# Overview of automated reasoning and ordering-based strategies

Maria Paola Bonacina

**Visiting:** Computer Science Laboratory, SRI International, Menlo Park, CA, USA
**Affiliation:** Dipartimento di Informatica, Università degli Studi di Verona, Verona, Italy, EU

May 19, 2019

# Automated reasoning

Automated reasoning is

- Symbolic computation
- Artificial intelligence
- Computational logic
- ...

- Knowledge described precisely: symbols
- Symbolic reasoning: Logico-deductive, Probabilistic ...

# The gist of this lecture

- Logico-deductive reasoning
- Focus: first-order logic (FOL)
- Take-home message:
  - FOL as machine language
  - Reasoning is about ignoring what's redundant as much as it is getting what's relevant
  - Expansion and Contraction
  - Ordering-based, instance-based, subgoal-reduction-based strategies
  - Inference, Search, and algorithmic building blocks

# Signature

- A finite set of constant symbols: $a$, $b$, $c$ ...
- A finite set of function symbols: $f$, $g$, $h$ ...
- A finite set of predicate symbols: $P$, $Q$, $\simeq$ ...
- Arities
- Sorts (important but key concepts can be understood without)

An infinite supply of variable symbols: $x$, $y$, $z$, $w$ ...

# Defined symbols and free symbols

- A symbol is defined if it comes with axioms, e.g., $\simeq$
- It is free otherwise, e.g., $P$
- Aka: interpreted/uninterpreted
- Equality ($\simeq$) comes with the congruence axioms

# Terms and atoms

- Terms: $a$, $x$, $f(a, b)$, $g(y)$
- Herbrand universe $\mathcal{U}$: all ground terms
  (add a constant if there is none in the given signature)
- Atoms: $P(a)$, $f(x, x) \simeq x$
- Literals: $P(a)$, $f(x, x) \simeq x$, $\neg P(a)$, $f(x, x) \not\simeq x$
- Herbrand base $\mathcal{B}$: all ground atoms
- If there is at least one function symbol, $\mathcal{U}$ and $\mathcal{B}$ are infinite
- This is key if the reasoner builds new terms and atoms

# Substitution

- A substitution is a function from variables to terms that is not identity on a finite set of variables
- $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$
- $\sigma = \{x \leftarrow a, y \leftarrow f(w), z \leftarrow w\}$
- Application: $h(x, y, z)\sigma = h(a, f(w), w)$

# Matching

- Given terms or atoms $s$ and $t$
- $f(x, g(y))$ and $f(g(b), g(a))$
- Find matching substitution: $\sigma$ s.t. $s\sigma = t$
  $\sigma = \{x \leftarrow g(b), y \leftarrow a\}$
- $s\sigma = t$: $t$ is instance of $s$, $s$ is more general than $t$

# Unification

- Given terms or atoms $s$ and $t$
- $f(g(z), g(y))$ and $f(x, g(a))$
- Find substitution $\sigma$ s.t. $s\sigma = t\sigma$:
  $\sigma = \{x \leftarrow g(z), y \leftarrow a\}$
- Unification problem: $E = \{s_i =^? t_i\}_{i=1}^n$
- Most general unifier (mgu): e.g., not
  $\sigma' = \{x \leftarrow g(b), y \leftarrow a, z \leftarrow b\}$

# Orderings

- View $\mathcal{U}$ and $\mathcal{B}$ as ordered sets
- With variables: partial order
- Extend to literals (add sign) and clauses
- Extend to proofs (e.g., equational chains)
- Why? To detect and delete or replace redundant data
- E.g., replace something by something smaller in a well-founded ordering

# Precedence

- A partial order $>$ on the signature
- Example: the Ackermann function
  - $ack(0, y) \simeq succ(y)$
  - $ack(succ(x), 0) \simeq ack(x, succ(0))$
  - $ack(succ(x), succ(y)) \simeq ack(x, ack(succ(x), y))$
- Precedence $ack > succ > 0$

# Stability

- $\succ$ ordering
- $s \succ t$
- $f(f(x, y), z) \succ f(x, f(y, z))$
- Stability: $s\sigma \succ t\sigma$ for all substitutions $\sigma$
- $f(f(g(a), b), z) \succ f(g(a), f(b, z))$
  $\sigma = \{x \leftarrow g(a), y \leftarrow b\}$

# Monotonicity

- $\succ$ ordering
- $s \succ t$
- Example: $f(x, i(x)) \succ e$
- Monotonicity: $r[s] \succ r[t]$ for all contexts $r$
  (A context is an expression, here a term or atom, with a hole)
- $f(f(x, i(x)), y) \succ f(e, y)$

# Subterm property

- $\succ$ ordering
- $s[t] \succ t$
- Example: $f(x, i(x)) \succ i(x)$

# Simplification ordering

- Stable, monotonic, and with the subterm property: simplification ordering
- A simplification ordering is well-founded or equivalently Noetherian
- No infinite descending chain $s_0 \succ s_1 \succ \ldots s_i \succ s_{i+1} \succ \ldots$

(Noetherian from Emmy Noether)

# Multiset extension

- Multisets, e.g., $\{a, a, b\}$, $\{5, 4, 4, 4, 3, 1, 1\}$
- From $\succ$ to $\succ_{mul}$:
  - $M \succ_{mul} \emptyset$
  - $M \cup \{a\} \succ_{mul} N \cup \{a\}$ if $M \succ_{mul} N$
  - $M \cup \{a\} \succ_{mul} N \cup \{b\}$ if $a \succ b$ and $M \cup \{a\} \succ_{mul} N$
- $\{5\} \succ_{mul} \{4, 4, 4, 3, 1, 1\}$
- If $\succ$ is well-founded then $\succ_{mul}$ is well-founded

# Recursive path ordering (RPO)

$s = f(s_1, \ldots, s_n) \succ g(t_1, \ldots, t_m) = t$ if

- Either $f > g$ and $\forall k$, $1 \leq k \leq m$, $s \succ t_k$
- Or $f = g$ and $\{s_1, \ldots, s_n\} \succ_{mul} \{t_1, \ldots, t_n\}$
- Or $\exists k$ such that $s_k \succeq t$

# Distributivity by RPO

- Precedence: $* > +$
- $x * (y + z) \succ x * y + x * z$ because
  - $* > +$ and
  - $x * (y + z) \succ x * y$ since $\{x, y + z\} \succ_{mul} \{x, y\}$
  - $x * (y + z) \succ x * z$ since $\{x, y + z\} \succ_{mul} \{x, z\}$

# Lexicographic extension

- Tuples, vectors, words, e.g., $(a, a, b)$, $(5, 4, 4, 4, 3, 1, 1)$
- From $\succ$ to $\succ_{lex}$:
  $(a_1, \ldots, a_n) \succ_{lex} (b_1, \ldots, b_m)$ if $\exists i$ s.t. $\forall j,\ 1 \leq j < i,\ a_j = b_j$, and $a_i \succ b_i$
- $(5) \succ_{lex} (4, 4, 4, 3, 1, 1)$
- $(1, 2, 3, 5, 1) \succ_{lex} (1, 2, 3, 3, 4)$
- If $\succ$ is well-founded then $\succ_{lex}$ is well-founded

# Lexicographic path ordering (LPO)

$s = f(s_1, \ldots, s_n) \succ g(t_1, \ldots, t_m) = t$ if

- Either $f > g$ and $\forall k$, $1 \leq k \leq m$, $s \succ t_k$
- Or $f = g$, $(s_1, \ldots, s_n) \succ_{lex} (t_1, \ldots, t_n)$, and $\forall k$, $i < k \leq n$, $s \succ t_k$
- Or $\exists k$ such that $s_k \succeq t$

Multiset and lexicographic extension can be mixed: give each function symbol either multiset or lexicographic status

# Ackermann function by LPO

- Precedence $ack > succ > 0$
- $ack(0, y) \succ succ(y)$
  because $ack > succ$ and $ack(0, y) \succ y$
- $ack(succ(x), 0) \succ ack(x, succ(0))$
  because $(succ(x), 0) \succ_{lex} (x, succ(0))$,
  as $succ(x) \succ x$, and $ack(succ(x), 0) \succ succ(0)$,
  since $ack > succ$ and $ack(succ(x), 0) \succ 0$
- $ack(succ(x), succ(y)) \succ ack(x, ack(succ(x), y))$
  because $(succ(x), succ(y)) \succ_{lex} (x, ack(succ(x), y))$,
  since $succ(x) \succ x$ and $ack(succ(x), succ(y)) \succ ack(succ(x), y)$,
  because $(succ(x), succ(y)) \succ_{lex} (succ(x), y)$,
  as $succ(x) = succ(x)$ and $succ(y) \succ y$

# Ordering atoms and literals

- Atoms are treated like terms
  - Also predicate symbols in the precedence $>$
  - $\simeq$ is typically the smallest predicate symbol in $>$
  - $\simeq$ has multiset status: $s \simeq t$ as $\{s, t\}$
- Literals: make the positive version smaller than the negative
  - Add $\top$ and $\bot$ both $>$-smaller than any other symbol and with $\bot > \top$
  - For literal $L$ take multiset $\{atom(L), \bot\}$ if $L$ negative, $\{atom(L), \top\}$, otherwise

# Variables cause partiality

- Let $s$ and $t$ be two distinct non-ground terms or atoms
- If $\exists x \in Var(s) \setminus Var(t)$ then $t \not\succ s$
- $g(x) \not\succ f(x, y)$
- If $\exists y \in Var(t) \setminus Var(s)$ then $s \not\succ t$
- Both: $t\#s$ (uncomparable)
- $f(x)\#g(y)$, $f(x)\#f(y)$, $g(x, z)\#f(x, y)$

# Complete simplification ordering (CSO)

- LPO and RPO are simplification orderings
- Simplification ordering total on ground terms and atoms: complete simplification ordering (CSO)
- LPO and RPO with a total precedence are CSO
- LPO and RPO do not correlate with size e.g., $f(a) \succ g^5(a)$ if $f > g$
- Knuth-Bendix ordering (KBO): based on precedence and a weight function

# Summary of the first part

- Language: signature, terms, atoms, literals
- Substitutions instantiate variables
- Matching and unification
- A partially ordered world of terms, atoms, literals
- More building blocks: indexing to detect matching and unification fast

## At the dawn of computer science

- Kurt Gödel: completeness of first-order logic
  Later: Leon Henkin (consistency implies satisfiability)
- Alan Turing: Entscheidungsproblem; "computor;" Turing machine; universal computer; halting problem; undecidability; undecidability of first-order logic
- Herbrand theorem: semi-decidability of first-order logic

(Herbrand theorem: Jacques Herbrand + Thoralf Skolem + Kurt Gödel)

("Computor:" Robert I. Soare "Computability and recursion" Bulletin of Symbolic Logic 2:284–321, 1996)

# The theorem-proving problem

- A set $H$ of formulas viewed as assumptions or hypotheses
- A formula $\varphi$ viewed as conjecture
- Theorem-proving problem: $H \models^? \varphi$
- Equivalently: is $H \cup \{\neg\varphi\}$ unsatisfiable?
- If $H \models \varphi$, then $\varphi$ is a theorem of $H$, or $H \supset \varphi$ is a theorem
- $Th(H) = \{\varphi \colon H \models \varphi\}$
- Infinitely many interpretations on infinitely many domains: how do we start?

# Two simplifications

- Restrict formulas to clauses: less expressive, but suitable as machine language
- Restrict interpretations to Herbrand interpretations: a semantics built out of syntax
- All we have in machine's memory are symbols, that is, syntax

# Clausal form

- Clause: disjunction of literals where all variables are implicitly universally quantified
- $\neg P(f(z)) \vee \neg Q(g(z)) \vee R(f(z), g(z))$
- Ordering $\succ$ on literals extended to clauses by multiset extension
- No loss of generality: every formula can be transformed into an equisatisfiable set of clauses
- Every clause has its own variables

# Transformation into clausal form

- Eliminate $\equiv$ and $\supset$: $F \equiv G$ becomes $(F \supset G) \wedge (G \supset F)$ and $F \supset G$ becomes $\neg F \vee G$
- Reduce the scope of all occurrences of $\neg$ to an atom: (each quantifier occurrence binds a distinct variable$\neg(F \vee G)$ becomes $\neg F \wedge \neg G$, $\neg(F \wedge G)$ becomes $\neg F \vee \neg G$, $\neg\neg F$ becomes $F$, $\neg\exists F$ becomes $\forall\neg F$, and $\neg\forall F$ becomes $\exists\neg F$
- Standardize variables apart
  (each quantifier occurrence binds a distinct variable symbol)
- Skolemize $\exists$ and then drop $\forall$
- Distributivity and associativity: $F \vee (G \wedge H)$ becomes $(F \vee G) \wedge (F \vee H)$ and $F \vee (G \vee H)$ becomes $F \vee G \vee H$
- Replace $\wedge$ by comma and get a set of clauses

# Skolemization

- Outermost $\exists$:
  - $\exists x\ F[x]$ becomes $F[a]$ (all occurrences of $x$ replaced by $a$)
    $a$ is a new Skolem constant
  - There exists an element such that $F$: let this element be named $a$
- $\exists$ in the scope of $\forall$:
  - $\forall y \exists x\ F[x, y]$ becomes $\forall y\ F[g(y), y]$
    (all occurrences of $x$ replaced by $g(y)$)
    $g$ is a new Skolem function
  - For all $y$ there is an $x$ such that $F$: $x$ depends on $y$;
    let $g$ be the map of this dependence

# A simple example

- $\neg\{[\forall x\ P(x)] \supset [\exists y\ \forall z\ Q(y,z)]\}$
- $\neg\{\neg[\forall x\ P(x)] \vee [\exists y\ \forall z\ Q(y,z)]\}$
- $[\forall x\ P(x)] \wedge \neg[\exists y\ \forall z\ Q(y,z)]$
- $[\forall x\ P(x)] \wedge [\forall y\ \exists z\ \neg Q(y,z)]$
- $[\forall x\ P(x)] \wedge [\forall y\ \neg Q(y,f(y))]$ where $f$ is a Skolem function
- $\{P(x),\ \neg Q(y,f(y))\}$: a set of two unit clauses

# Clausal form and Skolemization

- All steps in the transformation into clauses except Skolemization preserve logical equivalence (for every interpretation, $F$ is true iff $F'$ is true)
- Skolemization only preserves equisatisfiability ($F$ is (un)satisfiable iff $F'$ is (un)satisfiable)
- Why Skolem symbols must be new? So that we can interpret them as in the model of $F$ when building a model of $F'$

# Herbrand interpretations

- First-order interpretation $\mathcal{M} = \langle \mathcal{D}, \Phi \rangle$
- Let $\mathcal{D}$ be the Herbrand universe $\mathcal{U}$
- Let $\Phi$ interpret constant and function symbols as themselves:
  - $\Phi(a) = a$
  - $\Phi(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$
- Predicate symbols? All possibilities
- The powerset $\mathcal{P}(\mathcal{B})$ gives all possible Herbrand interpretations
- Herbrand model: a satisfying Herbrand interpretation

## Clausal form and Herbrand interpretations

- Theorem-proving problem: is $H \cup \{\neg\varphi\}$ unsatisfiable?
- Transform $H \cup \{\neg\varphi\}$ into set $S$ of clauses ($S = T \uplus SOS$ where $SOS$ contains the clauses from $\neg\varphi$)
- $H \cup \{\neg\varphi\}$ and $S$ are equisatisfiable
- Theorem-proving problem: is $S$ unsatisfiable?
- $S$ is unsatisfiable iff $S$ has no Herbrand model
- From now on: only Herbrand interpretations

## Not for formulas

- $\exists x\ P(x) \wedge \neg P(a)$
- Is it satisfiable? Yes
- Herbrand model? No!
- $\emptyset$ and $\{P(a)\}$ or $\{\neg P(a)\}$ and $\{P(a)\}$
- Clausal form: $\{P(b),\ \neg P(a)\}$
- Herbrand model: $\{P(b)\}$ or $\{P(b),\ \neg P(a)\}$

# Satisfaction

- $\mathcal{M}$: Herbrand interpretation
- $\mathcal{M} \models S$ if $\mathcal{M} \models C$ for all $C \in S$
- $\mathcal{M} \models C$ if $\mathcal{M} \models C\sigma$ for all ground instances $C\sigma$ of $C$
- $\mathcal{M} \models C\sigma$ if $\mathcal{M} \models L\sigma$ for some ground literal $L\sigma$ in $C\sigma$

# Herbrand theorem

- $S$: set of clauses
- $S$ is unsatisfiable iff there exists a finite set $S'$ of ground instances of clauses in $S$ such that $S'$ is unsatisfiable
- Finite sets of ground instances can be enumerated and tested for propositional satisfiability which is decidable: the first-order theorem-proving problem is semi-decidable

# Instance-based strategies: basic idea

- Generate finite set of ground instances
- Test for satisfiability by SAT-solver
- Unsatisfiable: done
- Satisfiable with propositional model $\mathcal{M}$: generate ground instances false in $\mathcal{M}$ and repeat
- Model-driven instance generation

# Equality

- Congruence axioms in clausal form:
  - $x \simeq x$
  - $x \not\simeq y \vee y \simeq x$
  - $x \not\simeq y \vee y \not\simeq z \vee x \simeq z$
  - $x \not\simeq y \vee f(\ldots, x, \ldots) \simeq f(\ldots, y, \ldots)$
  - $x \not\simeq y \vee \neg P(\ldots, x, \ldots) \vee P(\ldots, y, \ldots)$
- $E$-satisfiability, $E$-interpretations, Herbrand $E$-interpretations

# Herbrand theorem

- $S$: set of clauses
- $S$ is $E$-unsatisfiable iff there exists a finite set $S'$ of ground instances of clauses in $S$ such that $S'$ is $E$-unsatisfiable

# Summary of the second part

- First-order theorem-proving problem
- Clauses and Herbrand interpretations
- Herbrand theorem
- Theorem proving in first-order logic is semi-decidable
- Design theorem-proving strategies that are semi-decision procedures and implement the Herbrand theorem
- Instance-based strategies aim at implementing directly the Herbrand theorem by emphasizing instance generation

# Expansion and contraction

Like many search procedures, most reasoning methods combine various forms of growing and shrinking:

- Ordering-based strategies: expansion and contraction of a set of clauses
- Ordering $\succ$ on clauses extended to sets of clauses by multiset extension

# Expansion

An inference

$$\frac{A}{B}$$

where $A$ and $B$ are sets of clauses is an expansion inference if

- $A \subset B$: something is added
- Hence $A \prec B$
- $(B \setminus A) \subseteq Th(A)$ hence $B \subseteq Th(A)$ hence $Th(B) \subseteq Th(A)$ (soundness)

# Contraction

An inference

$$\frac{A}{B}$$

where $A$ and $B$ are sets of clauses is a contraction inference if

- $A \not\subseteq B$: something is deleted or replaced
- $B \prec A$: if replaced, replaced by something smaller
- $(A \setminus B) \subseteq Th(B)$ hence $A \subseteq Th(B)$ hence $Th(A) \subseteq Th(B)$ (monotonicity or adequacy)
- Every step sound and adequate: $Th(A) = Th(B)$

# Propositional resolution

$$\frac{P \vee \neg Q \vee \neg R, \;\; \neg P \vee O}{O \vee \neg Q \vee \neg R}$$

where $O$, $P$, $Q$, and $R$ are propositional atoms
(aka propositional variables, aka 0-ary predicates)

# Propositional resolution

is an expansion inference rule:

$$\frac{S \cup \{L \vee C,\ \neg L \vee D\}}{S \cup \{L \vee C,\ \neg L \vee D,\ C \vee D\}}$$

- $S$ is a set of clauses
- $L$ is an atom
- $C$ and $D$ are disjunctions of literals
- $L$ and $\neg L$ are the literals resolved upon
- $C \vee D$ is called resolvent

# First-order resolution

$$\frac{S \cup \{L_1 \vee C,\ \neg L_2 \vee D\}}{S \cup \{L_1 \vee C,\ \neg L_2 \vee D,\ (C \vee D)\sigma\}}$$

where $L_1\sigma = L_2\sigma$ for $\sigma$ mgu

# First-order resolution

$$\frac{P(g(z), g(y)) \vee \neg R(z, y), \;\; \neg P(x, g(a)) \vee Q(x, g(x))}{\neg R(z, a) \vee Q(g(z), g(g(z)))}$$

where $\sigma = \{x \leftarrow g(z), \;\; y \leftarrow a\}$

# Ordered resolution

$$\frac{S \cup \{L_1 \vee C,\ \neg L_2 \vee D\}}{S \cup \{L_1 \vee C,\ \neg L_2 \vee D,\ (C \vee D)\sigma\}}$$

where

- $L_1\sigma = L_2\sigma$ for $\sigma$ mgu
- $L_1\sigma \not\preceq M\sigma$ for all $M \in C$
- $\neg L_2\sigma \not\preceq M\sigma$ for all $M \in D$

# Ordered resolution

$$\frac{P(g(z),g(y)) \vee \neg R(z,y),\ \ \neg P(x,g(a)) \vee Q(x,g(x))}{\neg R(z,a) \vee Q(g(z),g(g(z)))}$$

- $\sigma = \{x \leftarrow g(z),\ y \leftarrow a\}$
- $P(g(z),g(a)) \not\preceq \neg R(z,a)$
- $\neg P(g(z),g(a)) \not\preceq Q(g(z),g(g(z)))$
- Allowed, e.g., with $P > R > Q > g$
- Not allowed, e.g., with $Q > R > P > g > a$

## Subsumption

$$\frac{S \cup \{P(x,y) \vee Q(z),\ Q(a) \vee P(b,b) \vee R(u)\}}{S \cup \{P(x,y) \vee Q(z)\}}$$

$C = P(x,y) \vee Q(z)$ subsumes $D = Q(a) \vee P(b,b) \vee R(u)$
as there is a substitution $\sigma = \{z \leftarrow a, x \leftarrow b, y \leftarrow b\}$
such that $C\sigma \subset D$ hence $\{C\} \models \{D\}$ (adequacy)

# Subsumption ordering

- Subsumption ordering: $C \stackrel{\bullet}{\leq} D$ if $\exists\sigma\ C\sigma \subseteq D$ (as multisets)
- Strict subsumption ordering: $C \stackrel{\bullet}{<} D$ if $C \stackrel{\bullet}{\leq} D$ and $C \stackrel{\bullet}{\not\geq} D$
- The strict subsumption ordering $\stackrel{\bullet}{<}$ is well-founded
- Equality up to variable renaming: $C \stackrel{\bullet}{=} D$ if $C \stackrel{\bullet}{\leq} D$ and $C \stackrel{\bullet}{\geq} D$ ($C$ and $D$ are variants)

## Subsumption

is a contraction inference rule:

$$\frac{S \cup \{C,\ D\}}{S \cup \{C\}}$$

- Either $C \lessdot D$ (strict subsumption)
- Or $C \doteq D$ and $C \prec D$ where $\prec$ is the lexicographic combination of $\lessdot$ and another well-founded ordering (e.g., $C$ was generated before $D$) (subsumption of variants)
- Clause $D$ is redundant
- Subsumption uses matching, resolution uses unification

# And equality?

Replacing equals by equals as in ground rewriting:

$$\frac{S \cup \{f(a,a) \simeq a,\ P(f(a,a)) \vee Q(a)\}}{S \cup \{f(a,a) \simeq a,\ P(a) \vee Q(a)\}}$$

It can be done as $f(a,a) \succ a$ (by the subterm property)

# Simplification

is a contraction inference rule:

$$\frac{S \cup \{f(x,x) \simeq x,\ P(f(a,a)) \vee Q(a)\}}{S \cup \{f(x,x) \simeq x,\ P(a) \vee Q(a)\}}$$

- $f(x,x)$ matches $f(a,a)$ with $\sigma = \{x \leftarrow a\}$
- $f(a,a) \succ a$

# Simplification

$$\frac{S \cup \{s \simeq t,\ L[r] \vee C\}}{S \cup \{s \simeq t,\ L[t\sigma] \vee C\}}$$

- $L$ is a literal with $r$ as subterm ($L$ could be another equation)
- $C$ is a disjunction of literals
- $\exists\sigma$ such that $s\sigma = r$ and $s\sigma \succ t\sigma$
- $L[t\sigma] \vee C$ is entailed by the original set (soundness)
- $L[r] \vee C$ is entailed by the resulting set (adequacy)
- $L[r] \vee C$ is redundant

# Expansion for equality reasoning

- Simplification is a powerful rule that often does most of the work in presence of equality
- But it is not enough
- Equality reasoning requires to generate new equations
- We need an expansion rule that builds equality into resolution and uses unification not only matching

# Superposition/Paramodulation

$$\frac{f(z,e) \simeq z, \;\; f(I(x,y),y) \simeq x}{I(x,e) \simeq x}$$

- $f(z,e)\sigma = f(I(x,y),y)\sigma$
- $\sigma = \{z \leftarrow I(x,e), \;\; y \leftarrow e\}$ most general unifier
- $f(I(x,e),e) \succ I(x,e)$ (by the subterm property)
- $f(I(x,e),e) \succ x$ (by the subterm property)
- Superposing two equations yields a peak:
  $I(x,e) \leftarrow f(I(x,e),e) \rightarrow x$

# Superposition/Paramodulation

is an expansion inference rule:

$$\frac{S \cup \{l \simeq r,\ p[s] \bowtie q\}}{S \cup \{l \simeq r,\ p[s] \bowtie q,\ (p[r] \bowtie q)\sigma\}}$$

- $\bowtie$ is either $\simeq$ or $\not\simeq$
- $s$ is not a variable
- $l\sigma = s\sigma$ with $\sigma$ mgu
- $l\sigma \not\preceq r\sigma$ and $p\sigma \not\preceq q\sigma$

# Completion

- New equations closing such peaks are called critical pairs, as they complete the set of equations into a confluent one
- Confluence ensures uniqueness of normal forms
- This procedure is known as Knuth-Bendix completion
- Unfailing or Ordered Knuth-Bendix completion ensures ground confluence (unique normal form of ground terms) which suffices for theorem proving in equational theories as the Skolemized form of $\neg(\forall \bar{x}\ s \simeq t)$ is ground

# Superposition/Paramodulation

$$\frac{S \cup \{l \simeq r \vee C,\ L[s] \vee D\}}{S \cup \{l \simeq r \vee C,\ L[s] \vee D,\ (L[r] \vee C \vee D)\sigma\}}$$

- $C$ and $D$ are disjunctions of literals
- $L[s]$: literal paramodulated into
- $s$ is not a variable
- $l\sigma = s\sigma$ with $\sigma$ mgu
- $l\sigma \not\preceq r\sigma$ and if $L[s]$ is $p[s] \bowtie q$ then $p\sigma \not\preceq q\sigma$
- $(l \simeq r)\sigma \not\prec M\sigma$ for all $M \in C$
- $L[s]\sigma \not\preceq M\sigma$ for all $M \in D$

## What's in a name

- Paramodulation was used first in resolution-based theorem proving where simplification was called demodulation
- Superposition and simplification, or rewriting, were used first in Knuth-Bendix completion
- Some authors use superposition between unit equations and paramodulation otherwise
- Other authors use superposition when the literal paramodulated into is an equational literal and paramodulation otherwise

# Derivation

- Input set $S$
- Inference system $\mathcal{I}$: a set of inference rules
- $\mathcal{I}$-derivation from $S$:

$$S_0 \vdash_{\mathcal{I}} S_1 \vdash_{\mathcal{I}} \ldots S_i \vdash_{\mathcal{I}} S_{i+1} \vdash_{\mathcal{I}} \ldots$$

  where $S_0 = S$ and for all $i$, $S_{i+1}$ is derived from $S_i$ by an inference rule in $\mathcal{I}$
- Refutation: a derivation such that $\Box \in S_k$ for some $k$

# Refutational completeness

An inference system $\mathcal{I}$ is refutationally complete if for all sets $S$ of clauses, if $S$ is unsatisfiable, there exists an $\mathcal{I}$-derivation from $S$ that is a refutation.

# Ordering-based inference system

An inference system with

- Expansion rules: resolution, factoring, superposition/paramodulation, equational factoring, reflection (resolution with $x \simeq x$)
- Contraction rules: subsumption, simplification, tautology deletion, clausal simplification (unit resolution + subsumption)

is refutationally complete

# Summary of the third part

- Expansion and contraction
- Resolution and subsumption
- Paramodulation/superposition and simplification
- Contraction uses matching, expansion uses unification
- Ordering-based inference system
- Derivation
- Refutational completeness

# Search

- An inference system is non-deterministic
- Given $S$ and $\mathcal{I}$, many $\mathcal{I}$-derivations from $S$ are possible
- Which one to build? Search problem
- Search space
- Rules and moves: inference rules and inference steps

# Strategy

- Theorem-proving strategy: $\mathcal{C} = \langle \mathcal{I}, \Sigma \rangle$
- $\mathcal{I}$: inference system
- $\Sigma$: search plan
- The search plan picks at every stage of the derivation which inference to do next
- A deterministic proof procedure

# Completeness

- Inference system: refutational completeness
  there exist refutations
- Search plan: fairness
  ensure that the generated derivation is a refutation
- Refutationally complete inference system + fair search plan = complete theorem-proving strategy

# Fairness

- Fairness: consider eventually all needed steps: What is needed?
- Dually: what is not needed, or: what is redundant?
- Fairness and redundancy are related

# Redundancy

- Based on ordering $\succ$ on clauses:
  a clause is redundant if all its ground instances are;
  a ground clause is redundant if there are ground instances of other clauses that entail it and are smaller
- Based on ordering $\succ$ on proofs:
  a clause is redundant if adding it does not decrease any minimal proofs (dually, removing it does not increase proofs)
- Agree if proofs are measured by maximal premises
- Redundant inference: uses/generates redundant clause

# Fairness

- A derivation is fair if whenever a minimal proof of the target theorem is reducible by inferences, it is reduced eventually
- A derivation is uniformly fair if all non-redundant inferences are done eventually
- A search plan is (uniformly) fair if all its derivations are

## Contraction first

- Eager-contraction search plan:
- Schedule contraction before expansion

# The given-clause algorithm

- Two lists: *ToBeSelected* and *AlreadySelected*
  (Other names: *SOS* and *Usable*; *Active* and *Passive*)
- Initialization:
  - $\mathit{ToBeSelected} = S_0$ (the input clauses)
  - $\mathit{AlreadySelected} = \emptyset$
- Alternative: the set of support strategy
  - $\mathit{ToBeSelected} = \mathit{clauses}(\neg\varphi)$ (clauses from the goal)
  - $\mathit{AlreadySelected} = \mathit{clauses}(H)$ (the other input clauses)

# The given-clause algorithm: expansion

- Loop until either proof found or $\mathit{ToBeSelected} = \emptyset$, the latter meaning satisfiable
- At every iteration: pick a given-clause from *ToBeSelected*
- How? Best-first search: the best according to an evaluation function (e.g., weight, FIFO, pick-given ratio)
- Perform all expansion steps with the given-clause and clauses in *AlreadySelected* as premises
- Move the given-clause from *ToBeSelected* to *AlreadySelected*
- Insert all newly generated clauses in *ToBeSelected*

# Forward contraction

- Forward contraction: contract newly generated clauses by pre-existing ones
- Forward contract each new clause prior to insertion in *ToBeSelected*
- A very high number of clauses gets deleted typically by forward contraction

# Backward contraction

- Backward contraction: contract pre-existing clauses by new ones
- For fairness backward contraction must be applied after forward contraction (e.g., subsumption)
- Detect which clauses can be backward-contracted and treat them as new
- Every backward-contracted clause may backward-contract others
- How much to do? How often?

# A choice of invariants

- Keep *ToBeSelected* $\cup$ *AlreadySelected* contracted
- Keep only *AlreadySelected* contracted
  - Backward-contract $\{$*given–clause*$\}$ $\cup$ *AlreadySelected* right after picking the given-clause
  - Deletion of "orphans" in *ToBeSelected*

# Proof reconstruction

- The derivation is not the proof
- At the end of a successful derivation:
  - Proof reconstruction
  - The ancestor-graph of $\Box$

# Theorem provers

- Proof assistant $\sim$ interpreter
- Theorem prover $\sim$ compiler
  - Iterative experimentation with settings (options, parameters)
  - Incomplete strategies
  - Auto mode
  - Machine learning of settings

# Some theorem provers

- Otter, EQP, and Prover9 by the late Bill McCune
- SNARK by the late Mark E. Stickel
- SPASS by Christoph Weidenbach et al.
- E by Stephan Schulz and EHOH by Petar Vukmirovic
- Vampire by Andrei Voronkov et al.
- Waldmeister by Thomas Hillenbrand et al.
- leanCoP by Jens Otten
- iProver by Konstantin Korovin et al.
- Metis by Joe Leslie-Hurd and MetiTarski by Larry Paulson et al.
- Zipperposition by Simon Cruanes

## Some applications

- Analysis, verification, synthesis of systems, e.g.:
  - Cryptographic protocols
  - Message-passing systems
  - Software specifications
  - Theorem-proving support to model checking
- Mathematics: proving non-trivial theorems in, e.g.,
  - Boolean algebras (e.g., the Robbins conjecture)
  - Theories of rings (e.g., the Moufang identities), groups and quasigroups
  - Many-valued logics (e.g., Lukasiewicz logic)

# Some research topics

- Strategies seeking proof/counter-model in one search: model-based first-order reasoning
- Adding built-in theories
- Integration of theorem-provers and SAT/SMT solvers
- Theorem-proving strategies as decision procedures
- Parallel/distributed theorem proving
- Goal-sensitive or target-oriented strategies
- Machine-independent evaluation of strategies: strategy analysis, search complexity

# Some textbooks


- Chin-Liang Chang, Richard Char-Tung Lee. Symbolic Logic and Mechanical Theorem Proving. Computer Science Classics, Academic Press, 1973
- Alexander Leitsch. The Resolution Calculus. Texts in Theoretical Computer Science, An EATCS Series, Springer, 1997
- Rolf Socher-Ambrosius, Patricia Johann. Deduction Systems. Graduate Texts in Computer Science, Springer, 1997
- John Harrison. Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, 2009

# More textbooks


- Raymond M. Smullyan. First-order logic. Dover Publications 1995 (republication of the original published by Springer Verlag in 1968)
- Allan Ramsay. Formal Methods in Artificial Intelligence. Cambridge Tracts in Theoretical Computer Science 6, Cambridge University Press, 1989
- Ricardo Caferra, Alexander Leitsch, Nicolas Peltier. Automated Model Building. Applied Logic Series 31, Kluwer Academic Publishers, 2004
- Martin Davis. The Universal Computer. The Road from Leibniz to Turing. Turing Centenary Edition. Mathematics/Logic/Computing Series. CRC Press, Taylor and Francis Group, 2012

## Some surveys


- Maria Paola Bonacina. A taxonomy of theorem-proving strategies. In *Artificial Intelligence Today – Recent Trends and Developments*, LNAI 1600:43–84, Springer, 1999 [providing 150 references]
- Maria Paola Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence* 29(1/4):223–257, 2000 [providing 104 references]
- Maria Paola Bonacina. On theorem proving for program checking – Historical perspective and recent developments. In *Proc. of the 12th Int. Symp. on Principles and Practice of Declarative Programming*, 1–11, ACM Press, 2010 [providing 119 references]

# More surveys


- Maria Paola Bonacina, Ulrich Furbach, Viorica Sofronie-Stokkermans. On first-order model-based reasoning. In *Logic, Rewriting, and Concurrency*, LNCS 9200:181–204, Springer, 2015 [providing 88 references]
- Maria Paola Bonacina. On conflict-driven reasoning. In *Proc. of the 6th Workshop on Automated Formal Methods (May 2017)*, Kalpa Publications, 5:31–49, EasyChair, 2018 [providing 60 references]
- Maria Paola Bonacina. Parallel theorem proving. In *Handbook of Parallel Constraint Reasoning*, Ch. 6, 179–235, Springer, 2018 [providing 230 references]

# Thanks

**Thank you!**