
# Automatic Theorem Proving with Vampire: Finite Model Finding

Martin Suda

Czech Technical University in Prague, Czech Republic

VTSA 2025, Liege, Belgium

# Motivation

- What if the conjecture *does not follow* from the axioms?

$$A_1, \ldots, A_n \not\models C$$

# Motivation

- What if the conjecture *does not follow* from the axioms?

$$A_1, \ldots, A_n \not\models C$$

- It means there is a "counter-example", i.e., a structure $\mathcal{I}$ s.t.

$$\mathcal{I} \models A_1, \ldots, A_n, \neg C$$

# Motivation

- What if the conjecture *does not follow* from the axioms?

$$A_1, \ldots, A_n \not\models C$$

- It means there is a "counter-example", i.e., a structure $\mathcal{I}$ s.t.

$$\mathcal{I} \models A_1, \ldots, A_n, \neg C$$

- Can we, in such a case, always find it?

# Motivation

- What if the conjecture *does not follow* from the axioms?

$$A_1, \ldots, A_n \not\models C$$

- It means there is a "counter-example", i.e., a structure $\mathcal{I}$ s.t.

$$\mathcal{I} \models A_1, \ldots, A_n, \neg C$$

- Can we, in such a case, always find it?

- What if $\mathcal{I}$ was finite?

# Finite Model Finding

A useful complementary technique to proving:

- debugging specifications (program traces, ontologies, . . . )
- search for finite algebraic structures
- . . .

[1] *New Techniques that Improve MACE-style Finite Model Finding*, Claessen Sörensson

# Finite Model Finding

A useful complementary technique to proving:

- debugging specifications (program traces, ontologies, . . . )
- search for finite algebraic structures
- . . .

In this lecture: "MACE-style" finite model finding

- pioneered by McCune's tool MACE, popularised by Paradox[1]

[1] *New Techniques that Improve MACE-style Finite Model Finding*, Claessen Sörensson

# Finite Model Finding

A useful complementary technique to proving:

- debugging specifications (program traces, ontologies, . . . )
- search for finite algebraic structures
- . . .

In this lecture: "MACE-style" finite model finding

- pioneered by McCune's tool MACE, popularised by Paradox[1]
- Given a set of f.o. clauses $N$ and a model size $s \in \mathbb{N}$, construct a propositional formula $P_s^N$ that is satisfiable if and only if there is an $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ such that $|\mathcal{D}| = s$ and $\mathcal{I} \models N$.

[1] *New Techniques that Improve MACE-style Finite Model Finding*, Claessen Sörensson

# Finite Model Finding

A useful complementary technique to proving:

- debugging specifications (program traces, ontologies, . . . )
- search for finite algebraic structures
- . . .

In this lecture: "MACE-style" finite model finding

- pioneered by McCune's tool MACE, popularised by Paradox[1]
- Given a set of f.o. clauses $N$ and a model size $s \in \mathbb{N}$, construct a propositional formula $P_s^N$ that is satisfiable if and only if there is an $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ such that $|\mathcal{D}| = s$ and $\mathcal{I} \models N$.
- Use a SAT solver to attack problems $P_1^N, P_2^N, P_3^N, \ldots$

[1] *New Techniques that Improve MACE-style Finite Model Finding*, Claessen Sörensson

# Outline

# Unsorted First-Order Logic

Syntax:

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols,

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms;

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

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

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain,

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$,

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$, $p^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{B}$ for $p/n \in \Sigma_P$,

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$, $p^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{B}$ for $p/n \in \Sigma_P$, $x^{\mathcal{I}} \in \mathcal{D}$

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$, $p^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{B}$ for $p/n \in \Sigma_P$, $x^{\mathcal{I}} \in \mathcal{D}$
- interpretation $\mathcal{I}$ evaluates formula $\varphi$ to true: $\mathcal{I} \models \varphi$

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$, $p^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{B}$ for $p/n \in \Sigma_P$, $x^{\mathcal{I}} \in \mathcal{D}$
- interpretation $\mathcal{I}$ evaluates formula $\varphi$ to true: $\mathcal{I} \models \varphi$
- interpretation

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$, $p^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{B}$ for $p/n \in \Sigma_P$, $x^{\mathcal{I}} \in \mathcal{D}$
- interpretation $\mathcal{I}$ evaluates formula $\varphi$ to true: $\mathcal{I} \models \varphi$
- interpretation $\sim$ structure

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$, $p^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{B}$ for $p/n \in \Sigma_P$, $x^{\mathcal{I}} \in \mathcal{D}$
- interpretation $\mathcal{I}$ evaluates formula $\varphi$ to true: $\mathcal{I} \models \varphi$
- interpretation $\sim$ structure $\sim$ model;

# Unsorted First-Order Logic

Syntax:

- Signature $\Sigma = (\Sigma_F, \Sigma_P)$ of func. $\Sigma_F$ and pred. $\Sigma_P$ symbols, each have a fixed arity, i.e. the number of arguments they expect
- a term $t$ is built using first-order variables $x, y, z, \ldots \in \mathcal{V}$ and function symbols $f, g, h, \ldots$ (called constants $a, b, c, \ldots$, if arity 0)
- an atom $A$ is built using predicates symbols $p, q, r, \ldots$ applied to terms; (recall $\approx$ denotes the equality predicate)
- a literal is a possibly negated atom: $A, \neg A, t_1 \not\approx t_2$
- a clause is a (multi-)set of literals understood disjunctively

Semantics:

- $\Sigma$-interpretation $\mathcal{I} = (\mathcal{D}, \mathcal{A})$, $\mathcal{D}$ non-empty domain, $\mathcal{A}$ assignment:
- $f^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{D}$ for $f/n \in \Sigma_F$, $p^{\mathcal{I}} : \mathcal{D}^n \to \mathcal{B}$ for $p/n \in \Sigma_P$, $x^{\mathcal{I}} \in \mathcal{D}$
- interpretation $\mathcal{I}$ evaluates formula $\varphi$ to true: $\mathcal{I} \models \varphi$
- interpretation $\sim$ structure $\sim$ model; called finite, if $\mathcal{D}$ is finite

# Outline

# Only the Size Matters

**Consider the following situation**

Let $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ be a model of a formula $N$ (a set of clauses).

# Only the Size Matters

**Consider the following situation**

Let $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ be a model of a formula $N$ (a set of clauses). Given a set $\mathcal{D}'$ and a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}'$, we can construct a new structure $\mathcal{I}' = (\mathcal{D}', \mathcal{A}')$ setting (for $\mathbf{v}_i \in \mathcal{D}'$)

$$p^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = p^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_m))$$
$$f^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \pi(f^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_n)))$$

# Only the Size Matters

**Consider the following situation**

Let $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ be a model of a formula $N$ (a set of clauses). Given a set $\mathcal{D}'$ and a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}'$, we can construct a new structure $\mathcal{I}' = (\mathcal{D}', \mathcal{A}')$ setting (for $\mathbf{v}_i \in \mathcal{D}'$)

$$p^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = p^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_m))$$
$$f^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \pi(f^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_n)))$$

It is easy to see that $\mathcal{I}' \models N$.

# Only the Size Matters

**Consider the following situation**

Let $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ be a model of a formula $N$ (a set of clauses). Given a set $\mathcal{D}'$ and a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}'$, we can construct a new structure $\mathcal{I}' = (\mathcal{D}', \mathcal{A}')$ setting (for $\mathbf{v}_i \in \mathcal{D}'$)

$$p^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = p^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_m))$$
$$f^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \pi(f^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_n)))$$

It is easy to see that $\mathcal{I}' \models N$. (Better: $\mathcal{I}' \models N$ iff $\mathcal{I} \models N$ for any $N$.)

# Only the Size Matters

**Consider the following situation**

Let $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ be a model of a formula $N$ (a set of clauses). Given a set $\mathcal{D}'$ and a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}'$, we can construct a new structure $\mathcal{I}' = (\mathcal{D}', \mathcal{A}')$ setting (for $\mathbf{v}_i \in \mathcal{D}'$)

$$p^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = p^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_m))$$
$$f^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \pi(f^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_n)))$$

It is easy to see that $\mathcal{I}' \models N$. (Better: $\mathcal{I}' \models N$ iff $\mathcal{I} \models N$ for any $N$.)

## Structure isomorphism

We call $\mathcal{I}$ and $\mathcal{I}'$ isomorphic, whenever there is such a bijection $\pi$.

# Only the Size Matters

**Consider the following situation**

Let $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ be a model of a formula $N$ (a set of clauses). Given a set $\mathcal{D}'$ and a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}'$, we can construct a new structure $\mathcal{I}' = (\mathcal{D}', \mathcal{A}')$ setting (for $\mathbf{v}_i \in \mathcal{D}'$)

$$p^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = p^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_m))$$
$$f^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \pi(f^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_n)))$$

It is easy to see that $\mathcal{I}' \models N$. (Better: $\mathcal{I}' \models N$ iff $\mathcal{I} \models N$ for any $N$.)

## Structure isomorphism

We call $\mathcal{I}$ and $\mathcal{I}'$ isomorphic, whenever there is such a bijection $\pi$.

**Implications**

- When searching for a finite model of size $s \in \mathbb{N}$, we can (arbitrarily) choose $\mathcal{D} = \{1, 2, \ldots, s\}$.

# Only the Size Matters

**Consider the following situation**

Let $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ be a model of a formula $N$ (a set of clauses). Given a set $\mathcal{D}'$ and a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}'$, we can construct a new structure $\mathcal{I}' = (\mathcal{D}', \mathcal{A}')$ setting (for $\mathbf{v}_i \in \mathcal{D}'$)

$$p^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = p^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_m))$$
$$f^{\mathcal{I}'}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \pi(f^{\mathcal{I}}(\pi^{-1}(\mathbf{v}_1), \ldots, \pi^{-1}(\mathbf{v}_n)))$$

It is easy to see that $\mathcal{I}' \models N$. (Better: $\mathcal{I}' \models N$ iff $\mathcal{I} \models N$ for any $N$.)

## Structure isomorphism

We call $\mathcal{I}$ and $\mathcal{I}'$ isomorphic, whenever there is such a bijection $\pi$.

**Implications**

- When searching for a finite model of size $s \in \mathbb{N}$, we can (arbitrarily) choose $\mathcal{D} = \{1, 2, \ldots, s\}$.
- This will also be relevant for us when $\mathcal{D}' = \mathcal{D}$.

# How Shall We Encode into SAT?

# How Shall We Encode into SAT?

To encode an interpretation $\mathcal{I}$ into SAT we will use:

- propositional variables "$p(\mathbf{v}_1, \ldots, \mathbf{v}_m)$" for every $p/m \in \Sigma_P$ and $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathcal{D}$ to represent

$$p^{\mathcal{I}}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = \mathbf{1},$$

# How Shall We Encode into SAT?

To encode an interpretation $\mathcal{I}$ into SAT we will use:

- propositional variables "$p(\mathbf{v}_1, \ldots, \mathbf{v}_m)$" for every $p/m \in \Sigma_P$ and $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathcal{D}$ to represent

$$p^{\mathcal{I}}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = \mathbf{1},$$

- propositional variables "$f(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \mathbf{v}$" for every $f/n \in \Sigma_F$ and $\mathbf{v}, \mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{D}$ to represent

$$f^{\mathcal{I}}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \mathbf{v}.$$

# How Shall We Encode into SAT?

To encode an interpretation $\mathcal{I}$ into SAT we will use:

- propositional variables "$p(\mathbf{v}_1, \ldots, \mathbf{v}_m)$" for every $p/m \in \Sigma_P$ and $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathcal{D}$ to represent

$$p^{\mathcal{I}}(\mathbf{v}_1, \ldots, \mathbf{v}_m) = \mathbf{1},$$

- propositional variables "$f(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \mathbf{v}$" for every $f/n \in \Sigma_F$ and $\mathbf{v}, \mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{D}$ to represent

$$f^{\mathcal{I}}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \mathbf{v}.$$

Using the "quotes" here to stress that the SAT solver sees no structure in these, only the (distinct) names.

# Translating Clauses I

**Flattening**, a necessary step to reduce complex terms:

$$C[t] \quad \longrightarrow \quad t \not\approx x \vee C[x], \qquad x \text{ a fresh variable}$$

applied exhaustively (until "suitable" normal form reached).

# Translating Clauses I

**Flattening**, a necessary step to reduce complex terms:

$$C[t] \quad \longrightarrow \quad t \not\approx x \vee C[x], \qquad x \text{ a fresh variable}$$

applied exhaustively (until "suitable" normal form reached).

## Example (with a trick)

Let's flatten $p(f(a,b), g(f(a,b)))$:

# Translating Clauses I

**Flattening**, a necessary step to reduce complex terms:

$$C[t] \quad \longrightarrow \quad t \not\approx x \vee C[x], \qquad x \text{ a fresh variable}$$

applied exhaustively (until "suitable" normal form reached).

## Example (with a trick)

Let's flatten $p(f(a,b), g(f(a,b)))$:

1. getting $f(a,b) \not\approx x_1 \vee p(x_1, g(x_1))$ using $t = f(a,b)$,

# Translating Clauses I

**Flattening**, a necessary step to reduce complex terms:

$$C[t] \quad \longrightarrow \quad t \not\approx x \vee C[x], \qquad x \text{ a fresh variable}$$

applied exhaustively (until "suitable" normal form reached).

## Example (with a trick)

Let's flatten $p(f(a,b), g(f(a,b)))$:

1. getting $f(a,b) \not\approx x_1 \vee p(x_1, g(x_1))$ using $t = f(a,b)$,
2. getting $g(x_1) \not\approx x_2 \vee f(a,b) \not\approx x_1 \vee p(x_1, x_2)$ using $t = g(x_1)$,

# Translating Clauses I

**Flattening**, a necessary step to reduce complex terms:

$$C[t] \quad \longrightarrow \quad t \not\approx x \vee C[x], \qquad x \text{ a fresh variable}$$

applied exhaustively (until "suitable" normal form reached).

## Example (with a trick)

Let's flatten $p(f(a,b), g(f(a,b)))$:

1. getting $f(a,b) \not\approx x_1 \vee p(x_1, g(x_1))$ using $t = f(a,b)$,
2. getting $g(x_1) \not\approx x_2 \vee f(a,b) \not\approx x_1 \vee p(x_1, x_2)$ using $t = g(x_1)$,
3. $a \not\approx x_3 \vee g(x_1) \not\approx x_2 \vee f(x_3, b) \not\approx x_1 \vee p(x_1, x_2)$ using $t = a$,

# Translating Clauses I

**Flattening**, a necessary step to reduce complex terms:

$$C[t] \quad \longrightarrow \quad t \not\approx x \vee C[x], \qquad x \text{ a fresh variable}$$

applied exhaustively (until "suitable" normal form reached).

## Example (with a trick)

Let's flatten $p(f(a,b), g(f(a,b)))$:

1. getting $f(a,b) \not\approx x_1 \vee p(x_1, g(x_1))$ using $t = f(a,b)$,
2. getting $g(x_1) \not\approx x_2 \vee f(a,b) \not\approx x_1 \vee p(x_1, x_2)$ using $t = g(x_1)$,
3. $a \not\approx x_3 \vee g(x_1) \not\approx x_2 \vee f(x_3, b) \not\approx x_1 \vee p(x_1, x_2)$ using $t = a$,
4. and finally

$$b \not\approx x_4 \vee a \not\approx x_3 \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee p(x_1, x_2)$$

using $t = b$.

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1"$$

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$\text{"}p(1,1)\text{"} \vee \neg\text{"}g(1) \approx 1\text{"} \vee \neg\text{"}f(1,1) \approx 1\text{"} \vee \neg\text{"}a \approx 1\text{"} \vee \neg\text{"}b \approx 1\text{"}$$
$$\text{"}p(2,1)\text{"} \vee \neg\text{"}g(2) \approx 1\text{"} \vee \neg\text{"}f(1,1) \approx 2\text{"} \vee \neg\text{"}a \approx 1\text{"} \vee \neg\text{"}b \approx 1\text{"}$$

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$\begin{aligned}
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,1)" \vee \neg "g(2) \approx 1" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,2)" \vee \neg "g(1) \approx 2" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1"
\end{aligned}$$

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$\begin{aligned}
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,1)" \vee \neg "g(2) \approx 1" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,2)" \vee \neg "g(1) \approx 2" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,2)" \vee \neg "g(2) \approx 2" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1"
\end{aligned}$$

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$\begin{aligned}
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,1)" \vee \neg "g(2) \approx 1" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,2)" \vee \neg "g(1) \approx 2" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,2)" \vee \neg "g(2) \approx 2" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(2,1) \approx 1" \vee \neg "a \approx 2" \vee \neg "b \approx 1"
\end{aligned}$$

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$\begin{aligned}
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,1)" \vee \neg "g(2) \approx 1" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,2)" \vee \neg "g(1) \approx 2" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,2)" \vee \neg "g(2) \approx 2" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(2,1) \approx 1" \vee \neg "a \approx 2" \vee \neg "b \approx 1" \\
&\ldots
\end{aligned}$$

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$\begin{aligned}
&"p(1,1)" \vee \neg"g(1) \approx 1" \vee \neg" f(1,1) \approx 1" \vee \neg"a \approx 1" \vee \neg"b \approx 1" \\
&"p(2,1)" \vee \neg"g(2) \approx 1" \vee \neg" f(1,1) \approx 2" \vee \neg"a \approx 1" \vee \neg"b \approx 1" \\
&"p(1,2)" \vee \neg"g(1) \approx 2" \vee \neg" f(1,1) \approx 1" \vee \neg"a \approx 1" \vee \neg"b \approx 1" \\
&"p(2,2)" \vee \neg"g(2) \approx 2" \vee \neg" f(1,1) \approx 2" \vee \neg"a \approx 1" \vee \neg"b \approx 1" \\
&"p(1,1)" \vee \neg"g(1) \approx 1" \vee \neg" f(2,1) \approx 1" \vee \neg"a \approx 2" \vee \neg"b \approx 1"
\end{aligned}$$

. . . number of clauses in total:

# Translating Clauses II

**Instantiation** (in all possible ways):

- For each flattened clause $C$ and
- each "substitution" $\sigma : Var(C) \to \mathcal{D}$

generate the "propositional" clause $C\sigma$.

Note: this yields $|\mathcal{D}|^k$ instances for every clause with $k$ variables!

Example (the clause from last example, domain size 2)

$$p(x_1, x_2) \vee g(x_1) \not\approx x_2 \vee f(x_3, x_4) \not\approx x_1 \vee a \not\approx x_3 \vee b \not\approx x_4$$

$$\begin{aligned}
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,1)" \vee \neg "g(2) \approx 1" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,2)" \vee \neg "g(1) \approx 2" \vee \neg "f(1,1) \approx 1" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(2,2)" \vee \neg "g(2) \approx 2" \vee \neg "f(1,1) \approx 2" \vee \neg "a \approx 1" \vee \neg "b \approx 1" \\
&"p(1,1)" \vee \neg "g(1) \approx 1" \vee \neg "f(2,1) \approx 1" \vee \neg "a \approx 2" \vee \neg "b \approx 1"
\end{aligned}$$

. . . number of clauses in total: 16

# Not Done Yet With the Encoding

# Not Done Yet With the Encoding

The problem: functions have been encoded as predicates!

# Not Done Yet With the Encoding

The problem: functions have been encoded as predicates!

**Functionality Axioms:** For each $f/n \in \Sigma_F$, arguments $\mathbf{v}_1, \ldots, \mathbf{v}_n$, and two distinct domain elements $\mathbf{v}$ and $\mathbf{v}'$ translate and add

$$f(\mathbf{v}_1, \ldots, \mathbf{v}_n) \not\approx \mathbf{v} \vee f(\mathbf{v}_1, \ldots, \mathbf{v}_n) \not\approx \mathbf{v}'.$$

*A function cannot return two different values for the same arguments.*

# Not Done Yet With the Encoding

The problem: functions have been encoded as predicates!

**Functionality Axioms:** For each $f/n \in \Sigma_F$, arguments $\mathbf{v}_1, \ldots, \mathbf{v}_n$, and two distinct domain elements $\mathbf{v}$ and $\mathbf{v}'$ translate and add

$$f(\mathbf{v}_1, \ldots, \mathbf{v}_n) \not\approx \mathbf{v} \vee f(\mathbf{v}_1, \ldots, \mathbf{v}_n) \not\approx \mathbf{v}'.$$

*A function cannot return two different values for the same arguments.*

**Totality Axioms:** For each $f/n \in \Sigma_F$ and arguments $\mathbf{v}_1, \ldots, \mathbf{v}_n$, having $\mathcal{D} = \{1, \ldots, s\}$, translate and add

$$f(\mathbf{v}_1, \ldots, \mathbf{v}_n) \approx 1 \vee \ldots \vee f(\mathbf{v}_1, \ldots, \mathbf{v}_n) \approx s.$$

*A function must return at least one value for each argument tuple.*

# Example

Given:

$$p(a), \qquad f(x) \not\approx f(y) \vee x \approx y, \qquad f(f(x)) \approx x.$$

# Example

Given:

$$p(a), \qquad f(x) \not\approx f(y) \vee x \approx y, \qquad f(f(x)) \approx x.$$

Flattened:

$$p(x) \vee a \not\approx x, \quad f(x) \not\approx z \vee f(y) \not\approx z \vee x \approx y, \quad f(y) \approx x \vee f(x) \not\approx y$$

# Example

Given:

$$p(a), \qquad f(x) \not\approx f(y) \vee x \approx y, \qquad f(f(x)) \approx x.$$

Flattened:

$$p(x) \vee a \not\approx x, \quad f(x) \not\approx z \vee f(y) \not\approx z \vee x \approx y, \quad f(y) \approx x \vee f(x) \not\approx y$$

Instances for $s = 2$, $\mathcal{D} = \{1, 2\}$:

$p(1) \vee b \not\approx 1$
$p(2) \vee b \not\approx 2$

$f(1) \not\approx 1 \vee f(1) \not\approx 1 \vee 1 \approx 1$
$f(2) \not\approx 1 \vee f(1) \not\approx 1 \vee 2 \approx 1$
$f(1) \not\approx 1 \vee f(2) \not\approx 1 \vee 1 \approx 2$
$f(2) \not\approx 1 \vee f(2) \not\approx 1 \vee 2 \approx 2$
$f(1) \not\approx 2 \vee f(1) \not\approx 2 \vee 1 \approx 1$
$f(2) \not\approx 2 \vee f(1) \not\approx 2 \vee 2 \approx 1$
$f(1) \not\approx 2 \vee f(2) \not\approx 2 \vee 1 \approx 2$
$f(2) \not\approx 2 \vee f(2) \not\approx 2 \vee 2 \approx 2$

$f(1) \approx 1 \vee f(1) \not\approx 1$
$f(1) \approx 2 \vee f(2) \not\approx 1$
$f(2) \approx 2 \vee f(2) \not\approx 2$
$f(2) \approx 1 \vee f(1) \not\approx 2$

# Example

Given:

$$p(a), \qquad f(x) \not\approx f(y) \vee x \approx y, \qquad f(f(x)) \approx x.$$

Flattened:

$$p(x) \vee a \not\approx x, \quad f(x) \not\approx z \vee f(y) \not\approx z \vee x \approx y, \quad f(y) \approx x \vee f(x) \not\approx y$$

Instances for $s = 2$, $\mathcal{D} = \{1, 2\}$:

$p(1) \vee b \not\approx 1$
$p(2) \vee b \not\approx 2$

~~$f(1) \not\approx 1 \vee f(1) \not\approx 1 \vee 1 \approx 1$~~
$f(2) \not\approx 1 \vee f(1) \not\approx 1 \vee 2 \approx 1$
$f(1) \not\approx 1 \vee f(2) \not\approx 1 \vee 1 \approx 2$
~~$f(2) \not\approx 1 \vee f(2) \not\approx 1 \vee 2 \approx 2$~~
~~$f(1) \not\approx 2 \vee f(1) \not\approx 2 \vee 1 \approx 1$~~
$f(2) \not\approx 2 \vee f(1) \not\approx 2 \vee 2 \approx 1$
$f(1) \not\approx 2 \vee f(2) \not\approx 2 \vee 1 \approx 2$
~~$f(2) \not\approx 2 \vee f(2) \not\approx 2 \vee 2 \approx 2$~~

$f(1) \approx 1 \vee f(1) \not\approx 1$
$f(1) \approx 2 \vee f(2) \not\approx 1$
$f(2) \approx 2 \vee f(2) \not\approx 2$
$f(2) \approx 1 \vee f(1) \not\approx 2$

# Example

Given:

$$p(a), \qquad f(x) \not\approx f(y) \vee x \approx y, \qquad f(f(x)) \approx x.$$

Flattened:

$$p(x) \vee a \not\approx x, \quad f(x) \not\approx z \vee f(y) \not\approx z \vee x \approx y, \quad f(y) \approx x \vee f(x) \not\approx y$$

Instances for $s = 2$, $\mathcal{D} = \{1, 2\}$:

$$\begin{array}{lll} p(1) \vee b \not\approx 1 & f(2) \not\approx 1 \vee f(1) \not\approx 1 & f(1) \approx 1 \vee f(1) \not\approx 1 \\ p(2) \vee b \not\approx 2 & f(1) \not\approx 1 \vee f(2) \not\approx 1 & f(1) \approx 2 \vee f(2) \not\approx 1 \\ & f(2) \not\approx 2 \vee f(1) \not\approx 2 & f(2) \approx 2 \vee f(2) \not\approx 2 \\ & f(1) \not\approx 2 \vee f(2) \not\approx 2 & f(2) \approx 1 \vee f(1) \not\approx 2 \end{array}$$

# Example

Given:

$$p(a), \qquad f(x) \not\approx f(y) \vee x \approx y, \qquad f(f(x)) \approx x.$$

Flattened:

$$p(x) \vee a \not\approx x, \quad f(x) \not\approx z \vee f(y) \not\approx z \vee x \approx y, \quad f(y) \approx x \vee f(x) \not\approx y$$

Instances for $s = 2$, $\mathcal{D} = \{1, 2\}$:

$$\begin{array}{lll} p(1) \vee b \not\approx 1 & f(2) \not\approx 1 \vee f(1) \not\approx 1 & f(1) \approx 1 \vee f(1) \not\approx 1 \\ p(2) \vee b \not\approx 2 & f(1) \not\approx 1 \vee f(2) \not\approx 1 & f(1) \approx 2 \vee f(2) \not\approx 1 \\ & f(2) \not\approx 2 \vee f(1) \not\approx 2 & f(2) \approx 2 \vee f(2) \not\approx 2 \\ & f(1) \not\approx 2 \vee f(2) \not\approx 2 & f(2) \approx 1 \vee f(1) \not\approx 2 \end{array}$$

Functionality:

$$a \not\approx 1 \vee a \not\approx 2, \quad f(1) \not\approx 1 \vee f(1) \not\approx 2, \quad f(2) \not\approx 1 \vee f(2) \not\approx 2$$

# Example

Given:

$$p(a), \qquad f(x) \not\approx f(y) \vee x \approx y, \qquad f(f(x)) \approx x.$$

Flattened:

$$p(x) \vee a \not\approx x, \quad f(x) \not\approx z \vee f(y) \not\approx z \vee x \approx y, \quad f(y) \approx x \vee f(x) \not\approx y$$

Instances for $s = 2$, $\mathcal{D} = \{1, 2\}$:

$$\begin{array}{lll} p(1) \vee b \not\approx 1 & f(2) \not\approx 1 \vee f(1) \not\approx 1 & f(1) \approx 1 \vee f(1) \not\approx 1 \\ p(2) \vee b \not\approx 2 & f(1) \not\approx 1 \vee f(2) \not\approx 1 & f(1) \approx 2 \vee f(2) \not\approx 1 \\ & f(2) \not\approx 2 \vee f(1) \not\approx 2 & f(2) \approx 2 \vee f(2) \not\approx 2 \\ & f(1) \not\approx 2 \vee f(2) \not\approx 2 & f(2) \approx 1 \vee f(1) \not\approx 2 \end{array}$$

Functionality:

$$a \not\approx 1 \vee a \not\approx 2, \quad f(1) \not\approx 1 \vee f(1) \not\approx 2, \quad f(2) \not\approx 1 \vee f(2) \not\approx 2$$

Totality:

$$a \approx 1 \vee a \approx 2, \quad f(1) \approx 1 \vee f(1) \approx 2, \quad f(2) \approx 1 \vee f(2) \approx 2$$

# Outline

# Reducing the Number of Variables in Clauses

Let us massage the FOL problem a bit before instantiation:

# Reducing the Number of Variables in Clauses

Let us massage the FOL problem a bit before instantiation:

**Definition introduction** to reduce the number of variables that we would need to add during flattening.
A clause $p(f(a,b), g(f(a,b)))$ becomes $p(c_1, c_2)$ and we add two definition clauses $c_1 \approx f(a,b)$ and $c_2 \approx g(c_1)$.

# Reducing the Number of Variables in Clauses

Let us massage the FOL problem a bit before instantiation:

**Definition introduction** to reduce the number of variables that we would need to add during flattening.
A clause $p(f(a,b), g(f(a,b)))$ becomes $p(c_1, c_2)$ and we add two definition clauses $c_1 \approx f(a,b)$ and $c_2 \approx g(c_1)$.

- Added two new constants $c_1, c_2$ into $\Sigma_F$
- The definitions are global $\Rightarrow$ share them across clauses.

# Reducing the Number of Variables in Clauses

Let us massage the FOL problem a bit before instantiation:

**Definition introduction** to reduce the number of variables that we would need to add during flattening.
A clause $p(f(a,b), g(f(a,b)))$ becomes $p(c_1, c_2)$ and we add two definition clauses $c_1 \approx f(a,b)$ and $c_2 \approx g(c_1)$.

- Added two new constants $c_1, c_2$ into $\Sigma_F$
- The definitions are global $\Rightarrow$ share them across clauses.

**Splitting** reduces the number of variables in clauses.
E.g., the clause $p(x,y) \vee q(y,z)$ is transformed to two new clauses

$$p(x,y) \vee s(y) \text{ and } \neg s(y) \vee q(y,z).$$

# Reducing the Number of Variables in Clauses

Let us massage the FOL problem a bit before instantiation:

**Definition introduction** to reduce the number of variables that we would need to add during flattening.
A clause $p(f(a,b), g(f(a,b)))$ becomes $p(c_1, c_2)$ and we add two definition clauses $c_1 \approx f(a,b)$ and $c_2 \approx g(c_1)$.

- Added two new constants $c_1, c_2$ into $\Sigma_F$
- The definitions are global $\Rightarrow$ share them across clauses.

**Splitting** reduces the number of variables in clauses.
E.g., the clause $p(x,y) \vee q(y,z)$ is transformed to two new clauses

$$p(x,y) \vee s(y) \text{ and } \neg s(y) \vee q(y,z).$$

Again, a new symbol is introduced into the signature (here $s/1 \in \Sigma_P$).

# Outline

# What Are Symmetries?

**Summary so far:** Given a FOL problem $N$ over $\Sigma$ and $s \in \mathbb{N}$, we have a propositional problem $P_s^N$ s.t. there is a 1-to-1 correspondence between prop. assignments $V$ with $V \models P_s^N$ and interpretations $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ with $\mathcal{I} \models N$ and $|\mathcal{D}| = s$.

# What Are Symmetries?

**Summary so far:** Given a FOL problem $N$ over $\Sigma$ and $s \in \mathbb{N}$, we have a propositional problem $P_s^N$ s.t. there is a 1-to-1 correspondence between prop. assignments $V$ with $V \models P_s^N$ and interpretations $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ with $\mathcal{I} \models N$ and $|\mathcal{D}| = s$.

Convention: In fact, let us assume $\mathcal{D} = \{1, \ldots, s\}$.

# What Are Symmetries?

**Summary so far:** Given a FOL problem $N$ over $\Sigma$ and $s \in \mathbb{N}$, we have a propositional problem $P_s^N$ s.t. there is a 1-to-1 correspondence between prop. assignments $V$ with $V \models P_s^N$ and interpretations $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ with $\mathcal{I} \models N$ and $|\mathcal{D}| = s$.

Convention: In fact, let us assume $\mathcal{D} = \{1, \ldots, s\}$.

**A potential source of inefficiency:** isomorphic interpretations!

# What Are Symmetries?

**Summary so far:** Given a FOL problem $N$ over $\Sigma$ and $s \in \mathbb{N}$, we have a propositional problem $P_s^N$ s.t. there is a 1-to-1 correspondence between prop. assignments $V$ with $V \models P_s^N$ and interpretations $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ with $\mathcal{I} \models N$ and $|\mathcal{D}| = s$.

Convention: In fact, let us assume $\mathcal{D} = \{1, \ldots, s\}$.

**A potential source of inefficiency:** isomorphic interpretations!

- Recall: a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}$ simply relabels the domain elements; isomorphic $\mathcal{I}$ and $\mathcal{I}'$ are "essentially the same"

# What Are Symmetries?

**Summary so far:** Given a FOL problem $N$ over $\Sigma$ and $s \in \mathbb{N}$, we have a propositional problem $P_s^N$ s.t. there is a 1-to-1 correspondence between prop. assignments $V$ with $V \models P_s^N$ and interpretations $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ with $\mathcal{I} \models N$ and $|\mathcal{D}| = s$.

Convention: In fact, let us assume $\mathcal{D} = \{1, \ldots, s\}$.

**A potential source of inefficiency:** isomorphic interpretations!

- Recall: a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}$ simply relabels the domain elements; isomorphic $\mathcal{I}$ and $\mathcal{I}'$ are "essentially the same"
- Why could that be a problem?

# What Are Symmetries?

**Summary so far:** Given a FOL problem $N$ over $\Sigma$ and $s \in \mathbb{N}$, we have a propositional problem $P_s^N$ s.t. there is a 1-to-1 correspondence between prop. assignments $V$ with $V \models P_s^N$ and interpretations $\mathcal{I} = (\mathcal{D}, \mathcal{A})$ with $\mathcal{I} \models N$ and $|\mathcal{D}| = s$.

Convention: In fact, let us assume $\mathcal{D} = \{1, \ldots, s\}$.

**A potential source of inefficiency:** isomorphic interpretations!

- Recall: a bijection $\pi : \mathcal{D} \leftrightarrow \mathcal{D}$ simply relabels the domain elements; isomorphic $\mathcal{I}$ and $\mathcal{I}'$ are "essentially the same"
- Why could that be a problem?
- In the satisfiable case? In the unsatisfiable one?

**Inspiration: the least number heuristic**

- origin: the "SEM-style" model finding methods, i.e., explicit search for the truth tables

# Symmetry Reduction in Finite Model Finding

**Inspiration: the least number heuristic**

- origin: the "SEM-style" model finding methods,
  i.e., explicit search for the truth tables

**Intuition:**

- constructs an interpretation $\mathcal{I} \models N$ incrementally
- for each decision, it only ever picks the *least* domain element
  (from each considered equivalence class)

# Static Symmetry Reduction

**For our SAT encoding: a static analogue of LNH**

- Let $a_1, a_2, \ldots, a_k$ be an enumeration of the constants of our $\Sigma_F$
- Constraints of type I: "only expand when needed"

$$
\begin{aligned}
& a_1 \approx 1, \\
& a_2 \approx 1 \vee a_2 \approx 2, \\
& a_3 \approx 1 \vee a_3 \approx 2 \vee a_3 \approx 3, \\
& \ldots
\end{aligned}
$$

# Static Symmetry Reduction

**For our SAT encoding: a static analogue of LNH**

- Let $a_1, a_2, \ldots, a_k$ be an enumeration of the constants of our $\Sigma_F$
- Constraints of type I: "only expand when needed"

$$
\begin{aligned}
& a_1 \approx 1, \\
& a_2 \approx 1 \vee a_2 \approx 2, \\
& a_3 \approx 1 \vee a_3 \approx 2 \vee a_3 \approx 3, \\
& \ldots
\end{aligned}
$$

These actually subsume the totality clauses for their constants!

# Static Symmetry Reduction

**For our SAT encoding: a static analogue of LNH**

- Let $a_1, a_2, \ldots, a_k$ be an enumeration of the constants of our $\Sigma_F$
- Constraints of type I: "only expand when needed"

$$
\begin{aligned}
& a_1 \approx 1, \\
& a_2 \approx 1 \vee a_2 \approx 2, \\
& a_3 \approx 1 \vee a_3 \approx 2 \vee a_3 \approx 3, \\
& \ldots
\end{aligned}
$$

  These actually subsume the totality clauses for their constants!

- Constraints of type II: "don't leave any gaps":

$$a_i \not\approx d \vee a_1 \approx (d-1) \vee a_2 \approx (d-1) \vee a_{i-1} \approx (d-1)$$

  for any $1 < i \in \mathcal{D}$ and $1 < d \leq i \in \mathcal{D}$

# Static Symmetry Reduction

**For our SAT encoding: a static analogue of LNH**

- Let $a_1, a_2, \ldots, a_k$ be an enumeration of the constants of our $\Sigma_F$
- Constraints of type I: "only expand when needed"

$$
\begin{aligned}
& a_1 \approx 1, \\
& a_2 \approx 1 \vee a_2 \approx 2, \\
& a_3 \approx 1 \vee a_3 \approx 2 \vee a_3 \approx 3, \\
& \ldots
\end{aligned}
$$

  These actually subsume the totality clauses for their constants!

- Constraints of type II: "don't leave any gaps":

$$a_i \not\approx d \vee a_1 \approx (d-1) \vee a_2 \approx (d-1) \vee a_{i-1} \approx (d-1)$$

  for any $1 < i \in \mathcal{D}$ and $1 < d \leq i \in \mathcal{D}$

Note: not just for constants: $\ldots, f(1), f(2), \ldots, g(1,1), g(2,2), \ldots$

# Outline

# Further Tricks and Extensions

**Sort Inference**

- Try to look for implicit sorts (in our unsorted problem)
- Can then apply symmetry reduction for each sort separately
- Some sorts do not need to be grown over certain size

# Further Tricks and Extensions

**Sort Inference**

- Try to look for implicit sorts (in our unsorted problem)
- Can then apply symmetry reduction for each sort separately
- Some sorts do not need to be grown over certain size

**Deciding Certain Fragments**

- e.g., Bernays-Schönfinkel-Ramsey class:
  in CNF and all function symbols are constants
- No need to grow $s$ beyond the number of constants!

# Further Tricks and Extensions

**Sort Inference**

- Try to look for implicit sorts (in our unsorted problem)
- Can then apply symmetry reduction for each sort separately
- Some sorts do not need to be grown over certain size

**Deciding Certain Fragments**

- e.g., Bernays-Schönfinkel-Ramsey class:
  in CNF and all function symbols are constants
- No need to grow $s$ beyond the number of constants!

**Handling Multi-sorted Input**

- A relatively straightforward extension, but:
- . . . which domain to grow next?

# Outline

# References

- *New Techniques that Improve MACE-style Finite Model Finding*, Claessen, Sörensson
  - These slides were mostly based on this!

# References


- *New Techniques that Improve MACE-style Finite Model Finding*, Claessen, Sörensson
  - These slides were mostly based on this!

- *Finding Finite Models in Multi-sorted First-Order Logic*, Reger, Suda, Voronkov
  - multi-sorted tricks
- *Symmetry Avoidance in MACE-Style Finite Model Finding*, Reger, Riener, Suda
  - more on symmetries
- *Towards Smarter MACE-style Model Finders*, Janota, Suda
  - a CEGAR-based (lazy) approach
- *Finite Model Finding in SMT*, Reynolds et al.
  - FMB in SMT