}]} = \mathbf{v}$  and  $y^{\mathcal{I}[x \mapsto \mathbf{v}]} = y^{\mathcal{I}}$  whenever  $x \neq y$ .

## First-Order Formula Semantics III

Interpreting terms (recursively)

$$f(t_1, \dots, t_k)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$$

## First-Order Formula Semantics III

Interpreting terms (recursively)

$$f(t_1, \dots, t_k)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$$

Interpretation  $\mathcal{I}$  evaluates a formula to true [Tarski]

- $\mathcal{I} \models p(t_1, \dots, t_k)$  iff  $p^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}}) = \mathbf{1}$

## First-Order Formula Semantics III

Interpreting terms (recursively)

$$f(t_1, \dots, t_k)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$$

Interpretation  $\mathcal{I}$  evaluates a formula to true [Tarski]

- $\mathcal{I} \models p(t_1, \dots, t_k)$  iff  $p^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}}) = \mathbf{1}$
- $\mathcal{I} \models \neg\varphi$  iff not  $\mathcal{I} \models \varphi$

## First-Order Formula Semantics III

Interpreting terms (recursively)

$$f(t_1, \dots, t_k)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$$

Interpretation  $\mathcal{I}$  evaluates a formula to true [Tarski]

- $\mathcal{I} \models p(t_1, \dots, t_k)$  iff  $p^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}}) = \mathbf{1}$
- $\mathcal{I} \models \neg\varphi$  iff not  $\mathcal{I} \models \varphi$
- $\mathcal{I} \models \varphi_1 \wedge \varphi_2$  iff  $\mathcal{I} \models \varphi_1$  and  $\mathcal{I} \models \varphi_2$

## First-Order Formula Semantics III

Interpreting terms (recursively)

$$f(t_1, \dots, t_k)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$$

Interpretation  $\mathcal{I}$  evaluates a formula to true [Tarski]

- $\mathcal{I} \models p(t_1, \dots, t_k)$  iff  $p^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}}) = \mathbf{1}$
- $\mathcal{I} \models \neg\varphi$  iff not  $\mathcal{I} \models \varphi$
- $\mathcal{I} \models \varphi_1 \wedge \varphi_2$  iff  $\mathcal{I} \models \varphi_1$  and  $\mathcal{I} \models \varphi_2$
- $\mathcal{I} \models \varphi_1 \vee \varphi_2$  iff  $\mathcal{I} \models \varphi_1$  or  $\mathcal{I} \models \varphi_2$

# First-Order Formula Semantics III

Interpreting terms (recursively)

$$f(t_1, \dots, t_k)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$$

Interpretation  $\mathcal{I}$  evaluates a formula to true [Tarski]

- $\mathcal{I} \models p(t_1, \dots, t_k)$  iff  $p^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}}) = \mathbf{1}$
- $\mathcal{I} \models \neg\varphi$  iff not  $\mathcal{I} \models \varphi$
- $\mathcal{I} \models \varphi_1 \wedge \varphi_2$  iff  $\mathcal{I} \models \varphi_1$  and  $\mathcal{I} \models \varphi_2$
- $\mathcal{I} \models \varphi_1 \vee \varphi_2$  iff  $\mathcal{I} \models \varphi_1$  or  $\mathcal{I} \models \varphi_2$
- $\mathcal{I} \models \forall x.\varphi$  iff  $\mathcal{I}[x \mapsto \mathbf{v}] \models \varphi$  for every  $\mathbf{v} \in \mathcal{D}$

# First-Order Formula Semantics III

Interpreting terms (recursively)

$$f(t_1, \dots, t_k)^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$$

Interpretation  $\mathcal{I}$  evaluates a formula to true [Tarski]

- $\mathcal{I} \models p(t_1, \dots, t_k)$  iff  $p^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}}) = \mathbf{1}$
- $\mathcal{I} \models \neg\varphi$  iff not  $\mathcal{I} \models \varphi$
- $\mathcal{I} \models \varphi_1 \wedge \varphi_2$  iff  $\mathcal{I} \models \varphi_1$  and  $\mathcal{I} \models \varphi_2$
- $\mathcal{I} \models \varphi_1 \vee \varphi_2$  iff  $\mathcal{I} \models \varphi_1$  or  $\mathcal{I} \models \varphi_2$
- $\mathcal{I} \models \forall x.\varphi$  iff  $\mathcal{I}[x \mapsto \mathbf{v}] \models \varphi$  for every  $\mathbf{v} \in \mathcal{D}$
- $\mathcal{I} \models \exists x.\varphi$  iff  $\mathcal{I}[x \mapsto \mathbf{v}] \models \varphi$  for some  $\mathbf{v} \in \mathcal{D}$

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .  
Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

## Tautologyhood

A formula  $\varphi$  is **logically valid** iff for every interpretation  $\mathcal{I}$ ,  $\mathcal{I} \models \varphi$ .

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

## Tautologyhood

A formula  $\varphi$  is **logically valid** iff for every interpretation  $\mathcal{I}$ ,  $\mathcal{I} \models \varphi$ .

**Note:**  $\varphi$  is logically valid iff  $\neg\varphi$  does not have a model, i.e. is **unsat**.

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

## Tautologyhood

A formula  $\varphi$  is **logically valid** iff for every interpretation  $\mathcal{I}$ ,  $\mathcal{I} \models \varphi$ .

**Note:**  $\varphi$  is logically valid iff  $\neg\varphi$  does not have a model, i.e. is **unsat**.

## Entailment/Theoremhood

A set of formulas  $\Psi$  **logically entails** a formula  $\varphi$ , written  $\Psi \models \varphi$ ,  
iff for every interpretation  $\mathcal{I}$ : if  $\mathcal{I} \models \psi$  for every  $\psi \in \Psi$  then  $\mathcal{I} \models \varphi$ .

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

## Tautologyhood

A formula  $\varphi$  is **logically valid** iff for every interpretation  $\mathcal{I}$ ,  $\mathcal{I} \models \varphi$ .

**Note:**  $\varphi$  is logically valid iff  $\neg\varphi$  does not have a model, i.e. is **unsat**.

## Entailment/Theoremhood

A set of formulas  $\Psi$  **logically entails** a formula  $\varphi$ , written  $\Psi \models \varphi$ , iff for every interpretation  $\mathcal{I}$ : if  $\mathcal{I} \models \psi$  for every  $\psi \in \Psi$  then  $\mathcal{I} \models \varphi$ .

We then also say that  $\varphi$  **logically follows** from  $\Psi$ .

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

## Tautologyhood

A formula  $\varphi$  is **logically valid** iff for every interpretation  $\mathcal{I}$ ,  $\mathcal{I} \models \varphi$ .

**Note:**  $\varphi$  is logically valid iff  $\neg\varphi$  does not have a model, i.e. is **unsat**.

## Entailment/Theoremhood

A set of formulas  $\Psi$  **logically entails** a formula  $\varphi$ , written  $\Psi \models \varphi$ , iff for every interpretation  $\mathcal{I}$ : if  $\mathcal{I} \models \psi$  for every  $\psi \in \Psi$  then  $\mathcal{I} \models \varphi$ .

We then also say that  $\varphi$  **logically follows** from  $\Psi$ .

- We often call the set of formulas  $\Psi$  the **axioms** (forming a theory),

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

## Tautologyhood

A formula  $\varphi$  is **logically valid** iff for every interpretation  $\mathcal{I}$ ,  $\mathcal{I} \models \varphi$ .

**Note:**  $\varphi$  is logically valid iff  $\neg\varphi$  does not have a model, i.e. is **unsat**.

## Entailment/Theoremhood

A set of formulas  $\Psi$  **logically entails** a formula  $\varphi$ , written  $\Psi \models \varphi$ , iff for every interpretation  $\mathcal{I}$ : if  $\mathcal{I} \models \psi$  for every  $\psi \in \Psi$  then  $\mathcal{I} \models \varphi$ .

We then also say that  $\varphi$  **logically follows** from  $\Psi$ .

- We often call the set of formulas  $\Psi$  the **axioms** (forming a theory),
- then call the formula  $\varphi$  a **conjecture**,

# First-order Theorem Proving

## Satisfiability

A formula  $\varphi$  is **satisfiable** iff there is an interpretation  $\mathcal{I}$  s.t.  $\mathcal{I} \models \varphi$ .

Such an  $\mathcal{I}$  is often called a **model** of  $\varphi$ .

If a formula does not have a model, it's called **unsatisfiable**.

## Tautologyhood

A formula  $\varphi$  is **logically valid** iff for every interpretation  $\mathcal{I}$ ,  $\mathcal{I} \models \varphi$ .

**Note:**  $\varphi$  is logically valid iff  $\neg\varphi$  does not have a model, i.e. is **unsat**.

## Entailment/Theoremhood

A set of formulas  $\Psi$  **logically entails** a formula  $\varphi$ , written  $\Psi \models \varphi$ , iff for every interpretation  $\mathcal{I}$ : if  $\mathcal{I} \models \psi$  for every  $\psi \in \Psi$  then  $\mathcal{I} \models \varphi$ .

We then also say that  $\varphi$  **logically follows** from  $\Psi$ .

- We often call the set of formulas  $\Psi$  the **axioms** (forming a theory),
- then call the formula  $\varphi$  a **conjecture**,
- and say that the conjecture is a **theorem** of the theory, if  $\Psi \models \varphi$ .

# Outline

First-Order Logic as a Formal Language

Clause Normal Form in First-Order Logic

A Static View: Inferences, Soundness, and Completeness

A Dynamic View: Saturation

## First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal:** either an atomic formula  $A$  or its negation  $\neg A$ .

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal:** either an atomic formula  $A$  or its negation  $\neg A$ .
- **Clause:** a disjunction  $L_1 \vee \dots \vee L_n$  of literals,  $n \geq 0$

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal:** either an atomic formula  $A$  or its negation  $\neg A$ .
- **Clause:** a disjunction  $L_1 \vee \dots \vee L_n$  of literals,  $n \geq 0$   
formally, this is going to be **multi-set** (not a set)

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal:** either an atomic formula  $A$  or its negation  $\neg A$ .
- **Clause:** a disjunction  $L_1 \vee \dots \vee L_n$  of literals,  $n \geq 0$   
formally, this is going to be **multi-set** (not a set)
- **Empty clause**, denoted by  $\square$ : the clause with 0 literals

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal:** either an atomic formula  $A$  or its negation  $\neg A$ .
- **Clause:** a disjunction  $L_1 \vee \dots \vee L_n$  of literals,  $n \geq 0$   
formally, this is going to be **multi-set** (not a set)
- **Empty clause**, denoted by  $\square$ : the clause with 0 literals
- A formula in **Clausal Normal Form (CNF)**: a conjunction of clauses.

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal:** either an atomic formula  $A$  or its negation  $\neg A$ .
- **Clause:** a disjunction  $L_1 \vee \dots \vee L_n$  of literals,  $n \geq 0$   
formally, this is going to be **multi-set** (not a set)
- **Empty clause**, denoted by  $\square$ : the clause with 0 literals
- A formula in **Clausal Normal Form (CNF)**: a conjunction of clauses.
- A clause is **ground** if it contains no variables.

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal**: either an atomic formula  $A$  or its negation  $\neg A$ .
- **Clause**: a disjunction  $L_1 \vee \dots \vee L_n$  of literals,  $n \geq 0$   
formally, this is going to be **multi-set** (not a set)
- **Empty clause**, denoted by  $\square$ : the clause with 0 literals
- A formula in **Clausal Normal Form (CNF)**: a conjunction of clauses.
- A clause is **ground** if it contains no variables.
- All variables are **(implicitly) universally quantified** on the “outside”:

# First-Order Clauses

Clauses are the basic “units of information” that the prover deals with.

- **Literal**: either an atomic formula  $A$  or its negation  $\neg A$ .
- **Clause**: a disjunction  $L_1 \vee \dots \vee L_n$  of literals,  $n \geq 0$   
formally, this is going to be **multi-set** (not a set)
- **Empty clause**, denoted by  $\square$ : the clause with 0 literals
- A formula in **Clausal Normal Form (CNF)**: a conjunction of clauses.
- A clause is **ground** if it contains no variables.
- All variables are **(implicitly) universally quantified** on the “outside”:

**Ex:** That is, we understand  $p(x) \vee q(x)$  as  $\forall x.(p(x) \vee q(x))$ .

# A Clausification Pipeline I

## Implication is a shorthand

$$(\varphi \rightarrow \psi) \implies \neg\varphi \vee \psi$$

# A Classification Pipeline I

## Implication is a shorthand

$$(\varphi \rightarrow \psi) \implies \neg\varphi \vee \psi$$

## Negation “pushed” inside

$$\neg\neg\varphi \implies \varphi$$

# A Clausification Pipeline I

## Implication is a shorthand

$$(\varphi \rightarrow \psi) \implies \neg\varphi \vee \psi$$

## Negation “pushed” inside

$$\neg\neg\varphi \implies \varphi$$

$$\neg(\varphi \vee \psi) \implies \neg\varphi \wedge \neg\psi$$

$$\neg(\varphi \wedge \psi) \implies \neg\varphi \vee \neg\psi$$

# A Classification Pipeline I

## Implication is a shorthand

$$(\varphi \rightarrow \psi) \implies \neg\varphi \vee \psi$$

## Negation “pushed” inside

$$\neg\neg\varphi \implies \varphi$$

$$\neg(\varphi \vee \psi) \implies \neg\varphi \wedge \neg\psi$$

$$\neg(\varphi \wedge \psi) \implies \neg\varphi \vee \neg\psi$$

$$\neg(\forall x.\varphi) \implies (\exists x.\neg\varphi)$$

$$\neg(\exists x.\varphi) \implies (\forall x.\neg\varphi)$$

# A Classification Pipeline I

## Implication is a shorthand

$$(\varphi \rightarrow \psi) \implies \neg\varphi \vee \psi$$

## Negation “pushed” inside

$$\neg\neg\varphi \implies \varphi$$

$$\neg(\varphi \vee \psi) \implies \neg\varphi \wedge \neg\psi$$

$$\neg(\varphi \wedge \psi) \implies \neg\varphi \vee \neg\psi$$

$$\neg(\forall x.\varphi) \implies (\exists x.\neg\varphi)$$

$$\neg(\exists x.\varphi) \implies (\forall x.\neg\varphi)$$

$$\neg(\varphi \leftrightarrow \psi) \implies (\varphi \oplus \psi)$$

$$\neg(\varphi \oplus \psi) \implies (\varphi \leftrightarrow \psi)$$

## Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

**Example (names may need arguments)**

$$\forall x \exists y \forall z [(\forall q)(q \in z \rightarrow q \in x) \leftrightarrow z \in y]$$

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

**Example (names may need arguments)**

$$\forall x \exists y \forall z [( \forall q )( q \in z \rightarrow q \in x ) \leftrightarrow z \in y]$$

Looking for the free variables of the named sub-formula:

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

**Example (names may need arguments)**

$$\forall x \exists y \forall z [( \forall q )( q \in z \rightarrow q \in x ) \leftrightarrow z \in y]$$

Looking for the free variables of the named sub-formula: **z,x**

# Formula Naming

**The distributive law may cause a blow-up**

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

**Formula naming (Tseitin trick):**

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

**Example (names may need arguments)**

$$\forall x \exists y \forall z [(\forall q)(q \in z \rightarrow q \in x) \leftrightarrow z \in y]$$

Looking for the free variables of the named sub-formula: **z,x**

$$\forall x \exists y \forall z [subs(z, x) \leftrightarrow z \in y] \wedge \forall z \forall x (subs(z, x) \leftrightarrow (\forall q)(q \in z \rightarrow q \in x))$$

# Formula Naming

The distributive law may cause a blow-up

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

Formula naming (Tseitin trick):

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

Example (names may need arguments)

$$\forall x \exists y \forall z [(\forall q)(q \in z \rightarrow q \in x) \leftrightarrow z \in y]$$

Looking for the free variables of the named sub-formula: **z,x**

$$\forall x \exists y \forall z [subs(z, x) \leftrightarrow z \in y] \wedge \forall z \forall x (subs(z, x) \leftrightarrow (\forall q)(q \in z \rightarrow q \in x))$$

NB: The new symbol definition is **always universally quantified!**

# Formula Naming

The distributive law may cause a blow-up

$$(A \wedge B) \vee (C \wedge D) \implies (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$$

Formula naming (Tseitin trick):

$$\dots (A \wedge B) \vee (C \wedge D) \dots \implies \dots (F_{AB} \vee (C \wedge D)) \dots \wedge (F_{AB} \leftrightarrow A \wedge B)$$

- a new **predicate symbol** to stand for the named subformula
- definition added at the “top level” (holds globally)

Example (names may need arguments)

$$\forall x \exists y \forall z [(\forall q)(q \in z \rightarrow q \in x) \leftrightarrow z \in y]$$

Looking for the free variables of the named sub-formula:  $z, x$

$$\forall x \exists y \forall z [subs(z, x) \leftrightarrow z \in y] \wedge \forall z \forall x (subs(z, x) \leftrightarrow (\forall q)(q \in z \rightarrow q \in x))$$

NB: The new symbol definition is **always universally quantified!**

Ex: One more example in Vampire

## A Clausification Pipeline cont.

### Finish Negation Normal Form

$$(\varphi \leftrightarrow \psi) \implies (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

$$(\varphi \oplus \psi) \implies (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$$

## A Clausification Pipeline cont.

### Finish Negation Normal Form

$$(\varphi \leftrightarrow \psi) \implies (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

$$(\varphi \oplus \psi) \implies (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$$

Note:  $\leftrightarrow/\oplus$  expansion is another potential source of a blowup!

## A Clausification Pipeline cont.

### Finish Negation Normal Form

$$(\varphi \leftrightarrow \psi) \implies (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

$$(\varphi \oplus \psi) \implies (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$$

Note:  $\leftrightarrow/\oplus$  expansion is another potential source of a blowup!

Note: new negations appearing  $\implies$  need to do more “pushing”.

## A Clausification Pipeline cont.

### Finish Negation Normal Form

$$(\varphi \leftrightarrow \psi) \implies (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

$$(\varphi \oplus \psi) \implies (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$$

Note:  $\leftrightarrow/\oplus$  expansion is another potential source of a blowup!

Note: new negations appearing  $\implies$  need to do more “pushing”.

### Get rid of the existential quantifiers:

- **Skolemization**: new function symbol to stand for the  $\exists$ -variable
- as with naming, arguments come from free variables

# A Clausification Pipeline cont.

## Finish Negation Normal Form

$$(\varphi \leftrightarrow \psi) \implies (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

$$(\varphi \oplus \psi) \implies (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$$

Note:  $\leftrightarrow/\oplus$  expansion is another potential source of a blowup!

Note: new negations appearing  $\implies$  need to do more “pushing”.

## Get rid of the existential quantifiers:

- **Skolemization**: new function symbol to stand for the  $\exists$ -variable
- as with naming, arguments come from free variables

## Example

$$\forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)]$$

# A Clausification Pipeline cont.

## Finish Negation Normal Form

$$(\varphi \leftrightarrow \psi) \implies (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

$$(\varphi \oplus \psi) \implies (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$$

Note:  $\leftrightarrow/\oplus$  expansion is another potential source of a blowup!

Note: new negations appearing  $\implies$  need to do more “pushing”.

## Get rid of the existential quantifiers:

- **Skolemization**: new function symbol to stand for the  $\exists$ -variable
- as with naming, arguments come from free variables

## Example

$$\forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)] \implies \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$$

# A Clausification Pipeline cont.

## Finish Negation Normal Form

$$(\varphi \leftrightarrow \psi) \implies (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$$

$$(\varphi \oplus \psi) \implies (\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi)$$

Note:  $\leftrightarrow/\oplus$  expansion is another potential source of a blowup!

Note: new negations appearing  $\implies$  need to do more “pushing”.

## Get rid of the existential quantifiers:

- **Skolemization**: new function symbol to stand for the  $\exists$ -variable
- as with naming, arguments come from free variables

## Example

$$\forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)] \implies \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$$

Free variables of  $\exists y(x \cdot y \approx 1)$  are  $x$ , therefore we created  $sk_y(x)$

## Correctness Considerations

**What does it mean that our CNF transformation is correct?**

## Correctness Considerations

**What does it mean that our CNF transformation is correct?**

- Idea: In each step, it replaces a formula by an equivalent one!

# Correctness Considerations

**What does it mean that our CNF transformation is correct?**

- Idea: In each step, it replaces a formula by an equivalent one!

## Equivalence

First-order formulas  $\varphi$  and  $\psi$  in language  $\Sigma$  are **equivalent** if

$$\mathcal{I} \models \varphi \quad \text{if and only if} \quad \mathcal{I} \models \psi$$

for every  $\Sigma$ -interpretation  $\mathcal{I}$ .

## Correctness Considerations

**What does it mean that our CNF transformation is correct?**

- Idea: In each step, it replaces a formula by an equivalent one!

### Equivalence

First-order formulas  $\varphi$  and  $\psi$  in language  $\Sigma$  are **equivalent** if

$$\mathcal{I} \models \varphi \quad \text{if and only if} \quad \mathcal{I} \models \psi$$

for every  $\Sigma$ -interpretation  $\mathcal{I}$ .

**But do naming and skolemization really preserve equivalence?**

## Correctness Considerations

### What does it mean that our CNF transformation is correct?

- Idea: In each step, it replaces a formula by an equivalent one!

### Equivalence

First-order formulas  $\varphi$  and  $\psi$  in language  $\Sigma$  are **equivalent** if

$$\mathcal{I} \models \varphi \quad \text{if and only if} \quad \mathcal{I} \models \psi$$

for every  $\Sigma$ -interpretation  $\mathcal{I}$ .

### But do naming and skolemization really preserve equivalence?

- Strictly speaking, this does not make sense (the  $\Sigma$  is different).

## Correctness Considerations

### What does it mean that our CNF transformation is correct?

- Idea: In each step, it replaces a formula by an equivalent one!

### Equivalence

First-order formulas  $\varphi$  and  $\psi$  in language  $\Sigma$  are **equivalent** if

$$\mathcal{I} \models \varphi \quad \text{if and only if} \quad \mathcal{I} \models \psi$$

for every  $\Sigma$ -interpretation  $\mathcal{I}$ .

### But do naming and skolemization really preserve equivalence?

- Strictly speaking, this does not make sense (the  $\Sigma$  is different).
- Can we pretend the symbol was already there?

# Correctness Considerations

## What does it mean that our CNF transformation is correct?

- Idea: In each step, it replaces a formula by an equivalent one!

## Equivalence

First-order formulas  $\varphi$  and  $\psi$  in language  $\Sigma$  are **equivalent** if

$$\mathcal{I} \models \varphi \quad \text{if and only if} \quad \mathcal{I} \models \psi$$

for every  $\Sigma$ -interpretation  $\mathcal{I}$ .

## But do naming and skolemization really preserve equivalence?

- Strictly speaking, this does not make sense (the  $\Sigma$  is different).
- Can we pretend the symbol was already there?

Ex: Well, no:  $M = \{A \mapsto 0, B \mapsto 0, C \mapsto 1, D \mapsto 1, F_{AB} \mapsto 1\}$

$$M \models (A \wedge B) \vee (C \wedge D) \quad \text{but} \quad M \not\models (F_{AB} \leftrightarrow A \wedge B)$$

## Correctness Considerations II

**But do we really need equivalence?**

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.
- So its enough to preserve existence of some model!

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.
- So its enough to preserve existence of **some** model!

### Example (for Skolemization (general proof, example formula))

Let  $\Sigma = (\Sigma_P, \Sigma_F)$  and  $\Sigma' = (\Sigma_P, \Sigma'_F)$  for  $\Sigma'_F = \Sigma_F \cup \{sk_y/1\}$ . Then there exists a  $\Sigma$ -interp.  $\mathcal{I}$  such that  $\mathcal{I} \models \forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)]$  iff there exists a  $\Sigma'$ -interp.  $\mathcal{I}'$  such that  $\mathcal{I}' \models \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$ .

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.
- So its enough to preserve existence of some model!

### Example (for Skolemization (general proof, example formula))

Let  $\Sigma = (\Sigma_P, \Sigma_F)$  and  $\Sigma' = (\Sigma_P, \Sigma'_F)$  for  $\Sigma'_F = \Sigma_F \cup \{sk_y/1\}$ . Then there exists a  $\Sigma$ -interp.  $\mathcal{I}$  such that  $\mathcal{I} \models \forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)]$  iff there exists a  $\Sigma'$ -interp.  $\mathcal{I}'$  such that  $\mathcal{I}' \models \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$ .  
“ $\Rightarrow$ ” Fix a  $\Sigma$ -interp.  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$  s.t. ...

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.
- So its enough to preserve existence of some model!

### Example (for Skolemization (general proof, example formula))

Let  $\Sigma = (\Sigma_P, \Sigma_F)$  and  $\Sigma' = (\Sigma_P, \Sigma'_F)$  for  $\Sigma'_F = \Sigma_F \cup \{sk_y/1\}$ . Then there exists a  $\Sigma$ -interp.  $\mathcal{I}$  such that  $\mathcal{I} \models \forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)]$  iff there exists a  $\Sigma'$ -interp.  $\mathcal{I}'$  such that  $\mathcal{I}' \models \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$ .  
“ $\Rightarrow$ ” Fix a  $\Sigma$ -interp.  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$  s.t. ...

Define  $f : \mathcal{D} \rightarrow 2^{\mathcal{D}}$  mapping  $\mathbf{v} \mapsto \{\mathbf{w} \mid \mathcal{I}[x \mapsto \mathbf{v}, y \mapsto \mathbf{w}] \models x \cdot y \approx 1\}$ .

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.
- So its enough to preserve existence of **some** model!

### Example (for Skolemization (general proof, example formula))

Let  $\Sigma = (\Sigma_P, \Sigma_F)$  and  $\Sigma' = (\Sigma_P, \Sigma'_F)$  for  $\Sigma'_F = \Sigma_F \cup \{sk_y/1\}$ . Then there exists a  $\Sigma$ -interp.  $\mathcal{I}$  such that  $\mathcal{I} \models \forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)]$  iff there exists a  $\Sigma'$ -interp.  $\mathcal{I}'$  such that  $\mathcal{I}' \models \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$ .  
“ $\Rightarrow$ ” Fix a  $\Sigma$ -interp.  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$  s.t. ...

Define  $f : \mathcal{D} \rightarrow 2^{\mathcal{D}}$  mapping  $\mathbf{v} \mapsto \{\mathbf{w} \mid \mathcal{I}[x \mapsto \mathbf{v}, y \mapsto \mathbf{w}] \models x \cdot y \approx 1\}$ .  
Let  $Sf$  be a **selector** on  $f$ , i.e., a function  $Sf : \mathcal{D} \rightarrow \mathcal{D}$  such that for every  $\mathbf{v} \in \mathcal{D}$ ,  $Sf(\mathbf{v}) \in f(\mathbf{v})$  whenever  $f(\mathbf{v})$  is non-empty.

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.
- So its enough to preserve existence of some model!

### Example (for Skolemization (general proof, example formula))

Let  $\Sigma = (\Sigma_P, \Sigma_F)$  and  $\Sigma' = (\Sigma_P, \Sigma'_F)$  for  $\Sigma'_F = \Sigma_F \cup \{sk_y/1\}$ . Then there exists a  $\Sigma$ -interp.  $\mathcal{I}$  such that  $\mathcal{I} \models \forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)]$  iff there exists a  $\Sigma'$ -interp.  $\mathcal{I}'$  such that  $\mathcal{I}' \models \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$ .  
“ $\Rightarrow$ ” Fix a  $\Sigma$ -interp.  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$  s.t. ...

Define  $f : \mathcal{D} \rightarrow 2^{\mathcal{D}}$  mapping  $\mathbf{v} \mapsto \{\mathbf{w} \mid \mathcal{I}[x \mapsto \mathbf{v}, y \mapsto \mathbf{w}] \models x \cdot y \approx 1\}$ .  
Let  $Sf$  be a selector on  $f$ , i.e., a function  $Sf : \mathcal{D} \rightarrow \mathcal{D}$  such that for every  $\mathbf{v} \in \mathcal{D}$ ,  $Sf(\mathbf{v}) \in f(\mathbf{v})$  whenever  $f(\mathbf{v})$  is non-empty.  
Now  $\mathcal{I}'$  extends  $\mathcal{I}$ 's  $\mathcal{A}$  to newly also interpret  $sk_y$  as  $sk_y^{\mathcal{I}'} = Sf$ .

“ $\Leftarrow$ ” Fix a  $\Sigma'$ -interpretation  $\mathcal{I}' = (\mathcal{D}, \mathcal{A}')$ . s.t. ...

## Correctness Considerations II

### But do we really need equivalence?

- Recall that we seek to show (un)satisfiability of our input formula.
- So its enough to preserve existence of **some** model!

### Example (for Skolemization (general proof, example formula))

Let  $\Sigma = (\Sigma_P, \Sigma_F)$  and  $\Sigma' = (\Sigma_P, \Sigma'_F)$  for  $\Sigma'_F = \Sigma_F \cup \{sk_y/1\}$ . Then there exists a  $\Sigma$ -interp.  $\mathcal{I}$  such that  $\mathcal{I} \models \forall x.[x \neq 0 \rightarrow \exists y(x \cdot y \approx 1)]$  iff there exists a  $\Sigma'$ -interp.  $\mathcal{I}'$  such that  $\mathcal{I}' \models \forall x.[x \neq 0 \rightarrow x \cdot sk_y(x) \approx 1]$ .  
“ $\Rightarrow$ ” Fix a  $\Sigma$ -interp.  $\mathcal{I} = (\mathcal{D}, \mathcal{A})$  s.t. ...

Define  $f : \mathcal{D} \rightarrow 2^{\mathcal{D}}$  mapping  $\mathbf{v} \mapsto \{\mathbf{w} \mid \mathcal{I}[x \mapsto \mathbf{v}, y \mapsto \mathbf{w}] \models x \cdot y \approx 1\}$ .  
Let  $Sf$  be a **selector** on  $f$ , i.e., a function  $Sf : \mathcal{D} \rightarrow \mathcal{D}$  such that for every  $\mathbf{v} \in \mathcal{D}$ ,  $Sf(\mathbf{v}) \in f(\mathbf{v})$  whenever  $f(\mathbf{v})$  is non-empty.  
Now  $\mathcal{I}'$  extends  $\mathcal{I}$ 's  $\mathcal{A}$  to newly also interpret  $sk_y$  as  $sk_y^{\mathcal{I}'} = Sf$ .

“ $\Leftarrow$ ” Fix a  $\Sigma'$ -interpretation  $\mathcal{I}' = (\mathcal{D}, \mathcal{A}')$ . s.t. ...

Return  $\mathcal{I}$  which simply “forgets” what  $\mathcal{I}'$  assigned to  $sk_y$ .

## Let's quickly have a look

```
./vampire --mode clausify Problems/PUZ/PUZ031+1.p
```

## Let's quickly have a look

```
./vampire --mode clausify Problems/PUZ/PUZ031+1.p

./vampire --mode clausify
fof(pel47_14,axiom,
  ( ! [X] :
    ( ( caterpillar(X)
      | snail(X) )
    => ? [Y] :
      ( plant(Y)
        & eats(X,Y) ) ) ) ).
```

# Outline

First-Order Logic as a Formal Language

Clause Normal Form in First-Order Logic

A Static View: Inferences, Soundness, and Completeness

A Dynamic View: Saturation

# Inference System

- An **inference** has the form

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

where  $n \geq 0$  and  $F_1, \dots, F_n, G$  are formulas.

# Inference System

- An **inference** has the form

$$\frac{F_1 \quad \dots \quad F_n}{G} ,$$

where  $n \geq 0$  and  $F_1, \dots, F_n, G$  are formulas.

- The formula  $G$  is called the **conclusion** of the inference;

# Inference System

- An **inference** has the form

$$\frac{F_1 \quad \dots \quad F_n}{G} ,$$

where  $n \geq 0$  and  $F_1, \dots, F_n, G$  are formulas.

- The formula  $G$  is called the **conclusion** of the inference;
- The formulas  $F_1, \dots, F_n$  are called its **premises**.

# Inference System

- An **inference** has the form

$$\frac{F_1 \quad \dots \quad F_n}{G} ,$$

where  $n \geq 0$  and  $F_1, \dots, F_n, G$  are formulas.

- The formula  $G$  is called the **conclusion** of the inference;
- The formulas  $F_1, \dots, F_n$  are called its **premises**.

# Inference System

- An **inference** has the form

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

where  $n \geq 0$  and  $F_1, \dots, F_n, G$  are formulas.

- The formula  $G$  is called the **conclusion** of the inference;
- The formulas  $F_1, \dots, F_n$  are called its **premises**.
- An **inference rule**  $R$  is a set of inferences.

# Inference System

- An **inference** has the form

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

where  $n \geq 0$  and  $F_1, \dots, F_n, G$  are formulas.

- The formula  $G$  is called the **conclusion** of the inference;
- The formulas  $F_1, \dots, F_n$  are called its **premises**.
  
- An **inference rule**  $R$  is a set of inferences.
- Every inference  $I \in R$  is called an **instance of**  $R$ .

# Inference System

- An **inference** has the form

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

where  $n \geq 0$  and  $F_1, \dots, F_n, G$  are formulas.

- The formula  $G$  is called the **conclusion** of the inference;
- The formulas  $F_1, \dots, F_n$  are called its **premises**.
  
- An **inference rule**  $R$  is a set of inferences.
- Every inference  $I \in R$  is called an **instance of**  $R$ .
- An **inference system**  $\mathbb{I}$  is a set of inference rules.

- **Derivation** in an inference system  $\mathbb{I}$ :  
a finite DAG “built from inferences” in  $\mathbb{I}$ .

- **Derivation** in an inference system  $\mathbb{I}$ :  
a finite DAG “built from inferences” in  $\mathbb{I}$ .
- **Derivation of  $E$  from  $E_1, \dots, E_m$** : a derivation whose every leaf is one of the expressions  $E_1, \dots, E_m$  and the root of which is  $E$ .

# Derivation, Proof

- **Derivation** in an inference system  $\mathbb{I}$ :  
a finite DAG “built from inferences” in  $\mathbb{I}$ .
- **Derivation of  $E$  from  $E_1, \dots, E_m$** : a derivation whose every leaf is one of the expressions  $E_1, \dots, E_m$  and the root of which is  $E$ .
- A **refutation** is a derivation of the empty clause  $\square$ .

# Binary Resolution Inference System

The **binary resolution inference system**, denoted by  $\mathbb{BR}$  is an inference system on **propositional** clauses (or **ground** clauses).

It consists of two inference rules:

# Binary Resolution Inference System

The **binary resolution inference system**, denoted by **BR** is an inference system on **propositional** clauses (or **ground** clauses).

It consists of two inference rules:

- **Binary resolution**, denoted by **BR**:

$$\frac{p \vee C_1 \quad \neg p \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

# Binary Resolution Inference System

The **binary resolution inference system**, denoted by **BR** is an inference system on **propositional** clauses (or **ground** clauses).

It consists of two inference rules:

- **Binary resolution**, denoted by **BR**:

$$\frac{p \vee C_1 \quad \neg p \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- **Factoring**, denoted by **Fact**:

$$\frac{L \vee L \vee C}{L \vee C} \text{ (Fact).}$$

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G} ,$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G} ,$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .  
i.e., if  $F_1, \dots, F_n \models G$ .

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G} ,$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .  
i.e., if  $F_1, \dots, F_n \models G$ .

- (An inference system is sound if every inference in every inference rule in this system is sound.)

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .  
i.e., if  $F_1, \dots, F_n \models G$ .

- (An inference system is sound if every inference in every inference rule in this system is sound.)

Ex:  $\mathbb{B}\mathbb{R}$  is sound.

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .  
i.e., if  $F_1, \dots, F_n \models G$ .

- (An inference system is sound if every inference in every inference rule in this system is sound.)

Ex:  $\mathbb{BR}$  is sound.

## Consequence of soundness:

Let  $N$  be a set of clauses. If  $\square$  can be derived from  $N$  by a sound inference system (e.g.  $\mathbb{BR}$ ), then  $N$  is **unsatisfiable**.

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .  
i.e., if  $F_1, \dots, F_n \models G$ .

- (An inference system is sound if every inference in every inference rule in this system is sound.)

Ex:  $\mathbb{BR}$  is sound.

## Consequence of soundness:

Let  $N$  be a set of clauses. If  $\square$  can be derived from  $N$  by a sound inference system (e.g.  $\mathbb{BR}$ ), then  $N$  is **unsatisfiable**.

Proof:

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .  
i.e., if  $F_1, \dots, F_n \models G$ .

- (An inference system is sound if every inference in every inference rule in this system is sound.)

Ex:  $\mathbb{BR}$  is sound.

## Consequence of soundness:

Let  $N$  be a set of clauses. If  $\square$  can be derived from  $N$  by a sound inference system (e.g.  $\mathbb{BR}$ ), then  $N$  is **unsatisfiable**.

Proof: 1) induction over the derivation,

# Soundness

- An inference is **sound**

$$\frac{F_1 \quad \dots \quad F_n}{G},$$

if its conclusion  $G$  logically follows from its premises  $F_1, \dots, F_n$ .  
i.e., if  $F_1, \dots, F_n \models G$ .

- (An inference system is sound if every inference in every inference rule in this system is sound.)

Ex:  $\mathbb{BR}$  is sound.

## Consequence of soundness:

Let  $N$  be a set of clauses. If  $\square$  can be derived from  $N$  by a sound inference system (e.g.  $\mathbb{BR}$ ), then  $N$  is **unsatisfiable**.

Proof: 1) induction over the derivation, 2)  $\mathcal{I} \not\models \square$  for any  $\mathcal{I}$

## Example

Consider the following set of clauses

$$N = \{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q\}.$$

The following derivation derives the empty clause from  $N$ :

$$\frac{\frac{\frac{p \vee q \quad p \vee \neg q}{p \vee p} \text{ (BR)}}{p} \text{ (Fact)}}{\quad} \quad \frac{\frac{\frac{\neg p \vee q \quad \neg p \vee \neg q}{\neg p \vee \neg p} \text{ (BR)}}{\neg p} \text{ (Fact)}}{\quad} \text{ (BR)}$$

□

Hence,  $N$  is **unsatisfiable**.

## Can this be used for checking (un)satisfiability?

1. What if the empty clause **cannot be derived** from  $N$ ?

## Can this be used for checking (un)satisfiability?

1. What if the empty clause **cannot be derived** from  $N$ ?
2. Even if we **systematically** search for possible derivations of the empty clause?

## Can this be used for checking (un)satisfiability?

1. What if the empty clause **cannot be derived** from  $N$ ?
2. Even if we **systematically** search for possible derivations of the empty clause?

### **Completeness:**

An inference system  $\mathbb{I}$  is **complete**, if for every set of clauses  $N$ , if  $N$  is unsatisfiable then there exists a refutation in  $\mathbb{I}$  from  $N$ .

## Can this be used for checking (un)satisfiability?

1. What if the empty clause **cannot be derived** from  $N$ ?
2. Even if we **systematically** search for possible derivations of the empty clause?

### **Completeness:**

An inference system  $\mathbb{I}$  is **complete**, if for every set of clauses  $N$ , if  $N$  is unsatisfiable then there exists a refutation in  $\mathbb{I}$  from  $N$ .

Ex:  $\mathbb{BR}$  is complete.

## Can this be used for checking (un)satisfiability?

1. What if the empty clause **cannot be derived** from  $N$ ?
2. Even if we **systematically** search for possible derivations of the empty clause?

### Completeness:

An inference system  $\mathbb{I}$  is **complete**, if for every set of clauses  $N$ , if  $N$  is unsatisfiable then there exists a refutation in  $\mathbb{I}$  from  $N$ .

Ex:  $\mathbb{BR}$  is complete.

### Note:

- It's not at all obvious that a sound and complete calculus for a given logic  $X$  even exists!

## Can this be used for checking (un)satisfiability?

1. What if the empty clause **cannot be derived** from  $N$ ?
2. Even if we **systematically** search for possible derivations of the empty clause?

### Completeness:

An inference system  $\mathbb{I}$  is **complete**, if for every set of clauses  $N$ , if  $N$  is unsatisfiable then there exists a refutation in  $\mathbb{I}$  from  $N$ .

Ex:  $\mathbb{BR}$  is complete.

### Note:

- It's not at all obvious that a sound and complete calculus for a given logic  $X$  even exists!
- $\vdash$  means witnessing finitely;  $\models$  goes over infinitely many structures

# Outline

First-Order Logic as a Formal Language

Clause Normal Form in First-Order Logic

A Static View: Inferences, Soundness, and Completeness

A Dynamic View: Saturation

## Idea of Saturation

Completeness is formulated in terms of **derivability** of the empty clause from a set  $S_0$  of clauses in an inference system  $\mathbb{I}$ . However, this formulations gives **no hint on how to search** for such a derivation.

## Idea of Saturation

Completeness is formulated in terms of **derivability** of the empty clause from a set  $S_0$  of clauses in an inference system  $\mathbb{I}$ . However, this formulations gives **no hint on how to search** for such a derivation.

Idea:

# Idea of Saturation

Completeness is formulated in terms of **derivability** of the empty clause from a set  $S_0$  of clauses in an inference system  $\mathbb{I}$ . However, this formulations gives **no hint on how to search** for such a derivation.

Idea:

- Take a set of clauses  $S$ , initially  $S := S_0$ .

# Idea of Saturation

Completeness is formulated in terms of **derivability** of the empty clause from a set  $S_0$  of clauses in an inference system  $\mathbb{I}$ . However, this formulations gives **no hint on how to search** for such a derivation.

Idea:

- Take a set of clauses  $S$ , initially  $S := S_0$ .
- **Repeatedly apply inferences** of  $\mathbb{I}$  to clauses in  $S$  and add their conclusions to  $S$ , unless these conclusions are already in  $S$ .

# Idea of Saturation

Completeness is formulated in terms of **derivability** of the empty clause from a set  $S_0$  of clauses in an inference system  $\mathbb{I}$ . However, this formulations gives **no hint on how to search** for such a derivation.

Idea:

- Take a set of clauses  $S$ , initially  $S := S_0$ .
- **Repeatedly apply inferences** of  $\mathbb{I}$  to clauses in  $S$  and add their conclusions to  $S$ , unless these conclusions are already in  $S$ .
- If, at any stage, we obtain  $\square$ , we terminate and **report unsatisfiability** of  $S_0$ .

# Saturation Algorithm

A **saturation algorithm** tries to **saturate** a set of clauses with respect to a given inference system.

# Saturation Algorithm

A **saturation algorithm** tries to **saturate** a set of clauses with respect to a given inference system.

**In theory** there are three possible scenarios:

1. At some moment the empty clause  $\square$  is generated, in this case the input set of clauses is **unsatisfiable**.

# Saturation Algorithm

A **saturation algorithm** tries to **saturate** a set of clauses with respect to a given inference system.

**In theory** there are three possible scenarios:

1. At some moment the empty clause  $\square$  is generated, in this case the input set of clauses is **unsatisfiable**.
2. Saturation will terminate without ever generating  $\square$ , in this case the input set of clauses is **satisfiable**.

# Saturation Algorithm

A **saturation algorithm** tries to **saturate** a set of clauses with respect to a given inference system.

**In theory** there are three possible scenarios:

1. At some moment the empty clause  $\square$  is generated, in this case the input set of clauses is **unsatisfiable**.
2. Saturation will terminate without ever generating  $\square$ , in this case the input set of clauses is **satisfiable**.
3. Saturation will run **forever**, but without generating  $\square$ . In this case the input set of clauses is **satisfiable**.

# Saturation Algorithm in Practice

In practice there are three possible scenarios:

1. At some moment the empty clause  $\square$  is generated, in this case the input set of clauses is **unsatisfiable**.

# Saturation Algorithm in Practice

In practice there are three possible scenarios:

1. At some moment the empty clause  $\square$  is generated, in this case the input set of clauses is **unsatisfiable**.
2. Saturation will terminate without ever generating  $\square$ , in this case the input set of clauses is **satisfiable**.

# Saturation Algorithm in Practice

In practice there are three possible scenarios:

1. At some moment the empty clause  $\square$  is generated, in this case the input set of clauses is **unsatisfiable**.
2. Saturation will terminate without ever generating  $\square$ , in this case the input set of clauses is **satisfiable**.
3. Saturation will run **until we run out of resources**, but without generating  $\square$ . In this case it is **unknown** whether the input set is satisfiable or not.

# Saturation with a Given-Clause Algorithm

## How to decide which inference to consider next?

- a **given-clause** saturation algorithm

# Saturation with a Given-Clause Algorithm

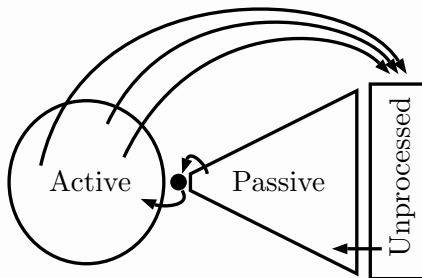
## How to decide which inference to consider next?

- a **given-clause** saturation algorithm
- inference selection via clause selection

# Saturation with a Given-Clause Algorithm

## How to decide which inference to consider next?

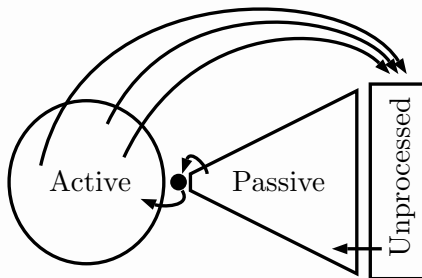
- a **given-clause** saturation algorithm
- inference selection via clause selection



# Saturation with a Given-Clause Algorithm

## How to decide which inference to consider next?

- a **given-clause** saturation algorithm
- inference selection via clause selection



Only apply inferences to

- the *selected clause* and the *previously selected clauses*.

# Anatomy of Saturation

## Active

- initially empty
- backed up by sophisticated data structures (indexes) to allow efficient processing of inferences

# Anatomy of Saturation

## Active

- initially empty
- backed up by sophisticated data structures (indexes) to allow efficient processing of inferences

## Passive

- initially contains the classified input
- typically consists of several queues ordering clauses to process by various (heuristic) criteria
- fairness!

# Anatomy of Saturation

## Active

- **initially** empty
- backed up by **sophisticated data structures** (indexes) to allow efficient processing of inferences

## Passive

- **initially** contains the classified input
- typically consists of **several queues** ordering clauses to process by various (heuristic) criteria
- fairness!

## Unprocessed:

- a temporary container
- just after generation, simplify before put into passive

# The Clause Selection Task

Selecting the given clause is arguably the most important choice point in the implementation of a saturation algorithm

- If we only knew which to select up front . . .
- the standard approach: two queues (age, weight) and a ratio
- a natural target for clever heuristics

# How is Clause Selection Traditionally Done?

## Take simple clause evaluation criteria:

- *age*: prefer clauses that were generated long time ago
- *weight*: prefer clauses with fewer symbols

# How is Clause Selection Traditionally Done?

## Take simple clause evaluation criteria:

- *age*: prefer clauses that were generated long time ago
- *weight*: prefer clauses with fewer symbols

## Combine them into a single scheme:

- have a *priority queue* ordering *Passive* for each criterion
- *alternate* between selecting from the queues using a fixed *ratio*

# How is Clause Selection Traditionally Done?

## Take simple clause evaluation criteria:

- *age*: prefer clauses that were generated long time ago
- *weight*: prefer clauses with fewer symbols

## Combine them into a single scheme:

- have a *priority queue* ordering *Passive* for each criterion
- *alternate* between selecting from the queues using a fixed *ratio*

## Example (Organizing *Passive* via two priority queues)

