Journal of Automated Reasoning manuscript No.
DOI 10.1007/s10817-022-09628-0 in 66(4):463–497, November 2022

Set of Support, Demodulation, Paramodulation: A Historical Perspective

Maria Paola Bonacina

Received: 31 March 2021 / Accepted: 2 April 2022 / Published online: 24 May 2022

Abstract This article is a tribute to the scientific legacy of automated reasoning pioneer and JAR founder Lawrence T. (Larry) Wos. Larry's main technical contributions were the *set-of-support strategy* for resolution theorem proving, and the *demodulation* and *paramodulation* inference rules for building equality into resolution. Starting from the original definitions of these concepts in Larry's papers, this survey traces their evolution, unearthing the often forgotten trails that connect Larry's original definitions to those that became standard in the field.

Keywords Resolution · Set of support · Demodulation · Paramodulation

Prologue: Larry Wos, A Recollection

My first encounter with Larry Wos dates to when I was a PhD student at the State University of New York at Stony Brook and Larry was the head of the theorem-proving group at the Argonne National Laboratory. I had submitted to CADE (the international Conference on Automated DEduction) a paper, co-authored with Siva Anantharaman of the Université d'Orléans, about proving mechanically with the theorem prover SBR3 a theorem in *Łukasiewicz many-valued logic* [2].

Łukasiewicz had conjectured that a set of five axioms, together with modus ponens, constitutes an axiomatization of the many-valued logic [171] that would be later called after him. The conjecture had been proved first by Wajsberg, and then independently by Rose and Rosser [153] and by Chang [62, 64]. Then Meredith [129] and Chang [63] also independently had derived the fifth axiom from the other four. The latter problem had been brough to my attention by Daniele Mundici of the Università degli Studi di Milano, as a

Università degli Studi di Verona, Strada Le Grazie 15, 37134 Verona, Italy, EU. E-mail: mariapaola.bonacina@univr.it ORCID 0000-0001-9104-2692

challenge for automated theorem provers. My CADE submission with Siva presented a mechanical proof of the *dependency of the fifth axiom* in an equivalent equational formulation in *Wajsberg algebras*, a class of algebras connected to Łukasiewicz many-valued logic [85,152]. The submission was rejected, and a referee report said that the reason for rejection was that they had given the problem to OTTER, the theorem prover developed at Argonne, and OTTER also had succeeded. Siva and I continued the investigation of many-valued logic and Wajsberg algebras, proving other problems with SBR3 [3].

Several months after the CADE rejection, I read in the Newsletter of the Association for Automated Reasoning (AAR) an article by Larry Wos [182], presenting the dependency of the fifth axiom, in both formulations, as a challenge for theorem provers, and citing my unpublished work on the topic. After consulting with my advisor Jieh Hsiang, I wrote an e-mail to Larry. To my great surprise, Larry called me on the phone, and we talked for quite a while. Larry encouraged me to send to the AAR Newsletter my own presentation of both original and equational formulations of the problem, which appeared several months later [41]. Łukasiewicz many-valued logic became a source of challenge problems for theorem provers at Argonne. Larry and I became friends with that phone call and remained friends ever since.

When Bill McCune served on my PhD defense committee, he invited me to visit Argonne during the interval between my departure from Stony Brook and the start of a postdoc at INRIA Lorraine. During that visit I worked with Bill, but I often had the chance to go for lunch or otherwise meet informally with Larry. Larry talked fondly and often of his wife Nancy, who, like him, could not see. Larry loved to make fun of people around him, but would never make fun of Bill. I thought that Larry had so much respect for Bill, as the author of OTTER, that he would not tease him. It is also possible that Larry knew that he would not get much satisfaction out of Bill's reserved personality. At times I was a bit worried that Larry would tease me, but that never happened. Perhaps Larry guessed that I would have neither reacted nor let my feelings show, and hence there was not much fun in trying, or else he respected me for my conduct in relation to the events around the Łukasiewicz problem.

A few years later, when I was on the faculty of the University of Iowa, Larry invited me to succeed Bob Veroff as Secretary of the AAR, a job that I was glad to accept. The following summer I returned to Argonne to discuss research with Bill, but unfortunately I did not get to meet with Larry. I was delighted when Larry resumed calling me on the phone to discuss AAR matters, and all the more so because that was after I had moved to the Università degli Studi di Verona: a different time zone was obviously no deterrent for Larry when he wanted to talk on the phone. During one of these conversations he once told me: "Don't you ever leave CADE and the AAR: we can't do it without you." I knew it could not be true, but it is an eloquent example of Larry's style in letting people know how much they were appreciated.

My last interaction with Larry was when the late Mark Stickel and I took the initiative of editing a volume in memory of Bill McCune: Mark and I were thrilled that Larry contributed the opening chapter in the volume [183]. While

Larry's preference for the telephone over e-mail was most likely due to the fact that he could not see, I think that it was also a kind of wisdom, as e-mail lacks the information that the human voice can convey.

1 Introduction

Larry Wos broke new ground in the design of both fundamental components of a *theorem-proving strategy* or *proof procedure*, namely the *inference system* and the *search plan*. His seminal contributions include the *set-of-support strategy* for resolution-based theorem proving [187], and the *demodulation* [188] and *paramodulation* [147] inference rules for equational reasoning. Each of them had a profound impact on theorem proving.

The idea of set of support was a primer in controlling resolution [149] with semantic knowledge (i.e., that a part of the problem is satisfiable), opening the way to *semantic* or *semantically-guided*, *supported*, and *goal-sensitive* strategies (e.g., [161,140,162,17,142,59,60]). The set-of-support strategy is also at the origin of the *given-clause algorithm* implemented first in OTTER (e.g., [128, 127]) and then in many theorem provers (e.g., [157,96,146,180,110,66,159, 170,175]).

With demodulation, Larry Wos posed the problem of *well-founded replacement of equals by equals in theorem proving*. With the concept of paramodulation, he challenged the field with the *Wos–Robinson conjecture* on the refutational completeness of paramodulation *without paramodulating into variables and without functionally reflexive axioms*. The successful solution of these problems involved decades of research, leading to a merger of resolution-based and completion-based [107,101,8,98,9,6,7,54] theorem proving that shaped the field of theorem proving for first-order logic with equality. The resulting inference systems combine resolution, paramodulation, superposition, well-founded demodulation, and subsumption (e.g., [133,154,99,155,10]). These inference systems have been called *completion-based*, *rewrite-based*, *saturation-based*, or *ordering-based* [45], given the key role played by well-founded orderings on terms, literals, and clauses.

Larry was interested mainly in devising inference rules and search plans, and refining them through experiments with the Argonne provers. Accordingly, this survey focuses on the history of inference rules and search plans, and covers neither that of fundamental concepts in theorem proving, such as *completeness*, *fairness*, *saturation*, *redundancy*, and *canonicity*, nor that of completeness proof techniques, such as *semantic trees*, *proof orderings*, *transfinite semantic trees*, and *rewrite models*.

Furthermore, this survey is about Larry Wos' contributions, and hence it considers resolution-based theorem proving and its extensions to equality reasoning leading to the above mentioned *ordering-based inference systems*. There are several other fundamental principles for theorem proving in first-order logic, including *model elimination* [118,119], *matings* [4] or *connections* [33], all three formalized as *tableaux-based strategies* (e.g., [18,114,

93,116]), *instance-based* methods (e.g., [112,38,88,15]), *tableaux with instance generation* (e.g., [14,16,20]), *model evolution* [22,21], SGGS (*Semantically-Guided Goal-Sensitive reasoning*) which is model-based, instance-based, and *conflict-driven* [59,60], and the list is certainly incomplete.

The interested reader can find complementary material in books (e.g., [65, 120,34,35,113,141,163]), surveys about theorem proving in general (e.g., [136, 37,45,117,139,48]), surveys about resolution, rewriting, and equational reasoning (e.g., [76,137,83,11,12,78,130]), and surveys of tableaux-based strategies (e.g., [18,114,93,116]), instance-based strategies (e.g., [103,108]), model-based methods [51], and conflict-driven methods [47]. This article has historic contents, but given its focus on one scientist–Larry Wos–it cannot be a well-rounded account of the early history of theorem proving. Sources dedicated to the early history of the field (e.g., [68,185,36]) and to its roots in the general history of mathematics and computer science (e.g., [69]) are available.

This article is organized as follows. Section 2 introduces the *theorem-proving problem*, reconstructing the state of the art prior to Larry Wos' work, and outlining some of his ideas about the properties that inference rules, search plans, and proofs ought to have. Sections 3, 4, and 5 are devoted to the *set-of-support strategy*, *demodulation*, and *paramodulation*, respectively. For each of them, the main phases of their evolution from Larry's time to their standardization are outlined, showing the impact of Larry's ideas. Section 6 discusses a selection of subsequent research directions whose origins can be connected to Larry's work.

2 Setting the Stage: Resolution-based Theorem Proving

The *theorem-proving problem* is about determining whether a formula φ is a logical consequence of a set of formulae H, written $H \models^? \varphi$, or, equivalently, whether the formula $H \supset \varphi$ is valid, written $\models^? H \supset \varphi$. Mechanical theorem proving approaches this problem *refutationally*, by trying to determine whether $H \cup \{\neg\varphi\}$ is unsatisfiable, and in *clausal form*, by turning $H \cup \{\neg\varphi\}$ into an equisatisfiable set S of *clauses*. A clause is a disjunction of literals with variables implicitly universally quantified. A set of clauses is understood logically as their conjunction, where each clause has its own variables. A clause is a *unit clause*, if it contains exactly one literal; it is a *positive clause*, if all its literals are positive; it is a *negative clause*, if all its literals are negative; it is a *Horn clause*, if it contains at most one positive literal. An *ordering* is a binary relation that is irreflexive and transitive. A *quasi-ordering* is a binary relation that is reflexive and transitive. An ordering is *well-founded*, if it admits no infinite descending chain.

2.1 Expansion and Contraction Inference Rules

In the context of refutational clausal theorem proving, an inference system Γ is a collection of *inference rules* that transform sets of clauses. In resolution-based

theorem proving the most important such rule is the *binary resolution* inference rule, that combines *unification* with resolving upon literals of opposite sign [149]. According to [36], the idea of propositional resolution appeared as early as 1937 [39], was rediscovered in 1955 [132], and applied to theorem proving in the *Davis–Putnam procedure* [70] as well as in another procedure [82]. The basic idea of unification already appeared in the work of Herbrand [95] and Prawitz [143]. Nonetheless, it was Alan Robinson who understood how to merge these two ideas in the resolution principle for first-order theorem proving [149]. Binary resolution generates from two clauses, termed *parents*, a new clause, termed *binary resolvent*, and adds it to the set:

$$\textit{Binary Resolution}: \quad \frac{S \cup \{L_1 \vee C,\ L_2 \vee D\}}{S \cup \{L_1 \vee C,\ L_2 \vee D,\ (C \vee D)\sigma\}} \quad L_1\sigma = \neg L_2\sigma,$$

where L_1 and L_2 are the *literals resolved upon* that unify and have opposite sign, and C and D are disjunctions of literals. Here, and in the sequel, unifiers are most general unifiers, abbreviated mgu's. Binary resolution is accompanied by *factoring*, which generates from a clause, termed *parent*, a new clause, termed *factor*, and adds it to the set [149]:

$$\textit{Factoring}: \quad \frac{S \cup \{L_1 \vee \ldots \vee L_k \vee C\}}{S \cup \{L_1 \vee \ldots \vee L_k \vee C,\ (L_1 \vee C)\sigma\}} \quad L_1\sigma = L_2\sigma = \ldots = L_k\sigma.$$

In the original presentation [149], binary resolution and factoring were integrated in the *resolution* inference rule, so that a *resolvent* of two clauses C and D is either a binary resolvent of C and D, or a binary resolvent of a factor of C and D, or a binary resolvent of C and a factor of D, or a binary resolvent of a factor of C and a factor of D [65]. Thus, factoring only needs to be applied to resolution parents. Larry Wos propounded considering binary resolution and factoring as distinct inference rules (e.g., [187]), so that binary resolution can be called simply resolution, as we will do from now on.

If a parent is a unit clause, resolution is *unit resolution*, also a feature of the Davis–Putnam procedure [70]. Unit resolution is advantageous, because the resolvent is one literal shorter than the non-unit parent. In general, the resolvent inherits all the literals of its parents, except the two literals resolved upon, so that inferred clauses grow longer and longer, and hence more expensive to process. This well-known disadvantage of resolution was later studied as *duplication by combination* [138,141].

Resolution and factoring are *expansion inference rules*, as they fit in the *expansion inference scheme*:

$$\textit{Expansion}: \quad \frac{S}{S'} \quad S \subset S',$$

where $S \subset S'$ says that the existing set S of clauses is expanded by adding some clause. Symmetrically, *contraction inference rules* are rules that *contract*

the set of clauses, because they delete clauses or replace them by smaller ones according to the *contraction inference scheme*:

$$\textit{Contraction}: \quad \frac{S}{S'} \quad S \not\subseteq S',\ S' \prec_{mul} S,$$

where $S \not\subseteq S'$ tells that something has been deleted, and $S' \prec_{mul} S$ says that S' is smaller than S in the *multiset extension* [77] of a *well-founded* ordering $\prec$ on clauses. The multiset extension of a well-founded ordering is also well-founded [77]. The double inference line [49] emphasizes the diversity of contraction with respect to the traditional notion of inference in logic (e.g., natural deduction). Contraction rules that only delete clauses are also called *deletion rules*, whereas contraction rules that delete clauses and replace them by smaller ones are also called *replacement rules*. Expansion rules and replacement rules together are called *generative inference rules* [57], because they are those that generate clauses.

Assuming $Th(S) = \{C : S \models C\}$, a generative inference rule is *sound*, if $S' \subseteq Th(S)$: whatever is added is a logical consequence of what pre-existed. A contraction rule is *adequate* [49], if $S \subseteq Th(S')$: whatever is deleted is a logical consequence of what remains. Adequacy implies *monotonicity* [53]: a contraction rule is *monotonic* if $Th(S) \subseteq Th(S')$. Soundness and adequacy together imply $Th(S) = Th(S')$. An inference system Γ is *sound* and *adequate* if its rules are.

The contraction inference rules for resolution-based theorem proving that were known prior to Larry Wos' contributions are *tautology deletion*, *purity deletion*, and *subsumption*, all three deletion rules. Tautology deletion appeared in the Davis–Putnam procedure [70]:

$$\textit{Tautology Deletion}: \quad \frac{S \cup \{L \vee \neg L \vee C\}}{S}$$

Purity deletion appeared first for propositional logic in the Davis–Putnam procedure [70], and, according to [36], in the procedure in [81]. It was generalized to first-order logic by integrating it with unification [149]:

$$\textit{Purity Deletion}: \quad \frac{S \cup \{L_1 \vee C\}}{S} \quad \textit{if } L_1 \textit{ is pure in } S \cup \{L_1 \vee C\},$$

where literal L_1 is *pure* in $S \cup \{L_1 \vee C\}$ if S contains no clause $L_2 \vee D$ such that $L_1\sigma = \neg L_2\sigma$. *Subsumption* also appeared in [149]:

$$\textit{Subsumption}: \quad \frac{S \cup \{C,\ D\}}{S \cup \{C\}} \quad C\sigma \subseteq D,\ |C| \leq |D|,$$

where σ is a matching substitution, $\subseteq$ is the subset relation between clauses viewed as sets of literals, and $|C|$ is the *number of literals* in clause C. The original definition [149] did not require that $|C| \leq |D|$, because factoring was integrated into resolution. If resolution and factoring are treated as separate

inference rules, either the $|C| \leq |D|$ condition must be added, or clauses must be treated as *multisets of literals*, in order to prevent a clause from subsuming its factors:

$$\textit{Subsumption}: \quad \frac{S \cup \{C,\ D\}}{S \cup \{C\}} \quad C\sigma \subseteq D,$$

where $\subseteq$ is the subset relation between clauses viewed as multisets of literals. From now on, clauses are considered as multisets of literals. On the other hand, a factor can subsume its parent, and the combination of factoring and subsumption where a factor is generated and then it subsumes its parent, is known as *condensation* [105,127]. If C is a unit clause, subsumption is called *unit subsumption*.

Subsumption and unit resolution can be combined into a replacement rule named *Clausal Simplification* [155]:

$$\textit{Clausal Simplification}: \quad \frac{S \cup \{L_1,\ L_2 \vee C\}}{S \cup \{L_1,\ C\}} \quad L_1\sigma = \neg L_2,$$

where the resolvent C, produced by unit resolution of L_1 and $L_2 \vee C$, subsumes its non-unit parent $L_2 \vee C$, because σ is a matching substitution that does not instantiate the variables in the literals of C.

Contraction inference rules use matching, whereas expansion inference rules use unification. When contraction is applied to delete or simplify a newly generated clause with respect to previously existing clauses, it is called *forward contraction*. When contraction is applied to delete or simplify previously existing clauses by a newly generated clause, it is called *backward contraction*. Showing his appreciation of the importance of contraction, Larry Wos wrote that subsumption should have been considered even more important than resolution among Alan Robinson's contributions [186].

2.2 Derivations and Refutational Completeness

Given input set S of clauses and inference system Γ, a *derivation* by Γ, or Γ-*derivation*, is a sequence of the form $S_0 \vdash_\Gamma S_1 \vdash_\Gamma \ldots S_i \vdash_\Gamma S_{i+1} \vdash_\Gamma \ldots$, where $S_0 = S$, and $\forall i, i \geq 0$, S_{i+1} is derived by applying to S_i an inference rule of Γ. An inference system discovers that S is unsatisfiable, by showing that S is inconsistent, that is, by deriving from S a contradiction, represented in clausal form by the *empty clause*, denoted with $\Box$. A derivation with input S is a *refutation* of S, if there exists a k, $k \geq 0$, such that $\Box \in S_k$. An inference system Γ is *refutationally complete*, if for all unsatisfiable clause sets S given as input there exists at least a Γ-derivation that is a refutation of S. A derivation is characterized by the set $S^* = \bigcup_{i \geq 0} S_i$ of all input or generated clauses and the set $S_\infty = \bigcup_{j \geq 0} \bigcap_{i \geq j} S_i$ of all *persistent clauses*. The latter set is called the *limit* of the derivation.

The inference system with *resolution* and *factoring* as expansion inference rules, and *tautology deletion*, *purity deletion*, *subsumption*, and *clausal simplification* as contraction inference rules, is sound and adequate, and it is

refutationally complete [149], provided *forward subsumption is applied before backward subsumption* [111,120,53]. As remarked in [120], the reason for this proviso is that the *subsumption ordering* defined by $C \preceq D$ if $C\sigma \subseteq D$ is *not well-founded*. More precisely, $\preceq$ is not an ordering, it is a quasi-ordering, and the induced equivalence relation $C \doteq D$ if $C \preceq D$ and $D \preceq C$ admits equivalence classes of infinite cardinality. Thus, a derivation can generate an infinite series of equivalent clauses and an infinite series of subsumption steps that preempt the resolution steps leading to a contradiction. A solution is to restrict subsumption as follows (e.g., [155]):

$$\textit{Proper Subsumption}: \quad \frac{S \cup \{C,\ D\}}{S \cup \{C\}} \quad C \prec D,$$

where the *strict subsumption ordering* $\prec$, defined by $C \prec D$ if $C \preceq D$ and $D \not\preceq C$, is *well-founded*. However, this solution prevents the inference system from subsuming one out of two clauses C and D such that $C \doteq D$. Such clauses are *variants* or *equal up to variable renaming*: $C\sigma \subseteq D$ and $D\rho \subseteq C$ (as multisets) by substitutions σ and ρ that are *variable renamings*, meaning that they replace variables by variables, without replacing distinct variables by the same. Clauses are *similar* [60] if $C\sigma \subseteq D$ and $D\rho \subseteq C$ (as sets) by substitutions σ and ρ that replace variables by variables, possibly replacing distinct variables by the same. Similar clauses are not variants and are not in the $\doteq$ relation, but they have factors that are variants.

Example 1 Clauses $C = P(x) \vee P(y) \vee Q(y)$ and $D = P(w) \vee Q(w) \vee Q(v)$ are similar as they satisfy $C\sigma \subseteq D$ with $\sigma = \{x \leftarrow w, y \leftarrow w\}$ and $D\rho \subseteq C$ with $\rho = \{w \leftarrow y, v \leftarrow y\}$). Clauses $P(x) \vee Q(y)$ and $P(w) \vee Q(v)$ are variants.

Assume that all clauses in S^* are given distinct increasing natural numbers as identifiers, as it happens in implementations. For clauses C and D such that $C \doteq D$, the solution is to take the lexicographic combination of $\prec$ and the ordering on the natural numbers and apply it to pairs (C, n), where n is the identifier of clause C. This ordering is well-founded, because a lexicographic combination of well-founded orderings is well-founded. Given pairs (C, n) and (D, m) such that $C \doteq D$, clause C subsumes D if $n < m$, that is, if C was generated before D. Applying forward subsumption before backward subsumption implements this concept.

2.3 Search Plans, Fairness, Strategies, and Proof Reconstruction

Given a set S of clauses and an inference system Γ, there is in general more than one way to apply rules in Γ to S. This means that Γ is *nondeterministic*, so that the Γ-derivation with input S is not unique. In order to obtain a *deterministic* procedure, Γ is coupled with a *search plan* Σ that chooses at each stage of the derivation which rule to apply to which clauses. A search plan Σ is *fair* for an inference system Γ, if for all input clause sets S, if there exist refutations of S by Γ, the Γ-derivation driven by Σ is a refutation of S.

A *theorem-proving strategy*, or *proof procedure*, is a pair $\mathcal{P} = \langle \Gamma, \Sigma \rangle$, where Γ is an inference system and Σ is a search plan. Given input set S of clauses, $\mathcal{P}$ generates the *unique* Γ-derivation driven by Σ. A theorem-proving strategy $\mathcal{P} = \langle \Gamma, \Sigma \rangle$ is *complete*, if for all unsatisfiable input clause sets S the Γ-derivation generated by $\mathcal{P}$ is a refutation of S. Thus, if Γ is refutationally complete and Σ is fair, $\mathcal{P} = \langle \Gamma, \Sigma \rangle$ is complete, and it is a *semidecision procedure* for validity in first-order logic.

As both S^* and S_∞ contain many clauses that are unrelated to the generation of $\Box$, when $\Box \in S_k$, the strategy *reconstructs* the generated *proof* in the form of the *ancestor-graph* [56] of $\Box$, denoted $\Pi(\Box)$. The reconstruction starts from $\Box$ and proceeds backward until it reaches input clauses, following the applications of generative rules, whereas applications of deletion rules do not appear in proofs. Since a clause may be used as premise more than once, $\Pi(C)$ is a tree, if different nodes are allowed to have the same clause as label, it is a directed acyclic graph otherwise. The ancestor-graph $\Pi(C)$ is defined for all $C \in S^*$. If C is an input clause, $\Pi(C)$ has one node labeled by C itself. If C is generated by a generative rule from premises $D_1, \ldots, D_n$, the ancestor-graph $\Pi(C)$ has root labeled C and subtrees $\Pi(D_1), \ldots, \Pi(D_n)$. Since every clause has its own variables, and variants are treated as distinct clauses, no clause is generated twice, and $\Pi(C)$ is uniquely defined for all $C \in S^*$.

2.4 The Theorem Proving Challenge

Theorem proving is a challenging problem, because it requires to balance contrasting requirements. The theorem-proving strategy should be *complete*, but also *efficient* in the use of time and memory, as Larry Wos emphasized in his seminal papers [187,147]. Thus, the inference system should be *refutationally complete*, while featuring powerful contraction rules to counter the combinatorial explosion of expansion inferences. The search plan should be *fair*, but *not exhaustive* as in a breadth-first search. Achieving simultaneously completeness and efficiency is so difficult in theorem proving, that not only in Larry Wos' time, but also nowadays, it is standard practice to establish completeness in theory, and then include incomplete strategies in implementations and experiments. Proving the refutational completeness of an inference system is crucial to understand it, while playing with incomplete strategies allows the experimenter to prove more theorems by machine and may give ideas for new, complete, and more efficient strategies.

For theorem-proving strategies, Larry Wos proposed *sensitivity* [187], later renamed *goal-sensitivity* [141]. The transformation of a problem of the form $H \models^? \varphi$ into a problem of the form $S \vdash_\Gamma^? \Box$, where S is the clausal form of $H \cup \{\neg\varphi\}$, loses the information about the distinction between H and φ, information that may be useful for efficiency of the search. A strategy is *goal-sensitive*, if it generates only, or preferably, clauses C such that at least a leaf of $\Pi(C)$ is labeled by a clause in the clausal form of $\neg\varphi$. Since being goal-

sensitive may not be beneficial for all problems, a strategy ought to be *flexible* with respect to goal-sensitivity [60].

For inference rules, Larry Wos suggested *immediacy*, *convergence*, and *generality* [147]. The first two properties mean that the inference rule generates neither intermediate results nor their consequences, a requirement fulfilled by *hyperinferences* in Larry's time, as we shall see in the next section. The combined usage of most general unifiers in expansion inferences, and a well-founded subsumption ordering for subsumption, fulfilled Larry's notion of *generality*, in the sense of avoiding reasoning with instances when it is possible to reason with more general clauses.

For proofs, Larry Wos stressed *brevity* and *naturalness* [187,147,186]. The quest for shorter proofs was a main driver of Larry's experimental work with OTTER, as described in another article of this issue [25]. Naturalness means that the mechanical proof ought to resemble a human proof. Although this is a recurring concern in the theorem proving literature (e.g., [151]), the development of automated theorem proving has rather led to the discovery of forms of mechanical reasoning that are different from human reasoning.

Larry Wos designed the *set-of-support strategy* [187], the *demodulation* inference rule [188], and the *paramodulation* inference rule [147], to begin addressing some of these issues. The following subsections present his ideas, connecting them with the research that followed.

3 The Set of Support Strategy

The *set-of-support strategy* was motivated by the objective of reducing *irrelevant* inferences [187] as advocated also in [67]. Larry Wos did not define formally in [187] the notion of irrelevant inference, but most likely he meant inferences that do not appear in any proofs. The set-of-support strategy was inspired by theorem-proving problems $H \models^{?} \varphi$ where H contains the axioms of a mathematical theory. Indeed, mathematics was Larry's preferred field of application for theorem proving. Since H is known to be satisfiable, generating logical consequences of H alone cannot lead to the discovery of a contradiction. Therefore, the idea of the set-of-support strategy is to forbid resolution inferences where both parents are in the clausal form of H. While the original description and completeness proof of the set-of-support strategy [187] were given for resolution and factoring only, in the next section the set-of-support strategy is presented for an inference system including the contraction rules encountered thus far, that are known to preserve the refutational completeness of resolution.

3.1 The Set of Support Strategy with Contraction

Given the input set S of clauses, obtained by transforming $H \cup \{\neg\varphi\}$ into clausal form, the set-of-support strategy partitions S into the set A of the

clauses in the clausal form of H, and the set SOS (acronym of *set of support*) of the clauses in the clausal form of $\neg\varphi$. Therefore, one can write $S = A \uplus SOS$, where $\uplus$ denotes the union of disjoint sets, or $A = S \setminus SOS$, where $\setminus$ denotes subtraction between sets. If H is satisfiable, hence consistent, so is A. If φ is an implication $\psi_1 \supset \psi_2$, so that $\neg\varphi$ is $\psi_1 \wedge \neg\psi_2$, one can also put in SOS only the clauses in the clausal form of $\neg\psi_2$, leaving in A the clauses in the clausal form of $H \cup \{\psi_1\}$, provided the resulting A is consistent [188]. Then, only resolution steps with at most one parent from the complement of the set of support are allowed, so that the set-of-support strategy is *goal-sensitive*. All resolvents are added to the set of support, leading to derivations of the form:

$$(A_0; SOS_0) \underset{\Gamma}{\vdash} (A_1; SOS_1) \underset{\Gamma}{\vdash} \ldots (A_i; SOS_i) \underset{\Gamma}{\vdash} (A_{i+1}; SOS_{i+1}) \underset{\Gamma}{\vdash} \ldots,$$

where $\forall i, i \geq 0$, $S_i = A_i \uplus SOS_i$. For the first component, $A_0 = A$ and $\forall i, i \geq 0$, A_{i+1} is derived from $(A_i; SOS_i)$ in one of the following ways:

- Add a factor of a clause in A_i;
- Delete a clause in A_i by tautology deletion or by purity deletion with respect to $A_i \uplus SOS_i$;
- Subsume a clause in A_i by a clause in $A_i \uplus SOS_i$;
- Apply clausal simplification to simplify a clause in A_i by a clause in A_i putting the simplified clause in A_{i+1}.

For the second component, $SOS_0 = SOS$ and $\forall i, i \geq 0$, SOS_{i+1} is derived from $(A_i; SOS_i)$ in one of the following ways:

- Add a resolvent of a clause in A_i and a clause in SOS_i;
- Add a resolvent of two clauses in SOS_i;
- Add a factor of a clause in SOS_i;
- Delete a clause in SOS_i by tautology deletion or by purity deletion with respect to $A_i \uplus SOS_i$;
- Subsume a clause in SOS_i by a clause in $A_i \uplus SOS_i$;
- Apply clausal simplification to simplify a clause in SOS_i by a clause in S_i putting the simplified clause in SOS_{i+1};
- Apply clausal simplification to simplify a clause in A_i by a clause in SOS_i putting the simplified clause in SOS_{i+1}.

Clauses in $\bigcup_{i \geq 0} SOS_i$ are said to be *supported*, and a resolution inference is *supported* if at least a parent is. In the original presentation of the set-of-support strategy [187], all the factors of clauses in A are added to A_0 in a pre-processing step, so that $\forall i, i \geq 0$, $A_i = A_0$, and only the set of support is expanded. Adding contraction while preserving the completeness of the set-of-support strategy requires to distinguish between deletion rules and replacement rules. The addition of a deletion rule is unproblematic: at any stage of the derivation a clause in either the set of support or its complement can be deleted by the deletion rule. Since a replacement rule is a generative rule, it can be added to the set-of-support strategy only in a way that preserves the consistency of the complement of the set of support: for clausal simplification

this means that when applying L_1 to simplify $L_2 \vee C$ to C, the new clause C can be placed in A_{i+1} only if both L_1 and $L_2 \vee C$ are in A_i, and must be placed in SOS_{i+1} otherwise [55].

Larry Wos suggested two incomplete refinements of the set-of-support strategy [187]: one is based on a *level bound* and it forbids generating clause C if the depth of $\Pi(C)$ is higher than the bound; the other one is based on a *literal bound* and it forbids generating clause C if $|C|$ is higher than the bound.

3.2 Other Supported Strategies

Other *supported strategies* [140,141,45] can be obtained by giving different definitions of the initial set of support SOS. In *resolution with forward support* SOS contains the positive input clauses. Thus, A contains the non-positive input clauses and it is satisfied by the *all-negative interpretation* $\mathcal{I}^-$ that satisfies all negative literals. In *resolution with backward support* SOS contains the negative input clauses. Thus, A contains the non-negative input clauses and it is satisfied by the *all-positive interpretation* $\mathcal{I}^+$ that satisfies all positive literals. In *resolution with user support* SOS contains any subset of S chosen by the user, provided that its complement A is satisfiable. Larry Wos' set-of-support strategy is an instance of resolution with user support.

Supported strategies where the initial set of support is defined based on sign are related to sign-based refinements of resolution. *Positive resolution*, also known as the *P1-strategy* [148,99] or *P1-deduction* [141], requires that every resolution step has a positive parent. *Negative resolution*, also known as *all-negative-resolution* [141], requires that every resolution step has a negative parent. Positive resolution is more restrictive than resolution with forward support, because the latter also allows resolutions between generated non-positive parents, as long as at least one of them is supported. The same holds for negative resolution and resolution with backward support, except in the special case of Horn clauses, because a resolution between a negative clause and a non-negative Horn clause generates a negative clause, so that only negative clauses are supported.

3.3 Semantic Strategies and Hyperinferences

The concept of not expanding a satisfiable subset of the set of clauses connects the set-of-support strategy with *semantic resolution* [161]. Semantic resolution restricts resolution by assuming a fixed interpretation $\mathcal{I}$ for *semantic guidance*. The input set S is partitioned into the subset $A = \{C : \mathcal{I} \models C\}$ of clauses satisfied by $\mathcal{I}$, and its complement $SOS = S \setminus A$, called SOS by analogy with the set-of-support strategy. However, semantic resolution moves the restriction from the parents (i.e., at most one from A) to the resolvent, by requiring that no resolvent C such that $\mathcal{I} \models C$ is generated. Given a clause $N = L_1 \vee \ldots \vee L_k \vee C$, termed *nucleus*, and k clauses $E_1 = M_1 \vee D_1, \ldots, E_k = M_k \vee D_k$, termed

electrons or *satellites*, where C and D_i, for $i = 1 \ldots k$, are disjunctions of literals, if there is a simultaneous mgu σ such that $L_i\sigma = \neg M_i\sigma$ for $i = 1 \ldots k$, semantic resolution generates the *semantic resolvent* $R = (C \vee D_1 \vee \ldots \vee D_k)\sigma$:

$$\textit{Semantic Resolution}: \quad \frac{S \cup \{N,\ E_1, \ldots, E_k\}}{S \cup \{N,\ E_1, \ldots, E_k,\ R\}} \quad \mathcal{I} \not\models R,$$

provided that $\mathcal{I} \not\models R$. Since it embeds multiple resolution steps, semantic resolution is a *hyperinference*, and it fulfills Larry's desiderata of *immediacy* and *convergence* [147], because the intermediate resolvents are not generated. *Hyperresolution* [148] is an instance of semantic resolution. If $\mathcal{I}$ is $\mathcal{I}^-$, semantic resolution yields *positive hyperresolution* that resolves away all negative literals in the nucleus with positive satellites to generate a *positive hyperresolvent*. If $\mathcal{I}$ is $\mathcal{I}^+$, semantic resolution yields *negative hyperresolution* that resolves away all positive literals in the nucleus with negative satellites to generate a *negative hyperresolvent*. Resolution with set of support [187] does not work by hyperinferences, but it fits in the paradigm of semantic resolution, assuming an ad hoc interpretation $\mathcal{I}$ such that $\mathcal{I} \models A$ and $\mathcal{I} \not\models SOS$.

Larry Wos recognized that semantic resolution is more restrictive than resolution with set of support, and that completeness of the latter can be derived from completeness of the former [188], but he was mostly concerned with the risk that neither hyperresolution nor resolution with set of support suffice in practice [188]. He was interested in enlarging what he called the *unit sections* [188] of a derivation, that is, the stretches of a derivation where the resolution steps are unit resolution steps. He had already proposed the *unit-preference strategy* [184], where unit resolution steps have priority over other resolution steps. The next move was to devise an inference rule to *generate unit clauses*. To this end, Larry Wos applied the hyperinference concept towards a syntactic property (i.e., being a unit clause) rather than a semantic one. The result was *unit-resulting resolution* [188,125], or *UR resolution* for short. UR resolution is a hyperinference geared to generate unit clauses. Given a nucleus $N = L_1 \vee \ldots \vee L_k \vee L_{k+1}$ with $k+1$ literals, and k unit satellites $M_1, \ldots, M_k$ ($k \geq 1$), UR resolution generates a unit resolvent:

$$\textit{UR Resolution}: \quad \frac{S \cup \{N,\ M_1, \ldots, M_k\}}{S \cup \{N,\ M_1, \ldots, M_k,\ L_{k+1}\sigma\}} \quad \forall i,\ 1{\leq}i{\leq}k,\ L_i\sigma = \neg M_i\sigma.$$

If literal L_{k+1} is allowed to be absent, UR resolution is allowed to generate $\Box$. This inference rule appears at the bottom of page 702 and is the main object of the first definition on page 703 of [188]. The name unit-resulting resolution appeared only much later [125], leading to the erroneous belief (e.g., [36]) that UR resolution appeared for the first time in [125]. In reality, and not surprisingly, UR resolution appeared in the same "milieu" of hyperresolution. This is testified also by the footnote on page 702 of [188], which relates the concept of UR resolution to that of *clash* in [150]. The term "clash" refers to the simultaneous resolution of multiple literals as in hyperresolution and semantic resolution (e.g., it is used systematically to present hyperinference rules in [65]).

According to [36], UR resolution was invented independently by Gerd Veenker in his PhD thesis in 1966, and published the following year [173], the same year as [188]. While the main contribution of Veenker's thesis was a complete procedure that can be considered an early forerunner of *connection-based methods* [4,33,34], Veenker also proposed a strategy, that he called the *NEU strategy*, combining unit resolution and UR resolution as in Wos' work [188]. An inference system including only unit resolution and UR resolution is incomplete, something that was well-known to both Wos and Veenker. Nonetheless, UR resolution is widely adopted as a useful enhancement, because it accelerates the generation of unit clauses that trigger in turn unit subsumption inferences that eliminate clauses and unit resolution inferences that generate shorter resolvents.

3.4 The Given-Clause Algorithm

The set-of-support strategy is also at the origin of the main algorithm inside most resolution-based theorem provers, up to those that represent the state of the art today (e.g., the E prover [157,159], SPASS [180], VAMPIRE [146, 110], WALDMEISTER [96], ZIPPERPOSITION [66], and GKC [170]). The reason is that the set-of-support strategy was built into the Argonne's provers AURA, LMA/ITP [122], and OTTER [128,127], and OTTER's main algorithm, called the *given-clause algorithm*, inspired most subsequent developers.

The given-clause algorithm maintains two lists of clauses, originally named `axioms` and `sos`. If `axioms` and `sos` are initialized with the clauses in A and SOS, respectively, the given-clause algorithm implements the set-of-support strategy, and it satisfies the invariant that no expansion inference whose premises are all in the initial `axioms` will ever be performed. If `axioms` is initialized to be empty, and `sos` is initialized to contain all input clauses, the given-clause algorithm performs all possible inferences, and it satisfies the above invariant vacuously. Thus, the connection between the given-clause algorithm and the set-of-support strategy was weakened by renaming `axioms` as `usable` in OTTER and its successor *Prover9* [126].

The given-clause algorithm executes a loop, exiting when either a proof is found, or `sos` becomes empty, which means that the input set of clauses is satisfiable, or the prover hits a predefined threshold of time or memory. At every iteration, the prover selects from `sos` a clause, termed the *given clause*, moves it from `sos` to `usable`, and performs all applicable expansion inferences having as premises the given clause and clauses in `usable`. The fact that the given clause moves from `sos` to `usable` means that even if `usable` and `sos` initially contain the clauses in A and SOS, respectively, the given-clause algorithm does *not* maintain the invariant that the clauses in `sos` are supported and those in `usable` are not supported, another reason for departing from the names `axioms` and `sos`.

If the given clause is the *best* clause according to some *heuristic evaluation function*, the given-clause algorithm performs a *best-first search*. For exam-

ple, the notion of *weight* of a clause, defined as the sum of the user-defined weights of the symbols occuring in the clause, was introduced for this purpose in OTTER [128,127]. Another feature of OTTER that became a fixture of the given-clause algorithm in most provers (e.g., [160]) is the `pick-given-ratio` parameter, which allows the strategy to mix best-first and breadth-first search. If the value of this parameter is k, the given-clause algorithm picks as given clause the oldest rather than the best clause once every $k+1$ choices. The description of the given-clause algorithm will be extended to include contraction after introducing demodulation.

4 The Demodulation Inference Rule

Larry Wos was very interested in applying theorem proving to mathematics, and since the vast majority of such problems involves *equality*, he proposed *demodulation* [188] as a contraction inference rule to replace equals by equals.

4.1 The Original Definition of Demodulation

Given an *equality unit clause*, or *equation*, $l \simeq r$, and a clause $C[l\sigma]$ containing as subterm an instance $l\sigma$ of the side l of $l \simeq r$, Larry Wos called $C[r\sigma]$ an *immediate modulant* of $C[l\sigma]$ [188]. Then a *k-modulant*, for $k > 0$, is the result of k such replacement steps, and a *modulant* is any k-modulant [188]. As a clause has infinitely many modulants in general, but only finitely many k-modulants for a fixed k, Larry Wos defined *k-modulation* as the generation of a resolvent of parents C_k and D_k, where C_k and D_k are k-modulants of clauses C and D [188]. However, Larry Wos also defined *demodulation* as replacement by a modulant, where each immediate modulant has strictly fewer symbols than its predecessor, and the final modulant has no immediate modulant with fewer symbols [188]. Thus, we can formalize his rule as follows:

$$\textit{Demodulation}: \quad \frac{S \cup \{l \simeq r,\ C[l\sigma]\}}{S \cup \{l \simeq r,\ C[r\sigma]\}} \quad \| C[l\sigma] \| \ > \ \| C[r\sigma] \|,$$

where $l \simeq r$ is called *demodulant*, $\| C \|$ is the number of symbols in C, and demodulation is defined as performing only one equational replacement step, according to the standard style for replacement rules. Subsequently, and especially in implementations, the name *demodulator* was also used in place of demodulant.

However, the intended notion of number of symbols was not made explicit. If a term is viewed as a string also parentheses contribute to the symbol count, whereas they do not if a term is viewed as a tree. Also, number of symbols is ambiguous with respect to how to count repeated occurrences of the same symbol.

The *size* of an atom is the number of occurrences of predicate, function, constant, and variable symbols. For example, $\| P(f(a), g(a)) \| = 5$. Assume

that the number of symbols in a clause is defined as the sum of the sizes of the atoms that occur in the clause. Then, the ordering whereby C is smaller than D if $\| C \| < \| D \|$ is well-founded. The ordering based on size was implemented in OTTER, and remained available alongside with more sophisticated orderings such as *recursive path orderings* [74] and *Knuth–Bendix orderings* [107] that were introduced later (cf. Sect. 4.2). Since there are infinitely many variants of a clause and they all have the same size, variants have to be eliminated by subsumption (cf. Sect. 2.2). Indeed, theorem provers such as OTTER apply subsumption *before* demodulation, so that if two clauses are variants, one is deleted by subsumption.

Nonetheless, the size-based ordering does not allow the system to apply as demodulants many equations that it would be useful to apply, because the two sides of the equation have the same number of symbols. Also, this ordering may not allow the system to apply an equation in the desired direction. For example, $\| x * (y + z) \| = 5$ and $\| x * y + x * z \| = 7$, so that the distributivity law $x * (y + z) \simeq x * y + x * z$ would be applied from right to left.

In summary, Larry Wos' definition of demodulation is well-founded, but the problem of *well-founded demodulation*, in the sense of finding more and better well-founded orderings to enable the demodulation of clauses, remained open.

4.2 Well-Founded Demodulation by Rewrite Rules

The discovery of a solution to the problem of well-founded demodulation was advanced significantly in the context of the *Knuth–Bendix completion procedure* [107,101]. This procedure works with *rewrite rules*, where a rewrite rule is an equation $l \simeq r$ that is written $l \to r$ because $l \succ r$ in a well-founded ordering $\succ$ on terms. A rewrite rule *reduces* or *rewrites* a term $t[l\sigma]_u$ to $t[r\sigma]_u$, where σ is a substitution, the notation $t[l\sigma]_u$ means that $l\sigma$ occurs as a subterm in t at position u, and $t[r\sigma]_u$ is the term obtained by replacing the occurrence of $l\sigma$ at position u with $r\sigma$. Positions are strings of natural numbers: if terms are viewed as trees and arcs are labeled with natural numbers, every subterm has a position defined as the string of natural numbers from the root to the subterm. From now on positions are omitted for simplicity.

Knuth and Bendix defined a well-founded ordering on terms, called since then the *Knuth–Bendix ordering* or KBO for short [107,124,121]. A KBO orders terms based on a *precedence* and a *weighting function*. A precedence is an ordering on symbols that may be partial or total. A weighting function assigns non-negative weights to symbols. Since the definition is parametric with respect to precedence and weighting function, it defines a family of orderings.

A KBO is a *reduction ordering*, meaning that it is well-founded, *stable* ($t \succ u$ implies $t\sigma \succ u\sigma$ for all substitutions σ), and *monotonic* ($t \succ u$ implies $c[t] \succ c[u]$ for all contexts c, where a context is a term with a hole). Another reduction ordering is the *recursive path ordering* [74], or RPO for short, that orders terms based on a *precedence* and a *status* (either multiset [74] or lexicographic [106])

of every symbol. If the status is lexicographic for all symbols, the ordering is called *lexicographic path ordering*, or LPO for short. Here too, since the definitions are parametric with respect to precedence and status, one gets families of orderings. The interested reader may find more information about orderings in surveys on rewriting [75,76,78]. Since weights are non-negative, KBO's correlate well with size, and therefore incorporate the intuition in Larry Wos' definition of demodulation, whereby clauses are made simpler by reducing the number of symbols.

The Knuth–Bendix completion procedure was formalized as an inference system [8,6,7] that transforms pairs $(E;R)$, where E is a set of equations, and R is a set of rewrite rules, such that for all rules $l \to r \in R$ it holds that $l \succ r$ in a given reduction ordering $\succ$ on terms. The inference rules of completion are seen as transforming the equational proofs of the theorems in $Th(E \cup R)$ with respect to a *proof ordering*, that is, a stable, monotonic (with respect to replacement of subproofs), and well-founded ordering $>$ on proofs [8,6,7,49]. A key property of completion is that the inference rules are *proof-reducing* [54] or *good* [49]: an inference rule deriving $(E';R')$ from $(E;R)$ is *good*, if for all theorems $s \simeq t \in Th(E \cup R)$ and for all proofs π of $s \simeq t$ in $E \cup R$ there exists a proof π' of $s \simeq t$ in $E' \cup R'$ such that $\pi \geq \pi'$.

Since the state of the derivation is a pair $(E;R)$, there are three contraction inference rules that realize well-founded demodulation by reducing a side of an equation or a side of a rewrite rule. *Simplify* reduces a side of an equation:

$$\textit{Simplify}: \quad \frac{(E \cup \{p[l\sigma] \simeq q\}; R \cup \{l \to r\})}{(E \cup \{p[r\sigma] \simeq q\}; R \cup \{l \to r\})}$$

where $\simeq$ is symmetric. *Compose* reduces the right-hand side of a rewrite rule, so that another rewrite rule is produced:

$$\textit{Compose}: \quad \frac{(E; R \cup \{p \to q[l\sigma], l \to r\})}{(E; R \cup \{p \to q[r\sigma],\ l \to r\})}$$

Collapse reduces the left-hand side of a rewrite rule, so that an equation is produced:

$$\textit{Collapse}: \quad \frac{(E; R \cup \{p[l\sigma] \to q, l \to r\})}{(E \cup \{p[r\sigma] \simeq q\}; R \cup \{l \to r\})} \qquad p[l\sigma] \rhd l,$$

where $\rhd$ is the strict *encompassment ordering* on terms. If an equation in E has the form $s \simeq s$, the *Delete* inference rule removes it. If an equation $p \simeq q$ in E is such that $p \succ q$, the *Orient* inference rule removes $p \simeq q$ from E and adds $p \to q$ to R.

The *encompassment ordering* is obtained by combining the *subterm ordering* and the *subsumption ordering* on terms. The subterm ordering is defined by $t \trianglelefteq s$ if $s = c[t]$ for some context c. The subsumption ordering on terms is defined by $s \preceq t$ if $t = s\vartheta$ for some substitution ϑ. Terms s and t are variants, written $s \doteq t$, if $s \preceq t$ and $t \preceq s$. The encompassment ordering is defined by

$t \trianglerighteq s$ if $t = c[s\vartheta]$ for some context c and substitution ϑ. The strict encompassment ordering is defined by $t \triangleright s$ if $t \trianglerighteq s$ and $s \not\trianglerighteq t$, that is, $t = c[s\vartheta]$ where either the context c is not empty or the substitution ϑ is not a variable renaming.

The purpose of the strict encompassment condition of the *Collapse* inference rule is to prevent $l \to r$ from reducing $p[l\sigma]$ if l and $p[l\sigma]$ are variants. The reason is that such a step is not good (in the above sense of proof-reducing) [8,6,7]. In the Knuth–Bendix procedure the co-existence of two rewrite rules whose left-hand sides are variants is avoided by giving *Simplification* higher priority than *Orient*. If $p \simeq q$ and $l \simeq r$ are two equations such that $p \succ q$, $l \succ r$, and $p \doteq l$, one of them, say $p \simeq q$, gets oriented first into $p \to q$, so that $p \to q$ simplifies $l \simeq r$ to $q \simeq r$ *before* $l \simeq r$ may get oriented into $l \to r$.

If an equation in E can be neither simplified, nor deleted, nor oriented, the procedure fails. Thus, Knuth–Bendix completion provided only a partial solution to the problem of well-founded demodulation.

4.3 Well-Founded Demodulation by Equations

Knuth–Bendix completion solved the problem of well-founded demodulation at the price of considering as demodulants only those equations that can be oriented into rewrite rules by the adopted ordering. This limitation was removed with the inception of *unfailing* [98] or *ordered* [9,6,7] completion, henceforth completion for short. Completion allows the inference system to use equations as demodulants provided the applied instance is oriented by the ordering.

Completion is a theorem-proving strategy for problems of the form $E \models^{?} \forall\bar{x}.s \simeq t$, where E is a set of equations, the presentation of an equational theory, and $\bar{x}$ is the vector of all variables in $s \simeq t$ [101,98,9,52,42,54]. The negation of the conjecture yields $\hat{s} \not\simeq \hat{t}$, where $\hat{s}$ and $\hat{t}$ are s and t, respectively, with all the variables in $\bar{x}$ replaced by Skolem constants. The given ordering $\succ$ on terms is assumed to be a reduction ordering [9,6,7], or a *complete simplification ordering* (CSO) [98]. A simplification ordering is stable, monotonic, and with the *subterm property*, which means that it includes the strict subterm ordering (i.e., $p \rhd l$ implies $p \succ l$). A simplification ordering is well-founded [74], hence it is a reduction ordering. A complete simplification ordering is also total on ground terms. KBO's, RPO's, and LPO's are simplification orderings. KBO's and LPO's are CSO's if the precedence is total, but not all RPO's are CSO's [10].

As it is no longer necessary to separate equations and rewrite rules, and completion is seen as theorem proving, the inference system can be written [52, 42,54] as transforming pairs $(E; \hat{s} \not\simeq \hat{t})$, where $\hat{s} \not\simeq \hat{t}$ is called the *target*. The inference rules of completion are good [49] or proof-reducing [42,54] with respect to all ground theorems, which is enough for theorem proving, since the target is ground. The objective of the derivation is to reduce $\hat{s}$ and $\hat{t}$ to a common form so as to discover a contradiction with $x \simeq x$, the clausal form of the reflexivity axiom for equality. Accordingly, one can distinguish between

Simplification of the target:

$$\frac{(E \cup \{l \simeq r\}; \hat{s}[l\sigma] \not\simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma] \not\simeq \hat{t})} \quad l\sigma \succ r\sigma,$$

and *Simplification of the presentation*:

$$\frac{(E \cup \{p[l\sigma] \simeq q,\ l \simeq r\}; \hat{s} \not\simeq \hat{t})}{(E \cup \{p[r\sigma] \simeq q,\ l \simeq r\}; \hat{s} \not\simeq \hat{t})} \quad l\sigma \succ r\sigma,\ (p[l\sigma] \rhd l \vee q \succ p[r\sigma]),$$

where $l \simeq r$ is called a *simplifier*, and the second condition incorporates the side condition of *Collapse*. This side condition for simplification lets $l \simeq r$ simplify $p[l\sigma] \simeq q$ when $p[l\sigma]$ is a variant of l, but q is not a variant of r, provided that $q \succ p[r\sigma]$, or, equivalently, $q \succ r\sigma$ (if $p[l\sigma] \doteq l$, the context p is empty, σ is a variable renaming, $p[l\sigma] = l\sigma$, and $p[r\sigma] = r\sigma$). For example, simplifying $f(e, x) \simeq x$ by $f(e, y) \simeq y$ is not allowed; simplifying $f(e, x) \simeq h(x)$ by $f(e, y) \simeq y$ is allowed as $h(x) \succ x$; simplifying $f(e, y) \simeq y$ by $f(e, x) \simeq h(x)$ is not allowed as $y \not\succ h(y)$.

The next challenge was to generalize simplification to clauses, as intended in Larry Wos' definition of demodulation, while preserving as much as possible the behavior of simplification in completion. This requires to extend the ordering $\succ$ beyond terms. A step in this direction was achieved with the inference system in [155]. This system assumes that the ordering $\succ$ is a CSO on terms and atoms that satisfies two additional properties. First, for all terms l, r, p, and q, such that $l \succeq r$, and for all atoms A, (i) if $l \trianglelefteq A$ and the predicate symbol of A is not $\simeq$, then $(l \simeq r) \prec A$; and (ii) if $l \triangleleft p$ or $l \triangleleft q$, then $(l \simeq r) \prec (p \simeq q)$. Second, for all ground terms l, r, and s, and for all ground atoms A, if $l \succ r$, $l \succ s$, and $(l \simeq r) \prec A \prec (l \simeq s)$, then A has the form $l \simeq t$ for some ground term t.

This definition is illustrated in [155] with a predicate-first extension of a CSO $\succ$ on terms to atoms. It assumes a total precedence on predicate symbols such that $\simeq$ is the smallest predicate symbol. Then, $P(s_1, \ldots, s_m) \prec Q(t_1, \ldots, t_n)$ holds if P is smaller than Q in the precedence, or $P = Q \neq \simeq$ and $(s_1, \ldots, s_m) \prec_{lex} (t_1, \ldots, t_n)$, or $P = Q = \simeq$ and $(s_1, s_2) \prec_{mul} (t_1, t_2)$, where $\prec_{lex}$ and $\prec_{mul}$ are the lexicographc and multiset extensions of $\prec$, respectively.

The inference system in [155] includes a simplification inference rule that allows a simplifier $l \simeq r$ to simplify a clause $C[l\sigma]$ to $C[r\sigma]$, if $l\sigma \succ r\sigma$ and $C[l\sigma]$ contains an atom A such that $A \succ (l\sigma \simeq r\sigma)$. While the simplification rule of [155] allows some simplification, it does not preserve the behavior of simplification in completion.

Example 2 Given equations $\{(1)\ f(x) \simeq g(x),\ (2)\ g(h(y)) \simeq k(y)\}$, target theorem $f(h(b)) \not\simeq k(b)$, and precedence $f > g > h > k > b$, the *Simplification* rule of completion allows equation (1) to simplify the target to $g(h(b)) \not\simeq k(b)$, with matching substitution $\sigma = \{x \leftarrow h(b)\}$, since $f(h(b)) \succ g(h(b))$ (and $f(h(b)) \rhd f(x)$ if the literal to be simplified were positive). Another step by

the same simplification rule applies equation (2) to simplify $g(h(b)) \not\simeq k(b)$ to $k(b) \not\simeq k(b)$, with matching substitution $\vartheta = \{y \leftarrow b\}$, since $g(h(b)) \succ k(b)$ (and $g(h(b)) \rhd g(h(y))$ in the positive case). On the other hand, the simplification rule of [155] cannot perform these steps, and hence cannot yield a refutation by simplification. For example, for the first step, $\{f(h(b)), k(b)\} \succ_{mul} \{f(h(b)), g(h(b))\}$ does not hold.

The footnote on page 2 of [10] says[1] that the method of [155] "does discuss simplification to some extent, but for practical purposes his simplification tecniques are inadequate even for the very simplest case – completion of sets of universally quantified equations." The issue is the generalization of the ordering beyond terms. The inference system of the *superposition calculus* [10] –henceforth $\mathcal{SP}$– offered a solution with a systematic way to extend a reduction ordering on terms to atoms, literals, and clauses. The reduction ordering is assumed to be complete or *completable*, which means it is included in a complete ordering. All RPO's are completable [10]. The first step is to treat non-equational literals as equational literals by treating non-equational atoms like terms, and reading a positive literal L as $L \simeq \top$ and a negative literal $\neg L$ as $L \not\simeq \top$, where $\top$ is a new symbol such that $t \succ \top$ for all terms t.

The second step is to extend the ordering $\succ$ on terms to literals. This can be done in one of two ways. One way is to treat an equation $p \simeq q$ as the multiset $\{p, q\}$, a negated equation $p \not\simeq q$ as the multiset $\{p, p, q, q\}$, and compare literals in the multiset extension of the ordering on terms. The other one is to treat an equation $p \simeq q$ as the multiset of multisets $\{\{p\}, \{q\}\}$, a negated equation $p \not\simeq q$ as the multiset of multisets $\{\{p, \bot\}, \{q, \bot\}\}$, where $\bot$ is a new symbol such that $t \succ \bot \succ \top$ for all terms t, and compare literals by taking twice the multiset extension of the ordering on terms. The third step is to extend the ordering on literals to clauses by taking once more the multiset extension.

Simplification appears in $\mathcal{SP}$ as an instance of an inference rule called *contextual reductive rewriting* [10]. If the simplifier is a unit equational clause, contextual reductive rewriting yields the following rule:

$$\textit{Simplification}: \quad \frac{S \cup \{C[l\sigma],\ l \simeq r\}}{S \cup \{C[r\sigma],\ l \simeq r\}} \quad l\sigma \succ r\sigma,\ C[l\sigma] \succ (l\sigma \simeq r\sigma).$$

The second side condition requires that the applied instance of the simplifier is smaller than the clause it simplifies. This condition is only superficially similar to the one of the simplification rule in [155] as the difference is in the ordering.

Example 3 Consider the problem in Example 2. The simplification rule of $\mathcal{SP}$ allows both simplification steps, because $\{f(h(b)), f(h(b)), k(b), k(b)\} \succ_{mul} \{f(h(b)), g(h(b))\}$ holds for the first step, and $\{g(h(b)), g(h(b)), k(b), k(b)\} \succ_{mul} \{g(h(b)), k(b)\}$ holds for the second step.

[1] This footnote appears on page 2 of the technical report version of [10] available at https://pure.mpg.de.

However, *Simplification* of $\mathcal{SP}$ and *Simplification* of completion do not behave in general in the same way in the purely equational case.

Example 4 If $b \succ c$, both the *Collapse* rule of Knuth–Bendix completion and the *Simplification* rule of completion allow $f(x) \to b$ to simplify $f(b) \to c$ to $b \to c$, with matching substitution $\sigma = \{x \leftarrow b\}$, because $f(b) \succ b$ and $f(b) \rhd f(x)$. On the other hand, $\{f(b), c\} \succ_{mul} \{f(b), b\}$ does not hold, so that *Simplification* of $\mathcal{SP}$ does not allow the step.

A comparison of the second condition for *Simplification* in completion with the second condition for *Simplification* of $\mathcal{SP}$ explains the difference. Assume that the ordering on terms is a CSO. The second condition for *Simplification* in completion is $p[l\sigma] \rhd l \vee q \succ p[r\sigma]$. The second condition for *Simplification* in $\mathcal{SP}$ when an equation $l \simeq r$ simplifies a unit positive equational clause $p[l\sigma] \simeq q$ is $\{p[l\sigma], q\} \succ_{mul} \{l\sigma, r\sigma\}$.

If the *Simplification* rule of completion applies because $p[l\sigma] \rhd l$ holds as p is not empty, we have $p[l\sigma] \succ l\sigma \succ r\sigma$ by the subterm property and the condition $l\sigma \succ r\sigma$ in both simplification rules. Thus, $\{p[l\sigma], q\} \succ_{mul} \{l\sigma, r\sigma\}$ follows and *Simplification* of $\mathcal{SP}$ applies.

If the *Simplification* rule of completion applies because $p[l\sigma] \rhd l$ does not hold (p is empty and σ is a variable renaming) and $q \succ p[r\sigma]$ holds, we have $p[l\sigma] = l\sigma$, $p[r\sigma] = r\sigma$, and hence $q \succ r\sigma$, so that $\{p[l\sigma], q\} \succ_{mul} \{l\sigma, r\sigma\}$ follows and *Simplification* of $\mathcal{SP}$ applies.

If the *Simplification* rule of completion applies because $p[l\sigma] \rhd l$ holds as p is empty, but σ is not a variable renaming, and $q \succ p[r\sigma]$ does not hold, then $\{p[l\sigma], q\} \succ_{mul} \{l\sigma, r\sigma\}$ does not follow, and *Simplification* of $\mathcal{SP}$ does not apply. Example 4 illustrates this situation, where completion lets a simplifier (i.e., $f(x) \to b$) rewrite a proper instance of its left-hand side (i.e., $f(b)$) even if the right-hand side of the simplifier (i.e., b) is larger than the right-hand side of the rewrite rule to be simplified (i.e., c).

It is also interesting to see how simplification is implemented. For instance, the E prover [157] distinguishes between *Simplification of negative literals* and *Simplification of positive literals*. The former is

$$\frac{S \cup \{p[l\sigma] \not\simeq q \vee D,\ l \simeq r\}}{S \cup \{p[r\sigma] \not\simeq q \vee D,\ l \simeq r\}} \quad l\sigma \succ r\sigma,$$

because $l\sigma \succ r\sigma$ implies $\{p[l\sigma], p[l\sigma], q, q\} \succ_{mul} \{l\sigma, r\sigma\}$. Indeed, if p is not empty, $\{p[l\sigma], p[l\sigma], q, q\} \succ_{mul} \{l\sigma, r\sigma\}$ follows from $p[l\sigma] \succ l\sigma \succ r\sigma$ as discussed above. If p is empty, $\{l\sigma, l\sigma, q, q\} \succ_{mul} \{l\sigma, r\sigma\}$ follows from $l\sigma \succ r\sigma$. Thus, it suffices to consider the literal being rewritten to establish the second condition of *Simplification* of $\mathcal{SP}$. On the other hand, *Simplification of positive literals* embeds conditions from the inference rules of completion:[2]

$$\frac{S \cup \{p[l\sigma] \simeq q \vee D,\ l \simeq r\}}{S \cup \{p[r\sigma] \simeq q \vee D,\ l \simeq r\}} \quad \begin{array}{c} l\sigma \succ r\sigma, \\ (\exists M \in D.M \succ (p[l\sigma] \simeq q)) \vee p[l\sigma] \not\succ q \vee p[l\sigma] \rhd l. \end{array}$$

[2] The inference rule in [157] is reproduced assuming that there is no selection function: the interested reader may find more details in [10,157].

Consider the second condition. If the first disjunct is true, either $\{M, \top\} \succ_{mul} \{p[l\sigma], q\}$ or $\{M, M, \top, \top\} \succ_{mul} \{p[l\sigma], q\}$. Either way, $M \succ p[l\sigma]$ and $M \succ q$, so that $M \succ p[l\sigma] \succeq l\sigma \succ r\sigma$ holds. Thus, either $\{M, \top\} \succ_{mul} \{l\sigma, r\sigma\}$ or $\{M, M, \top, \top\} \succ_{mul} \{l\sigma, r\sigma\}$ holds and the second condition of *Simplification* of $\mathcal{SP}$ is fulfilled. If the second disjunct $p[l\sigma] \not\succ q$ is true, the step is an instance of *Simplify* or *Compose*. If the third disjunct $p[l\sigma] \rhd l$ is true, the step is an instance of *Collapse*. Although the second disjunct $q \succ p[r\sigma]$ in the second condition of *Simplification* in completion does not appear, this confirms that the conditions for simplification from completion are important in the practice.

Larry Wos' intuition of demodulation as comprising multiple steps, until no further step can be applied, was captured in the context of completion and rewriting with the notion of *normalization*, or reduction to *normal form*. A clause C is in normal form with respect to a set S of clauses, if no unit equational clause in S can simplify it; equivalently, C is *irreducible* with respect to S, or S*-irreducible*. The normal form of C with respect to S is denoted $C{\downarrow_S}$, where $C{\downarrow_S} = C$ if C is S-irreducible.

4.4 Demodulation and the Given-Clause Algorithm

Larry Wos was interested in the application of demodulation in the context of the set-of-support strategy [188], which leads to the more general issue of the application of demodulation in the given-clause algorithm. The goal is to ensure that the given-clause algorithm implements an *eager-contraction search plan*, namely one where contraction has priority over expansion (e.g., [56]). In other words, the objective is to prevent a clause that can be deleted or replaced from playing the role of parent in an expansion inference.

In the given-clause algorithm, when a new clause C is generated by expansion, C is subject to *forward contraction*, that is, contraction with respect to a set S of already existing clauses. The prover tries first the deletion rules. Thus, C may be deleted by tautology deletion, or by purity deletion, or by subsumption by a clause in S (*forward subsumption*), or because it is a unit equational clause $s \simeq s$.

If clause C survives these tests, the prover tries the replacement rules. Thus, C may be simplified by clausal simplification by a clause in S, or reduced to $C{\downarrow_S}$ by demodulation with the demodulants in S. Let $C{\downarrow_S}$ represent the final result of the application of all applicable replacement rules. If C, and hence $C{\downarrow_S}$, is an equation, the test to determine whether it can be oriented is applied to $C{\downarrow_S}$. Thus, the implementation of contraction respects the requirement from completion of orienting equations only after their sides have been normalized (cf. Sect. 4.2).

Only at this stage clause $C{\downarrow_S}$ gets an identifier and is appended to the `sos` list. Therefore, forward contraction is part of the generation of a new clause. Indeed, in OTTER this phase is called *preprocessing* of a clause. Also the test for the generation of the empty clause happens during preprocessing: if $C{\downarrow_S}$ is a unit clause the prover tests whether it generates the empty clause with a unit

clause in `usable` or `sos`. This is because one wants to get the empty clause as soon as possible. Thus, the test for a contradiction is applied as soon as a unit clause is generated, without waiting until it is selected as given clause.

For *backward contraction* the prover tests whether $C\downarrow_S$ can contract a previously existing clause $D \in S$. In OTTER this phase is called *post-processing* of a clause. For all $D \in S$ for which this is the case, D is treated like if it were a newly generated clause, and subjected to forward contraction as described above. The resulting $D\downarrow_S$ gets a *new identifier* and is appended to the `sos` list. Thus, a clause generated by backward contraction is treated as a clause generated by expansion.

There are two versions of the given-clause algorithm, named from the OTTER prover [128,127] and the E prover [73,157,158,159], respectively. The two versions differ primarily in the implementation of backward contraction. In both versions the set S of clauses in the above description of forward contraction is given by `usable` $\cup$ `sos`, meaning the union of the set of clauses in `usable` and the set of clauses in `sos`. On the other hand, the set S of clauses in the above description of backward contraction is `usable` $\cup$ `sos` in the OTTER version, whereas it is `usable` in the E version.

The OTTER version of the given-clause algorithm aims at maintaining the set `usable` $\cup$ `sos` *inter-reduced* or, more generally, *contracted* [49]. Suppose that the expansion inferences between a given clause C and the clauses in `usable` generate a bunch of new clauses, each of whom is subjected to forward contraction as described above, so that clauses $C_0, \ldots, C_k$ get appended to `sos`. In the OTTER version, the prover tests whether C_i, for all i, $0 \leq i \leq k$, can backward-contract any clause in `usable` $\cup$ `sos`. Suppose that for all i, $0 \leq i \leq k$, backward-contraction by C_i appends clauses $D^i_0, \ldots, D^i_{n_i}$ to `sos`. Then, for all i, $0 \leq i \leq k$, for all j, $0 \leq j \leq n_i$, the prover tests whether D_j can backward-contract any clause in `usable` $\cup$ `sos`. The process continues until no more contraction applies.

Example 5 Suppose that `sos` contains (1) $f(g(x)) \simeq b$ and (2) $h(f(y)) \simeq c$, and OTTER derives $g(z) \simeq z$ by some inference and appends it to `sos` as (3) $g(z) \simeq z$. OTTER applies (3) to back-demodulate (1) to $f(x) \simeq b$, removes (1), and appends (4) $f(x) \simeq b$ to `sos`. Then OTTER applies (4) to back-demodulate (2) to $h(b) \simeq c$, removes (2), and appends (5) $h(b) \simeq c$ to `sos`.

The E version of the given-clause algorithm aims at maintaining `usable` contracted. The prover tests whether a clause C can backward-contract any clause in `usable` only when C is selected as given clause and moved from `sos` to `usable`. As `usable` may have changed since the time when C was subjected to forward contraction, the prover first applies the clauses in `usable` to contract C, and then applies C to contract the clauses in `usable`, before trying the expansion inferences between C and clauses in `usable`. If a clause in `usable` is removed by backward contraction, its descendants in `sos` are deleted as *orphans*. Except for orphan deletion, all backward contraction happens in `usable`. The rationale is that maintaining `usable` contracted is good enough, because the premises of expansion inferences come from `usable`.

Example 6 Given the initial situation as in Example 5, E applies no backward demodulation in `sos`. Suppose that E selects (3) $g(z) \simeq z$ as given clause before (1) and (2). Thus, (3) moves from `sos` to `usable`. E applies (3) to back-demodulate (1) $f(g(x)) \simeq b$ only when (1) is selected as given clause and joins (3) in `usable`. As a result, E deletes (1) and appends (4) $f(x) \simeq b$ to `sos`. Suppose that E selects (4) as given clause before (2), so that (4) moves from `sos` to `usable`. E applies (4) to back-demodulate (2) $h(f(y)) \simeq c$ only when (2) is selected as given clause and joins (4) in `usable`. As a result, E deletes (2) and appends (5) $h(b) \simeq c$ to `sos`. If E selects (2) as given clause before (4), E applies (4) to back-demodulate (2) only when (4) is selected as given clause and joins (2) in `usable`. As a result, E deletes (2), appends (5) $h(b) \simeq c$ to `sos`, and deletes any orphan of (2) in `sos`.

In the E version of the given-clause algorithm the lists `usable` and `sos` were renamed `active` and `passive`, respectively. The E version was born primarily from a concern that the cost of backward contraction as in the OTTER version could outweight its benefits. For example, it may happen that the prover spends a lot of time doing backward contraction, when it would be more beneficial to go ahead with expansion, because an expansion inference with the next given clause would generate a unit clause that yields the contradiction. On the other hand, the delay in backward contraction in the E version may cause the `passive` list to grow too much, reaching a memory limit, or it may delay finding a proof. For example, it may happen that the prover goes ahead to do more expansion, postponing backward demodulation steps in `sos` that would generate a unit clause that yields the contradiction.

In practice, most clauses that get deleted are deleted by forward contraction. Then, expansion and backward demodulation can be seen as two ways to generate clauses that need to be balanced. One could say that the OTTER version leans toward prioritizing backward demodulation and the E version leans toward prioritizing expansion. There is no conclusive evidence that one is better than the other in general. Most theorem provers feature both versions of the given-clause algorithm, because one pays off on some problems and the other on others.

5 The Paramodulation Inference Rule

Adding demodulation to resolution does not suffice for refutational completeness in first-order logic with equality. Larry Wos started the research on *paramodulation* [147], precisely to complement resolution and demodulation with an expansion inference rule for equality that would yield a refutationally complete inference system for first-order logic with equality. This quest turned out to be one of the most fascinating in the history of automated theorem proving.

5.1 The Original Definition of Paramodulation

Prior to the inception of paramodulation, the only way to reason about equality in resolution-based theorem proving was to add to the input set the clausal form of the axioms of equality:

$$x \simeq x \qquad (\textit{Reflexivity})$$
$$x \not\simeq y \vee y \simeq x \qquad (\textit{Symmetry})$$
$$x \not\simeq y \vee y \not\simeq z \vee x \simeq z \qquad (\textit{Transitivity})$$
$$\bigvee_{i=1}^{n} x_i \not\simeq y_i \vee f(\bar{x}) \simeq f(\bar{y}) \ (\textit{Function Substitutivity})$$
$$\bigvee_{i=1}^{n} x_i \not\simeq y_i \vee \neg P(\bar{x}) \vee P(\bar{y}) \ (\textit{Predicate Substitutivity})$$

for all function symbols f and predicate symbols P of arity n, where $\bar{x}$ and $\bar{y}$ stand for $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively. It soon emerged that these axioms are so general that their presence causes resolution to generate so many clauses that the efficiency of the inference system is unbearably compromised in most cases. Thus, George A. Robinson and Larry Wos introduced *paramodulation* [147] as a generalization of resolution with equality built-in:

$$\textit{Paramodulation}: \quad \frac{S \cup \{l \simeq r \vee C,\ M[t] \vee D\}}{S \cup \{l \simeq r \vee C,\ M[t] \vee D,\ (C \vee M[r] \vee D)\sigma\}} \quad l\sigma = t\sigma,$$

where $\simeq$ is regarded as symmetric, σ is the mgu of a side l of the equation $l \simeq r$ and a subterm t of a literal M in a clause $M[t] \vee D$, and C and D are disjunctions of literals. Clause $l \simeq r \vee C$ is called the *clause paramodulated from*, or *para-from clause* for short, and $l \simeq r$ is the *literal paramodulated from*, or *para-from literal* for short. Clause $M[t] \vee D$ is called the *clause paramodulated into*, or *para-into clause* for short, and $M[t]$ is the *literal paramodulated into*, or *para-into literal* for short. The generated clause $(C \vee M[r] \vee D)\sigma$ is termed a *paramodulant*.

While the appearance of paramodulation represented a breakthrough, a proof of refutational completeness could be obtained only under the assumption that the input set includes not only $x \simeq x$, but also the *functionally reflexive axioms*, that is, the instances of reflexivity of the form $f(\bar{x}) \simeq f(\bar{x})$, for all function symbols f. Furthermore, the original paramodulation inference rule is very prolific, because the term t paramodulated into can be a variable, which unifies with any term. However, paramodulation into variables could not be excluded, because it was necessary to prove a *paramodulation lifting lemma* [147] analogous to the lifting lemma used in the proof of completeness of resolution [149,65]. In order to show that to every paramodulation between ground instances of clauses corresponds a paramodulation between the general clauses themselves, paramodulation into variables was needed, because the ground term paramodulated into could be the instance of a variable. The

conjecture that paramodulation is refutationally complete without the functionally reflexive axioms and without paramodulating into variables became known as the *Wos–Robinson conjecture*.

A first step towards settling the Wos–Robinson conjecture was represented by the *modification method* [61]. This method consists of pre-processing the input set of clauses with respect to the equality axioms (the "modification" in the name), and then applying resolution and factoring to the modified set of clauses, without including axioms other than $x \simeq x$. The completeness of resolution, factoring, and paramodulation without functionally reflexive axioms follows via a simulation argument, provided some paramodulations into variables are allowed. The Wos–Robinson conjecture was still considered open, because a direct proof of the refutational completeness of resolution, factoring, and paramodulation, without functionally reflexive axioms, and with no paramodulation into variables, was not given. Another challenge that remained open was to prove refutational completeness in the presence of demodulation and other contraction inference rules.

5.2 Superposition Between Rewrite Rules or Equations

Unaware of paramodulation,[3] Knuth and Bendix coined the name *superposition* for a related inference rule, which is the main mechanism of the *Knuth–Bendix completion procedure* [107,101]. In the formalization of completion as an inference system [8,6,7], derivation states have the form $(E; R)$, where E is a set of equations, and R is a set of rewrite rules oriented by a reduction ordering $\succ$ on terms (cf. Sect. 4.2). Then, superposition is defined as follows:

$$\textit{Superposition}: \quad \frac{(E; R \cup \{l \to r,\ p[t] \to q\})}{(E \cup \{p[r]\sigma \simeq q\sigma\}; R \cup \{l \to r,\ p[t] \to q\})} \quad t \notin X,\ l\sigma{=}t\sigma,$$

where σ is the mgu of the left-hand side l of a rewrite rule and a non-variable subterm t of the left-hand side of another rewrite rule, X is the set of variable symbols, and the generated equation $p[r]\sigma \simeq q\sigma$ is called a *critical pair*. If the critical pair cannot be simplified, deleted, or oriented, the procedure fails.

As with demodulation, *unfailing* [98] or *ordered*[9,6,7] completion removed the limitation of working only with rewrite rules, leading to *Superposition* of equations:

$$\frac{E \cup \{l \simeq r,\ p[t] \simeq q\}}{E \cup \{l \simeq r,\ p[t] \simeq q,\ p[r]\sigma \simeq q\sigma\}} \quad t \notin X,\ l\sigma{=}t\sigma,\ l\sigma \not\preceq r\sigma,\ p[t]\sigma \not\preceq q\sigma,$$

where E is a set of equations, and the equations $l \simeq r$ and $p[t] \simeq q$ are allowed to superpose only if their instances according to the mgu σ are either orientable (i.e., $l\sigma \succ r\sigma$) or uncomparable (i.e., $l\sigma \not\succ r\sigma \wedge r\sigma \not\succ l\sigma \wedge l\sigma \neq r\sigma$, abbreviated $l\sigma \# r\sigma$). The ordering $\succ$ on terms is a CSO [98] or a reduction ordering [9,6,7].

[3] Mark E. Stickel, personal communication, October 1996.

This superposition inference rule is less general than paramodulation, as it applies only to unit equational clauses, but it avoids superposition into variables, is restricted by the ordering, and is refutationally complete for problems of the form $E \models^{?} \forall\bar{x}.s \simeq t$ also in the presence of contraction. The contraction rules of completion are deletion of equations of the form $s \simeq s$, simplification, subsumption, and another subsumption rule for equations based on the encompassment ordering [98]:

$$\textit{Functional Subsumption}\colon \quad \frac{E \cup \{l \simeq r,\ p \simeq q\}}{S \cup \{l \simeq r\}} \quad (p \simeq q) \rhd (l \simeq r),$$

where $(p \simeq q) \rhd (l \simeq r)$ if $p = c[l\vartheta]$, $q = c[r\vartheta]$, and either the context c is not empty or the substitution ϑ is not a variable renaming.

A *challenge related to the Wos–Robinson conjecture* was how to obtain an inference system for first-order logic with equality that avoids paramodulating or superposing into variables, is restricted by the ordering, is refutationally complete also in the presence of contraction, and reduces to completion if given an input of the form $E \cup \{\hat{s} \not\simeq \hat{t}\}$.

5.3 Paramodulation and Superposition

The next step towards settling the Wos–Robinson conjecture and related challenges was a proof that an inference system with *resolution*, *factoring*, *paramodulation*, *subsumption*, and *simplification* is refutationally complete for first-logic with equality, without adding equality axioms other than $x \simeq x$ and without paramodulating into variables [133]. A key feature of this approach is a CSO on terms and atoms that is *order-isomorphic to the positive integers*. This ordering is used for simplification, and, in the proof of completeness, to build semantic trees based on an enumeration of the Herbrand base, where an equation $l \simeq r$ appears *before* any atom that $l \simeq r$ can simplify. The issue encountered by Wos and Robinson with the paramodulation lifting lemma is solved by showing that it suffices to consider substitutions that replace variable by *irreducible* terms, so that the substitution cannot replace a variable with a ground term that can be simplified, or, equivalently, paramodulated into [133,154,99,155].

A KBO is order-isomorphic to the positive integers, provided weights are positive [133], but RPO's and most other orderings are not. Thus, the Wos–Robinson conjecture was considered truly solved only when this requirement on the ordering was lifted. This result was reached with the proof of refutational completeness of an inference system called the *ordered-literal strategy* [154,99,155]. The ordered-literal strategy, or, rather, the *ordered-literal inference system* features *resolution*, *factoring*, *paramodulation*, and *superposition* as expansion inference rules, and *tautology deletion*, *subsumption*, *clausal simplification*, *demodulation*, and *functional subsumption* as contraction inference rules.

A key characteristic of this inference system, and the reason for its name, is that the expansion inference rules are restricted to work on literals that are strictly maximal in a CSO on terms and atoms. Since clauses are multisets of literals, a literal L is *maximal* in a clause C if $\neg(\exists M \in C.\ M \succ L)$, or, equivalently, $\forall M \in C.\ L \not\prec M$. In other words, the other literals can only be smaller, equal, or uncomparable. A literal L is *strictly maximal* in a clause C if $\neg(\exists M \in C.\ M \succeq L)$, or, equivalently, $\forall M \in C.\ L \not\preceq M$. In other words, the other literals can only be smaller or uncomparable. If the ordering is defined on atoms as in [154,99,155], literals are identified with their atoms when applying the ordering. The proof that the ordered-literal inference system is refutationally complete without adding equality axioms other than $x \simeq x$, without paramodulating into variables, and without the requirement that the CSO on terms and atoms is order-isomorphic to the positive integers, was obtained by working with *transfinite semantic trees* [154,99,155].

Resolution and factoring are restricted to resolve upon strictly maximal literals:

$$\textit{Resolution}: \quad \frac{S \cup \{L_1 \vee C,\ L_2 \vee D\}}{S \cup \{L_1 \vee C,\ L_2 \vee D,\ (C \vee D)\sigma\}} \quad L_1\sigma = \neg L_2\sigma,\ (1),\ (2)$$

$$\textit{Factoring}: \quad \frac{S \cup \{L_1 \vee \ldots \vee L_k \vee C\}}{S \cup \{L_1 \vee \ldots \vee L_k \vee C,\ (L_1 \vee C)\sigma\}} \quad L_1\sigma = L_2\sigma = \ldots L_k\sigma,\ (1)$$

where (1) is $\forall M \in C.\ L_1\sigma \not\preceq M\sigma$ and (2) is $\forall M \in D.\ L_2\sigma \not\preceq M\sigma$. These ordering-based restrictions to resolution and factoring appeared in [99] and have remained in the subsequent ordering-based inference systems, including the superposition calculus $\mathcal{SP}$ where a reduction ordering on terms is extended to literals as seen in Sect. 4.3.

For paramodulation and superposition, the challenge of solving the Wos-Robinson conjecture was intertwined with the challenge of obtaining inference rules for first-order logic with equality that reduce to the superposition rule of completion in the purely equational case. In completion superposition is restricted to work on maximal sides of equations (cf. Sect. 5.2). Thus, collecting the restrictions on literals and those on sides of equational literals, one gets *four ordering-based conditions*. In order to state them, we recall some terminology and notation that applies to all versions of paramodulation and superposition in this section. The para-from clause is written $l \simeq r \vee C$, where $l \simeq r$ is the para-from literal. The para-into clause is written $M[t] \vee D$, where $M[t]$ is the para-into literal. If the inference system distinguishes the case where the para-into literal is an equational literal, the para-into clause is written $p[t] \simeq q \vee D$ or $p[t] \not\simeq q \vee D$, where $p[t] \simeq q$ or $p[t] \not\simeq q$ is the para-into literal, respectively. One can also say *superposed-from* and *superposed-into* with the analogous meanings. The subterm t is *not* a variable (i.e., $t \notin X$ where X is the set of variable symbols), and the substitution σ is the mgu of the terms l and t (i.e., $l\sigma = t\sigma$).

The *four ordering-based conditions* involved in restricting paramodulation and superposition are the following:

(i) The para-from literal is *strictly maximal* in the instance of the para-from clause: $\forall Q \in C.\ (l \simeq r)\sigma \not\preceq Q\sigma$;
(ii) The left-hand side of the para-from literal is *strictly maximal* in the instance of the para-from literal: $l\sigma \not\preceq r\sigma$;
(iii.a) The para-into literal is *strictly maximal* in the instance of the para-into clause: $\forall Q \in D.\ M[t]\sigma \not\preceq Q\sigma$ or $\forall Q \in D.\ (p[t] \simeq q)\sigma \not\preceq Q\sigma$;
(iii.b) If the para-into literal is a negative equational literal $p[t] \not\simeq q$, it is *maximal* in the instance of the para-into clause: $\forall Q \in D.\ (p[t] \not\simeq q)\sigma \not\prec Q\sigma$;
(iv) If the para-into literal is a positive equational literal $p[t] \simeq q$, its left-hand side is *strictly maximal* in the instance of the para-into literal: $p[t]\sigma \not\preceq q\sigma$.

The ordered-literal inference system in [99] added to resolution and factoring as above the following *Paramodulation* inference rule:

$$\frac{S \cup \{l \simeq r \vee C,\ M[t] \vee D\}}{S \cup \{l \simeq r \vee C,\ M[t] \vee D,\ (C \vee M[r] \vee D)\sigma\}} \quad (i),\ (ii),\ (iii.a).$$

Similar to the original paramodulation inference rule (cf. Sect. 5.1), this inference rule does not distinguish whether the para-into literal is equational or not. The requirement that $t \notin X$ and three ordering-based conditions out of four represented major restrictions with respect to the paramodulation inference rule of Robinson and Wos.

Aiming at the challenge of lifting to first-order logic superposition as in completion, the inference system of [155] replaced the paramodulation inference rule of [99] with two rules, one called *superposition* and one called *paramodulation*. *Superposition* applies if the *para-into literal* is a positive equational literal:

$$\frac{S \cup \{l \simeq r \vee C,\ p[t] \simeq q \vee D\}}{S \cup \{l \simeq r \vee C,\ p[t] \simeq q \vee D,\ (C \vee p[r] \simeq q \vee D)\sigma\}} \quad (ii),\ (iii.a),\ (iv).$$

Paramodulation was used if the *para-into literal* $M[t]$ is a non-equational literal or a negative equational literal:

$$\frac{S \cup \{l \simeq r \vee C,\ M[t] \vee D\}}{S \cup \{l \simeq r \vee C,\ M[t] \vee D,\ (C \vee M[r] \vee D)\sigma\}} \quad (ii),\ (iii.a).$$

Thus, *Superposition* has Conditions (ii) and (iv) from superposition in completion (cf. Sect. 5.2), but both rules had to drop Condition (i). The inference system in [155] includes the contraction inference rules.[4] However, as discussed in Sect. 4.3, due to the choice of the ordering, demodulation as in [155] does not reproduce the behavior of the simplification rule of completion in the equational case. Therefore, the inference system of [155] generalized completion only as far as the superposition inference rule is concerned.

[4] The inference system in [99] does not list the contraction inference rules referring to [154] for contraction.

The conjecture as to whether an ordering-based inference system is still refutationally complete, if *all four ordering-based conditions are imposed* remained open. It was answered affirmatively with the development of the *superposition calculus* $\mathcal{SP}$ [10]. As already discussed in Sect. 4.3 for demodulation, a basic, but crucial, ingredient is the appropriate extension of the ordering on terms to literals. Another key ingredient is the addition of a new expansion inference rule [10,130]:

$$\textit{Equational Factoring} \quad \frac{C \vee u \simeq s \vee u' \simeq s'}{(C \vee s \not\simeq s' \vee u \simeq s')\sigma} \quad u\sigma = u'\sigma,\ u\sigma \not\preceq s\sigma,\ (v),$$

where Condition (v) is $\forall Q \in C \cup \{u' \simeq s'\}.\ (u \simeq s)\sigma \not\prec Q\sigma$. This rule is a generalization of factoring that can be seen as a conditional factoring rule. If it holds that $u\sigma = u'\sigma$ and $s\sigma = s'\sigma$, that is $(u \simeq s)\sigma = (u' \simeq s')\sigma$, factoring can be applied. Equational factoring tests only $u\sigma = u'\sigma$, provided $u\sigma \not\preceq s\sigma$, and adds $s\sigma \simeq s'\sigma$ as a condition, hence negated, in the generated clause.

The superposition calculus $\mathcal{SP}$ uses only the name superposition [10,130]. In $\mathcal{SP}$ even resolution becomes a special case of superposition, because all literals are transformed into equational literals as seen in Sect. 4.3. However, for continuity, we refrain from subsuming resolution into superposition, and we still use the name *paramodulation* when the para-into literal is not equational. For *Paramodulation* the three applicable ordering-based conditions are restored:

$$\frac{S \cup \{l \simeq r \vee C,\ M[t] \vee D\}}{S \cup \{l \simeq r \vee C,\ M[t] \vee D,\ (C \vee M[r] \vee D)\sigma\}} \quad (i),\ (ii),\ (iii.a).$$

Superposition affords all four ordering-based conditions:

$$\frac{S \cup \{l \simeq r \vee C,\ p[t] \simeq q \vee D\}}{S \cup \{l \simeq r \vee C,\ p[t] \simeq q \vee D,\ (C \vee p[r] \simeq q \vee D)\sigma\}}\ (i),\ (ii),\ (iii.a),\ (iv),$$

$$\frac{S \cup \{l \simeq r \vee C,\ p[t] \not\simeq q \vee D\}}{S \cup \{l \simeq r \vee C,\ p[t] \not\simeq q \vee D,\ (C \vee p[r] \not\simeq q \vee D)\sigma\}}\ (i),\ (ii),\ (iii.b),\ (iv),$$

with the weaker version of the third one (Condition $(iii.b)$ in place of Condition $(iii.a)$) when the para-into literal is negative. These versions of *paramodulation* and *superposition*, together with *resolution*, *factoring*, *equational factoring*, *tautology deletion*, *subsumption*, and *simplification* form the refutationally complete inference system $\mathcal{SP}$ for first-order logic with equality. The proof of refutational completeness was obtained by an approach based on *rewrite models* [10] that became a standard (e.g., [123]) and was reformulated also in terms of semantic trees [92].

In summary, the superposition calculus [10,11,12] imposed the strongest known ordering-based restrictions on expansion rules, and met the challenge of getting a refutationally complete inference system for first-order logic with equality that reduces to completion if the input is purely equational.[5] For

[5] Modulo the discrepancies between simplification in completion and simplification in $\mathcal{SP}$ in the purely equational case as seen in Sect. 4.3.

these reasons, the superposition calculus became the standard ordering-based inference system.

6 Discussion

With *set of support* [187], *demodulation* [188], and *paramodulation* [147], Larry Wos contributed three fundamental ideas that have nurtured research on theorem proving for decades, and are still fruitful today.

The *set-of-support strategy* influenced both search plan design, as witnessed by the *given-clause algorithm* [128,127], and inference rule design, beginning with *semantic resolution* [161]. Since the given-clause algorithm is at the heart of contemporary provers (e.g., [96,180,110,66,170,159]), it continues to be an object of study. The design of *heuristic evaluation functions* for the selection of the given clause has been an active research topic (e.g., [1,71,128,72,44, 127,160]). For example, the search may employ multiple evaluation functions by maintaining multiple priority queues with either parallel search [46] or interleaving [159]. The weight of clauses can be used to break ties when the best clause according to an evaluation function is not unique [91].

The ideas in the set-of-support strategy and in semantic resolution were generalized and developed into notions of *semantic guidance*, *goal-sensitivity*, and *hyperinference*, that had an impact also beyond resolution-based theorem proving, including tableaux-based methods (e.g., [19,14,17]), instance-based methods (e.g., [142]), and SGGS [59,60] (see [51] for a survey with an emphasis on these features).

The challenge of getting the theorem-proving strategy to focus on the conjecture to be proved, that Larry Wos meant to address with the set-of-support strategy, is more relevant than ever, given the growth of large and very large knowledge bases, in mathematics and other domains (e.g., [145,172]). The existence of such knowledge bases also poses the problem of applying theorem proving to check their consistency: the meaning and impact of semantic guidance and goal-sensitivity for this problem is still uncharted territory.

Larry's concept of *irrelevant*, or, dually, *relevant*, inference and clauses was formalized and generalized [94], and his notion that inferences should be general resurfaced in investigations of *abstraction* in resolution theorem proving (e.g., [135]). The already mentioned *instance-based methods* (see [45, 103,108,51] for surveys) explore a complementary direction that is often most fruitful for model building given a satisfiable input.

Larry's *UR resolution* hyperinference rule [188] became a standard feature of resolution-based theorem provers and beyond. For example, UR resolution was used to generate unit lemmas for PTTP (*Prolog Technology Theorem Proving*) provers [167] that implemented *model elimination* [118,119] on top of a Prolog engine such as the *Warren Abstract Machine* [179]. A similar idea was pursued in a parallel setting [169]. Both sequential and parallel tableaux-based theorem provers such as SETHEO [115] and CPTHEO [86] preprocessed the input with respect to UR resolution, unit resolution, and unit subsumption.

The thread of research that Larry Wos opened with the notion of *demodulation* has been a major one in theorem proving and continues to the present. *Well-founded demodulation* is a fundamental inference rule for equality in all reasoning contexts. Under the name of *simplification* or *rewriting*, it was generalized to *conditional rewriting*, or reasoning in Horn equational logic (e.g., [109, 10, 50]), and to *contextual rewriting* (e.g, [189, 181, 97, 90]), with applications also beyond theorem proving. Furthermore, Larry's notion of applying the rule for a predefined number k of steps, as in k-modulation, may still be useful in practice. For example, it may be employed as a form of pre-processing when no suitable well-founded ordering orients defining equations in the desired direction [174].

The efficient implementation of demodulation, and more generally contraction, is an active research topic, because theorem provers may spend a lot of time performing contraction (e.g, [97]). As it is typical in theorem proving, the issue is one of finding a good balance between the eagerness and the cost of contraction. For example, one can distinguish between full-fledged contraction and *cheap simplification* (e.g., demodulation by rewrite rules) [158, 159] or *light simplification* (e.g., demodulation by ground rewrite rules) [80] that are less expensive and can be applied more eagerly. If given clause C generates a set N of new clauses, *immediate simplification* [80] consists of inter-reducing N and then applying it to backward-contract the clauses in `usable`. If clause C itself is deleted in the process, all clauses in N can be deleted as orphans, except those that justify the deletion of C.

Larry Wos pioneered a *paramodulation* principle for building equality into resolution, that many other researchers, over several decades, endeavoured to bring to maturity, merging it successfully with *completion-based theorem proving* (e.g., [107, 101, 98, 9, 6, 7, 54, 49]). The resulting *ordering-based inference systems* (e.g., [133, 99, 155, 10]) are refutationally complete, combining expansion inference rules such as resolution, factoring, paramodulation, and superposition, with contraction inference rules such as subsumption and well-founded demodulation or simplification. The number of years and people involved, starting from different angles and with different motivations, shows the greatness of Larry's original insight.

Subsumption and simplification are based on distinct well-founded orderings. An abstract framework to treat in a unified manner these two contraction principles was developed [178]. Another area of investigation is the reproduction and verification of the proofs of refutational completeness of ordering-based inference systems (e.g., [12]) in proof assistants [156, 178].

Ordering-based inference systems were implemented first in the Otter theorem prover [128, 127], that Larry used for his experiments throughout his long career, and then in most subsequent resolution-based theorem provers, up to those that represent the state of the art for first-order logic with equality (e.g., the E prover [159], Spass [180], Vampire [110], Waldmeister [96], Zipperposition [174], and GKC [170]). The growth of superposition-based theorem proving was a main reason for the evolution of the given-clause algorithm from an implementation of the set of support strategy into a general algo-

rithm for implementing multiple strategies. Indeed, the set-of-support strategy is not complete in general for either ordered resolution in first-order logic, or paramodulation and superposition in first-order logic with equality, unless the complement of the set of support is saturated with respect to the inference system in a preprocessing phase [10], which defeats the spirit of the strategy. Making reasoning goal-sensitive, or target-oriented, is more challenging in the presence of equality [54].

The power and flexibility of ordering-based inference systems is witnessed by the fact that they allow some *theory reasoning* (e.g., [166,134,104,100, 13,54,89,168,177,79]) yield *decision procedures* (e.g., [87,164,165,102,5,84]), get interfaced with other reasoning paradigms (e.g., [23,58,144,80]), form the basis of approaches to *parallel theorem proving* (e.g., [42,53,43,46] and [48] for a survey), and are generalized to higher-order logic as in *lambda-superposition* [24,40,28,175,176,26,131,27] and in *combinatory superposition* [29,30,31,32].

Acknowledgements Parts of this work were done while the author was participating in a program at the Simons Institute for the Theory of Computing, and visiting the Computer Science Laboratory of SRI International, whose support is greatly appreciated. The author thanks very much the anonymous reviewers for their precious technical remarks, and Wolfgang Bibel for his comments on the early history of theorem proving.

References

1. Anantharaman, S., Andrianarivelo, N.: Heuristical criteria in refutational theorem proving. In: A. Miola (ed.) Proceedings of 1st International Symposium on Design and Implementation of Symbolic Computation Systems (DISCO), *Lecture Notes in Computer Science*, vol. 429, pp. 184–193. Springer, Berlin (1990)
2. Anantharaman, S., Bonacina, M.P.: Automated proofs in Łukasiewicz logic. Tech. rep., Department of Computer Science, State University of New York at Stony Brook and LIFO, Université d'Orléans (1989)
3. Anantharaman, S., Bonacina, M.P.: An application of automated equational reasoning to many-valued logic. In: M. Okada, S. Kaplan (eds.) Proceedings of 2nd International Workshop on Conditional and Typed Term Rewriting Systems (CTRS 1990), *Lecture Notes in Computer Science*, vol. 516, pp. 156–161. Springer, Berlin (1991)
4. Andrews, P.B.: Theorem proving via general matings. J. ACM **28**(2), 193–214 (1981)
5. Armando, A., Bonacina, M.P., Ranise, S., Schulz, S.: New results on rewrite-based satisfiability procedures. ACM Trans. Comput. Log. **10**(1), 129–179 (2009)
6. Bachmair, L.: Canonical Equational Proofs. Birkhäuser, Boston (1991)
7. Bachmair, L., Dershowitz, N.: Equational inference, canonical proofs, and proof orderings. J. ACM **41**(2), 236–276 (1994)
8. Bachmair, L., Dershowitz, N., Hsiang, J.: Orderings for equational proofs. In: Proceedings of 1st Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 346–357. IEEE (1986)
9. Bachmair, L., Dershowitz, N., Plaisted, D.A.: Completion without failure. In: H. Aït-Kaci, M. Nivat (eds.) Resolution of Equations in Algebraic Structures, vol. II: Rewriting Techniques, pp. 1–30. Academic Press (1989)
10. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)
11. Bachmair, L., Ganzinger, H.: Equational reasoning in saturation-based theorem proving. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction - A Basis for Applications, *Applied Logic Series*, vol. I: Foundations - Calculi and Methods, chap. 11, pp. 352–397. Kluwer Academic Publishers, Dordrecht (1998)

12. Bachmair, L., Ganzinger, H., McAllester, D., Lynch, C.A.: Resolution theorem proving. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. 1, chap. 2, pp. 19–99. Elsevier, Amsterdam (2001)
13. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. Appl. Alg. Eng. Commun. and Comput. **5**, 193–212 (1994)
14. Baumgartner, P.: Hyper tableaux – the next generation. In: H. de Swart (ed.) Proceedings of 7th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX), *Lecture Notes in Artificial Intelligence*, vol. 1397, pp. 60–76. Springer, Berlin (1998)
15. Baumgartner, P.: Logical engineering with instance-based methods. In: F. Pfenning (ed.) Proceedings of 21st International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 4603, pp. 404–409. Springer, Berlin (2007)
16. Baumgartner, P., Eisinger, N., Furbach, U.: A confluent connection calculus. In: H. Ganzinger (ed.) Proceedings of 16th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 1632, pp. 329–343. Springer, Berlin (1999)
17. Baumgartner, P., Fröhlich, P., Furbach, U., Nejdl, W.: Semantically guided theorem proving for diagnosis applications. In: Proceedings of 16th International Joint Conference on Artificial Intelligence (IJCAI), vol. 1, pp. 460–465 (1997)
18. Baumgartner, P., Furbach, U.: Variants of clausal tableaux. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction - A Basis for Applications, vol. I: Foundations - Calculi and Methods, chap. 3, pp. 73–102. Kluwer Academic Publishers, Dordrecht (1998)
19. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper tableaux. In: J.J. Alferes, L.M. Pereira, E. Orłowska (eds.) Proceedings of 5th Joint European Workshop on Logic in Artificial Intelligence (JELIA), *Lecture Notes in Artificial Intelligence*, vol. 1126, pp. 1–17. Springer, Berlin (1996)
20. Baumgartner, P., Furbach, U., Pelzer, B.: The hyper tableaux calculus with equality and an application to finite model computation. J. Log. Comput. **20**(1), 77–109 (2008)
21. Baumgartner, P., Pelzer, B., Tinelli, C.: Model evolution with equality - revised and implemented. J. Symb. Comput. **47**(9), 1011–1045 (2012)
22. Baumgartner, P., Tinelli, C.: The model evolution calculus as a first-order DPLL method. Artificial Intelligence **172**(4–5), 591–632 (2008)
23. Baumgartner, P., Waldmann, U.: Superposition and model evolution combined. In: R.A. Schmidt (ed.) Proceedings of 22nd International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 5663, pp. 17–34. Springer, Berlin (2009)
24. Becker, H., Blanchette, J., Waldmann, U., Wand, D.: A transfinite Knuth-Bendix order for lambda-free higher-order terms. In: L. de Moura (ed.) Proceedings of 26th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 10395, pp. 432–453. Springer, Berlin (2017)
25. Beeson, M., Bonacina, M.P., Kinyon, M., Sutcliffe, G.: Larry Wos – Visions of automated reasoning. J. Autom. Reason. **in press** (2022). Available at http://doi.org/10.1007/s10817-022-09620-8
26. Bentkamp, A., Blanchette, J., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic. Log. Methods Comput. Sci. **17**(2), 1–38 (2021)
27. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P.: Superposition for full higher-order logic. In: A. Platzer, G. Sutcliffe (eds.) Proceedings of 28th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 12699, pp. 396–412. Springer, Berlin (2021)
28. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas. In: P. Fontaine (ed.) Proceedings of 27th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 11716, pp. 55–73. Springer, Berlin (2019)
29. Bhayat, A., Reger, G.: Set of support for higher-order reasoning. In: B. Konev, P. Rümmer, J. Urban (eds.) Proceedings of 6th Workshop on Practical Aspects in Automated Reasoning (PAAR), *CEUR Workshop Proceedings*, vol. 2162, pp. 2–16 (2018)

30. Bhayat, A., Reger, G.: Restricted combinatory unification. In: P. Fontaine (ed.) Proceedings of 27th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 11716, pp. 74–93. Springer, Berlin (2019)
31. Bhayat, A., Reger, G.: A combinator-based superposition calculus for higher-order logic. In: N. Peltier, V. Sofronie-Stokkermans (eds.) Proceedings of 10th International Joint Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 12166, pp. 278–296. Springer, Berlin (2020)
32. Bhayat, A., Reger, G.: A Knuth-Bendix-like ordering for orienting combinator equations. In: N. Peltier, V. Sofronie-Stokkermans (eds.) Proceedings of 10th International Joint Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 12166, pp. 259–277. Springer, Berlin (2020)
33. Bibel, W.: On matrices with connections. J. ACM **28**(4), 633–645 (1981)
34. Bibel, W.: Automated Theorem Proving, 2nd edn. Friedr. Vieweg & Sohn, Braunschweig (1987)
35. Bibel, W.: Deduction: Automated Logic. Academic Press, New York (1993)
36. Bibel, W.: Early history and perspectives of automated deduction. In: J. Hertzberg, M. Beetz, R. Englert (eds.) Proceedings of 31st German Annual Conference on Artificial Intelligence (KI), *Lecture Notes in Artificial Intelligence*, vol. 4667, pp. 2–18. Springer, Berlin (2007)
37. Bibel, W., Eder, E.: Methods and calculi for deduction. In: D.M. Gabbay, C.J. Hogger, J.A. Robinson (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, vol. I: Logical Foundations, pp. 68–183. Oxford University Press, Oxford (1993)
38. Billon, J.P.: The disconnection method. In: P. Miglioli, U. Moscato, D. Mundici, M. Ornaghi (eds.) Proceedings of 5th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX), *Lecture Notes in Artificial Intelligence*, vol. 1071, pp. 110–126. Springer, Berlin (1996)
39. Blake, A.: Canonical expressions in Boolean algebras. Ph.D. thesis, University of Chicago (1937)
40. Blanchette, J., Fontaine, P., Schulz, S., Waldmann, U.: Towards strong higher-order automation for fast interactive verification. In: G. Reger, D. Treytel (eds.) Proceedings of 1st Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements (ARCADE), *EPiC Series in Computing*, vol. 51, pp. 16–23. EasyChair (2017)
41. Bonacina, M.P.: Problems in Łukasiewicz logic. Newsletter of the Association for Automated Reasoning, No. 18, pages 5–12 (1991). Available at http://aarinc.org/Newsletters/018-1991-06.pdf
42. Bonacina, M.P.: Distributed automated deduction. Ph.D. thesis, Department of Computer Science, State University of New York at Stony Brook (1992)
43. Bonacina, M.P.: On the reconstruction of proofs in distributed theorem proving: a modified Clause-Diffusion method. J. Symb. Comput. **21**(4–6), 507–522 (1996)
44. Bonacina, M.P.: Mechanical proofs of the Levi commutator problem. In: P. Baumgartner, U. Furbach, M. Kohlhase, W.W. McCune, W. Reif, M.E. Stickel, T. Uribe (eds.) Proceedings of CADE Workshop on Problem Solving Methodologies with Automated Deduction, pp. 1–10 (1998)
45. Bonacina, M.P.: A taxonomy of theorem-proving strategies. In: M.J. Wooldridge, M. Veloso (eds.) Artificial Intelligence Today – Recent Trends and Developments, *Lecture Notes in Artificial Intelligence*, vol. 1600, pp. 43–84. Springer, Berlin (1999)
46. Bonacina, M.P.: Combination of distributed search and multi-search in Peers-mcd.d. In: R.P. Gore, A. Leitsch, T. Nipkow (eds.) Proceedings of 1st International Joint Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 2083, pp. 448–452. Springer, Berlin (2001)
47. Bonacina, M.P.: On conflict-driven reasoning. In: N. Shankar, B. Dutertre (eds.) Proceedings of 6th Workshop on Automated Formal Methods (AFM) May 2017, *Kalpa Publications*, vol. 5, pp. 31–49. EasyChair (2018)
48. Bonacina, M.P.: Parallel theorem proving. In: Y. Hamadi, L. Sais (eds.) Handbook of Parallel Constraint Reasoning, chap. 6, pp. 179–235. Springer, Berlin (2018)
49. Bonacina, M.P., Dershowitz, N.: Abstract canonical inference. ACM Trans. Comput. Log. **8**(1), 180–208 (2007)

50. Bonacina, M.P., Dershowitz, N.: Canonical ground Horn theories. In: A. Voronkov, C. Weidenbach (eds.) Programming Logics: Essays in Memory of Harald Ganzinger, *Lecture Notes in Computer Science*, vol. 7797, pp. 35–71. Springer, Berlin (2013)
51. Bonacina, M.P., Furbach, U., Sofronie-Stokkermans, V.: On first-order model-based reasoning. In: N. Martí-Oliet, P. Olveczky, C. Talcott (eds.) Logic, Rewriting, and Concurrency: Essays Dedicated to José Meseguer, *Lecture Notes in Computer Science*, vol. 9200, pp. 181–204. Springer, Berlin (2015)
52. Bonacina, M.P., Hsiang, J.: Completion procedures as semidecision procedures. In: M. Okada, S. Kaplan (eds.) Proceedings of 2nd International Workshop on Conditional and Typed Term Rewriting Systems (CTRS 1990), *Lecture Notes in Computer Science*, vol. 516, pp. 206–232. Springer, Berlin (1991)
53. Bonacina, M.P., Hsiang, J.: On subsumption in distributed derivations. J. Autom. Reason. **12**, 225–240 (1994)
54. Bonacina, M.P., Hsiang, J.: Towards a foundation of completion procedures as semidecision procedures. Theoret. Comput. Sci. **146**, 199–242 (1995)
55. Bonacina, M.P., Hsiang, J.: On semantic resolution with lemmaizing and contraction and a formal treatment of caching. New Gener. Comput. **16**(2), 163–200 (1998)
56. Bonacina, M.P., Hsiang, J.: On the modelling of search in theorem proving – towards a theory of strategy analysis. Inf. Comput. **147**, 171–208 (1998)
57. Bonacina, M.P., Johansson, M.: Interpolation systems for ground proofs in automated deduction: a survey. J. Autom. Reason. **54**(4), 353–390 (2015)
58. Bonacina, M.P., Lynch, C.A., de Moura, L.: On deciding satisfiability by theorem proving with speculative inferences. J. Autom. Reason. **47**(2), 161–189 (2011)
59. Bonacina, M.P., Plaisted, D.A.: Semantically-guided goal-sensitive reasoning: model representation. J. Autom. Reason. **56**(2), 113–141 (2016)
60. Bonacina, M.P., Plaisted, D.A.: Semantically-guided goal-sensitive reasoning: inference system and completeness. J. Autom. Reason. **59**(2), 165–218 (2017)
61. Brand, D.: Proving theorems with the modification method. SIAM J. Comput. **4**(4), 412–430 (1975)
62. Chang, C.C.: Algebraic analysis of many-valued logics. Trans. Am. Math. Soc. **88**, 467–490 (1958)
63. Chang, C.C.: Proof of an axiom of Łukasiewicz. Trans. Am. Math. Soc. **87**, 55–56 (1958)
64. Chang, C.C.: A new proof of the completeness of the Łukasiewicz axioms. Trans. Am. Math. Soc. **93**, 74–80 (1959)
65. Chang, C.L., Lee, R.C.T.: Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York (1973)
66. Cruanes, S.: Extending superposition with integer arithmetic, structural induction, and beyond. Ph.D. thesis, École Polytechnique, Université Paris-Saclay (2015)
67. Davis, M.: Eliminating the irrelevant from mechanical proofs. In: Proceedings of 15th Symposium for Applied Mathematics, pp. 15–30 (1963). Also in J. Siekmann and G. Wrightson (Eds.) Automation of Reasoning 1 – Classical Papers on Computational Logic 1957-1966, 315–330, Springer, Berlin, 1983
68. Davis, M.: The prehistory and early history of automated deduction. In: J. Siekmann, G. Wrightson (eds.) Automation of Reasoning 1 – Classical Papers on Computational Logic 1957-1966, pp. 1–28. Springer, Berlin (1983)
69. Davis, M.: The Universal Computer. The Road from Leibniz to Turing. Mathematics/Logic/Computing Series. CRC Press, Taylor and Francis Group (2012). Turing Centenary Edition
70. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**, 201–215 (1960)
71. Denzinger, J., Fuchs, M.: Goal-oriented equational theorem proving using Team-Work. In: B. Nebel, L. Dreschler-Fischer (eds.) Proceedings of 18th German Conference on Artificial Intelligence (KI), *Lecture Notes in Artificial Intelligence*, vol. 861, pp. 343–354. Springer, Berlin (1994)
72. Denzinger, J., Fuchs, M.: A comparison of equality reasoning heuristics. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction - A Basis for Applications, *Applied Logic Series*, vol. II: Systems and Implementation Techniques, chap. 13, pp. 361–382. Kluwer Academic Publishers, Dordrecht (1998)

73. Denzinger, J., Kronenburg, M., Schulz, S.: Discount: a distributed and learning equational prover. J. Autom. Reason. **18**(2), 189–198 (1997)
74. Dershowitz, N.: Orderings for term-rewriting systems. Theoret. Comput. Sci. **17**(3), 279–301 (1982)
75. Dershowitz, N.: Termination of rewriting. J. Symb. Comput. **3**, 69–116 (1987)
76. Dershowitz, N., Jouannaud, J.P.: Rewrite systems. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 243–320. Elsevier, Amsterdam (1990)
77. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM **22**(8), 465–476 (1979)
78. Dershowitz, N., Plaisted, D.A.: Rewriting. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. 1, chap. 9, pp. 535–610. Elsevier, Amsterdam (2001)
79. Dohan, K., Lynch, C.: Equational theorem proving modulo. In: A. Platzer, G. Sutcliffe (eds.) Proceedings of 28th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 12699, pp. 166–182. Springer, Berlin (2021)
80. Duarte, A., Korovin, K.: Implementing superposition in iProver. In: N. Peltier, V. Sofronie-Stokkermans (eds.) Proceedings of 10th International Joint Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 12167, pp. 388–397. Springer, Berlin (2020)
81. Dunham, B., Fridshal, R., Sward, G.L.: A non-heuristic program for proving elementary logical theorems. In: Proceedings of 1st International Conference on Information Processing, pp. 282–285. UNESCO House (1960). Also in J. Siekmann and G. Wrightson (Eds.) Automation of Reasoning 1 – Classical Papers on Computational Logic 1957-1966, 93–98, Springer, Berlin, 1983
82. Dunham, B., North, J.H.: Theorem testing by computer. In: Proceedings of Sympos, pp. 173–177. Polytechnic Press (1963). Also in J. Siekmann and G. Wrightson (Eds.) Automation of Reasoning 1 – Classical Papers on Computational Logic 1957-1966, 173–177, Springer, Berlin, 1983
83. Eisinger, N., Ohlbach, H.J.: Deduction systems based on resolution. In: D.M. Gabbay, C.J. Hogger, J.A. Robinson (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, vol. I: Logical Foundations, pp. 184–273. Oxford University Press, Oxford (1993)
84. Fietzke, A., Weidenbach, C.: Superposition as a decision procedure for timed automata. Math. in Comput. Sci. **6**(4), 409–425 (2012)
85. Font, J.M., Rodríguez, A.J., Torrens, A.: Wajsberg algebras. Stochastica **8**(1), 5–31 (1984)
86. Fuchs, M., Wolf, A.: Cooperation in model elimination: CPtheo. In: C. Kirchner, H. Kirchner (eds.) Proceedings of 15th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 1421, pp. 42–46. Springer, Berlin (1998)
87. Ganzinger, H., de Nivelle, H.: A superposition decision procedure for the guarded fragment with equality. In: Proceedings of 14th Annual IEEE Symposium on Logic in Computer Science (LICS). IEEE (1999)
88. Ganzinger, H., Korovin, K.: New directions in instantiation-based theorem proving. In: Proceedings of 18th Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 55–64. IEEE (2003)
89. Ganzinger, H., Waldmann, U.: Theorem proving in cancellative Abelian monoids. In: M.A. McRobbie, J.K. Slaney (eds.) Proceedings of 13th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 1104, pp. 388–402. Springer, Berlin (1996)
90. Gleiss, B., Kovàcs, L., Rath, J.: Subsumption demodulation in first-order theorem proving. In: N. Peltier, V. Sofronie-Stokkermans (eds.) Proceedings of 10th International Joint Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 12166, pp. 297–315. Springer, Berlin (2020)
91. Gleiss, B., Suda, M.: Layered clause selection for theory reasoning (short paper). In: N. Peltier, V. Sofronie-Stokkermans (eds.) Proceedings of 10th International Joint Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 12166, pp. 402–409. Springer, Berlin (2020)

92. Goubault-Larrecq, J., Jouannaud, J.P.: The blossom of finite semantic trees. In: A. Voronkov, C. Weidenbach (eds.) Programming Logics: Essays in Memory of Harald Ganzinger, *Lecture Notes in Computer Science*, vol. 7797, pp. 90–122. Springer, Berlin (2013)
93. Hähnle, R.: Tableaux and related methods. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, chap. 3, pp. 101–178. Elsevier, Amsterdam (2001)
94. Haifani, F., Tourret, S., Weidenbach, C.: Generalized completeness for SOS resolution and its application to a new notion of relevance. In: A. Platzer, G. Sutcliffe (eds.) Proceedings of 28th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 12699, pp. 327–343. Springer, Berlin (2021)
95. Herbrand, J.J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, École Normale Supérieure, Université de Paris (1930). Published in Travaux Soc. Sciences et Lettres Varsovie, Cl. 3 (Mathem. Phys.), 1930, and in Engl. transl. in W. D. Goldfarb (Ed.) Logical Writings of Jacques Herbrand, Reidel, Dordrecht, 1968
96. Hillenbrand, T.: Citius, altius, fortius: lessons learned from the theorem prover WALDMEISTER. In: I. Dahn, L. Vigneron (eds.) Proceedings of 4th International Workshop on First-Order Theorem Proving (FTP), *Electronic Notes in Theoretical Computer Science*, vol. 86. Elsevier, Amsterdam (2003)
97. Hillenbrand, T., Piskac, R., Waldmann, U., Weidenbach, C.: From search to computation: redundancy criteria and simplification at work. In: A. Voronkov, C. Weidenbach (eds.) Programming Logics: Essays in Memory of Harald Ganzinger, *Lecture Notes in Computer Science*, vol. 7797, pp. 169–193. Springer, Berlin (2013)
98. Hsiang, J., Rusinowitch, M.: On word problems in equational theories. In: T. Ottman (ed.) Proceedings of 14th International Colloquium on Automata, Languages, and Programming (ICALP), *Lecture Notes in Computer Science*, vol. 267, pp. 54–71. Springer, Berlin (1987)
99. Hsiang, J., Rusinowitch, M.: Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. J. ACM **38**(3), 559–587 (1991)
100. Hsiang, J., Rusinowitch, M., Sakai, K.: Complete inference rules for the cancellation laws. In: Proceedings of 10th International Joint Conference on Artificial Intelligence (IJCAI), pp. 990–992 (1987)
101. Huet, G.: A complete proof of correctness of the Knuth–Bendix completion algorithm. J. Comput. Syst. Sci. **23**(1), 11–21 (1981)
102. Ihlemann, C., Jacobs, S., Sofronie-Stokkermans, V.: On local reasoning in verification. In: C.R. Ramakrishnan, J. Rehof (eds.) Proceedings of 14th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), *Lecture Notes in Computer Science*, vol. 4963, pp. 265–281. Springer, Berlin (2008)
103. Jacobs, S., Waldmann, U.: Comparing instance generation methods for automated reasoning. J. Autom. Reason. **38**, 57–78 (2007)
104. Jouannaud, J., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM J. Comput. **15**(4), 1155–1194 (1986)
105. Joyner Jr., W.H.: Resolution strategies as decision procedures. J. ACM **23**(3), 398–417 (1976)
106. Kamin, S., Lévy, J.J.: Two generalizations of the recursive path ordering. Unpublished note, Department of Computer Science, University of Illinois at Urbana-Champaign (1980)
107. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: J. Leech (ed.) Proceedings of Conference on Computational Problems in Abstract Algebras, pp. 263–298. Pergamon Press, Oxford (1970)
108. Korovin, K.: An invitation to instantiation-based reasoning: from theory to practice. In: R.A. Schmidt (ed.) Proceedings of 22nd International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 5663, pp. 163–166. Springer, Berlin (2009)
109. Kounalis, E., Rusinowitch, M.: On word problems in Horn theories. J. Symb. Comput. **11**(1–2), 113–128 (1991)
110. Kovàcs, L., Voronkov, A.: First order theorem proving and Vampire. In: N. Sharygina, H. Veith (eds.) Proceedings of 25th International Conference on Computer-Aided Verification (CAV), *Lecture Notes in Computer Science*, vol. 8044, pp. 1–35. Springer, Berlin (2013)

111. Kowalski, R.A.: Studies in the completeness and efficiency of theorem proving by resolution. Ph.D. thesis, University of Edinburgh (1970)
112. Lee, S.J., Plaisted, D.A.: Eliminating duplication with the hyperlinking strategy. J. Autom. Reason. **9**, 25–42 (1992)
113. Leitsch, A.: The Resolution Calculus. Springer, Berlin (1997)
114. Letz, R.: Clausal tableaux. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction - A Basis for Applications, vol. I: Foundations - Calculi and Methods, chap. 2, pp. 43–72. Kluwer Academic Publishers, Dordrecht (1998)
115. Letz, R., Schumann, J., Bayerl, S., Bibel, W.: SEtheo: a high performance theorem prover. J. Autom. Reason. **8**(2), 183–212 (1992)
116. Letz, R., Stenz, G.: Model elimination and connection tableau procedures. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, chap. 28, pp. 2015–2114. Elsevier, Amsterdam (2001)
117. Lifschitz, V., Morgenstern, L., Plaisted, D.A.: Knowledge representation and classical logic. In: F. van Harmelen, V. Lifschitz, B. Porter (eds.) Handbook of Knowledge Representation, vol. 1, pp. 3–88. Elsevier, Amsterdam (2008)
118. Loveland, D.W.: A simplified format for the model elimination procedure. J. ACM **16**(3), 349–363 (1969)
119. Loveland, D.W.: A unifying view of some linear Herbrand procedures. J. ACM **19**(2), 366–384 (1972)
120. Loveland, D.W.: Automated Theorem Proving: A Logical Basis. North-Holland, Amsterdam (1978)
121. Ludwig, M., Waldmann, U.: An extension of the Knuth-Bendix ordering with LPO-like properties. In: N. Dershowitz, A. Voronkov (eds.) Proceedings of 14th International Conference on Logic, Programming and Automated Reasoning (LPAR), *Lecture Notes in Artificial Intelligence*, vol. 4790, pp. 348–362. Springer, Berlin (2007)
122. Lusk, E., McCune, W.W., Overbeek, R.: ITP at Argonne National Laboratory. In: J. Siekmann (ed.) Proceedings of 8th International Conference on Automated Deduction (CADE), *Lecture Notes in Computer Science*, vol. 230, pp. 697–698. Springer, Berlin (1986)
123. Lynch, C.A.: Constructing Bachmair-Ganzinger models. In: A. Voronkov, C. Weidenbach (eds.) Programming Logics: Essays in Memory of Harald Ganzinger, *Lecture Notes in Computer Science*, vol. 7797, pp. 285–301. Springer, Berlin (2013)
124. Martin, U.: How to choose the weights in the Knuth-Bendix ordering. In: P. Lescanne (ed.) Proceedings of 2nd International Conference on Rewriting Techniques and Applications (RTA), *Lecture Notes in Computer Science*, vol. 256, pp. 42–53. Springer, Berlin (1987)
125. McCharen, J., Overbeek, R., Wos, L.: Problems and experiments for and with automated theorem-proving programs. IEEE Trans. on Computers **C-25**(8), 773–782 (1976)
126. McCune, W.W.: Prover9 and Mace4. See http://www.cs.unm.edu/ mccune/prover9/
127. McCune, W.W.: Otter 3.3 reference manual. Tech. Rep. ANL/MSC-TM-263, Mathematics and Computer Science Division, Argonne National Laboratory (2003)
128. McCune, W.W., Wos, L.: Otter – the CADE-13 competition incarnations. J. Autom. Reason. **18**(2), 211–220 (1997)
129. Meredith, C.A.: The dependence of an axiom of Łukasiewicz. Trans. Am. Math. Soc. **87**, 54–54 (1958)
130. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. 1, chap. 7, pp. 371–443. Elsevier, Amsterdam (2001)
131. Nummelin, V., Bentkamp, A., Tourret, S., Vukmirović, P.: Superposition with first-class Booleans and inprocessing clausification. In: A. Platzer, G. Sutcliffe (eds.) Proceedings of 28th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 12699, pp. 378–395. Springer, Berlin (2021)
132. van Orman Quine, W.: A way to simplify truth functions. Am. Math. Mon. **62**, 627–631 (1955)
133. Peterson, G.E.: A technique for establishing completeness results in theorem proving with equality. SIAM J. Comput. **12**(1), 82–100 (1983)

134. Peterson, G.E., Stickel, M.E.: Complete sets of reductions for some equational theories. J. ACM **28**(2), 233–264 (1981)
135. Plaisted, D.A.: Abstraction using generalization functions. In: J. Siekmann (ed.) Proceedings of 8th International Conference on Automated Deduction (CADE), *Lecture Notes in Computer Science*, vol. 230, pp. 365–376. Springer, Berlin (1986)
136. Plaisted, D.A.: Mechanical theorem proving. In: R.B. Banerji (ed.) Formal Techniques in Artificial Intelligence, pp. 269–320. Elsevier, Amsterdam (1990)
137. Plaisted, D.A.: Equational reasoning and term rewriting systems. In: D.M. Gabbay, C.J. Hogger, J.A. Robinson (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, vol. I: Logical Foundations, pp. 273–364. Oxford University Press, Oxford (1993)
138. Plaisted, D.A.: The search efficiency of theorem proving strategies. In: A. Bundy (ed.) Proceedings of 12th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 814, pp. 57–71. Springer, Berlin (1994). Full version: Technical Report MPI I-94-233
139. Plaisted, D.A.: Automated theorem proving. Wiley Interdisciplinary Reviews: Cognitive Science **5**(2), 115–128 (2014)
140. Plaisted, D.A., Lee, S.J.: Inference by clause linking. In: Z.W. Ras, M. Zemankova (eds.) Intelligent Systems: State of the Art and Future Directions, Artificial Intelligence. Ellis Horwood (1990). Long version available as TR90-022, CS Dept., UNC Chapel Hill, `http://www.cs.unc.edu/techreports/90-022.pdf`
141. Plaisted, D.A., Zhu, Y.: The Efficiency of Theorem Proving Strategies. Friedr. Vieweg & Sohn, Braunschweig (1997)
142. Plaisted, D.A., Zhu, Y.: Ordered semantic hyper linking. J. Autom. Reason. **25**, 167–217 (2000)
143. Prawitz, D.: An improved proof procedure. Theoria **26**, 102–139 (1960)
144. Reger, G., Suda, M., Voronkov, A.: Playing with AVATAR. In: A.P. Felty, A. Middeldorp (eds.) Proceedings of 25th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 9195, pp. 399–415. Springer, Berlin (2015)
145. Reif, W., Schellhorn, G.: Theorem proving in large theories. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction - A Basis for Applications, *Applied Logic Series*, vol. III: Applications, chap. 9, pp. 225–241. Kluwer Academic Publishers, Dordrecht (1998)
146. Riazanov, A.: Implementing an efficient theorem prover. Ph.D. thesis, Department of Computer Science, The University of Manchester (2003)
147. Robinson, G.A., Wos, L.: Paramodulation and theorem-proving in first-order theories with equality. In: D. Michie, B. Meltzer (eds.) Machine Intelligence, vol. 4, pp. 135–150. Edinburgh University Press, Edinburgh (1969)
148. Robinson, J.A.: Automatic deduction with hyper-resolution. Int. J. Comput. Math. **1**, 227–234 (1965)
149. Robinson, J.A.: A machine oriented logic based on the resolution principle. J. ACM **12**(1), 23–41 (1965)
150. Robinson, J.A.: A review of automatic theorem-proving. In: Proceedings of Symp. Appl. Math., vol. 19. AMS (1967)
151. Robinson, J.A.: Formal and informal proofs. In: R.S. Boyer (ed.) Automated Reasoning: Essays in Honor of Woody Bledsoe, Automated Reasoning Series, pp. 267–282. Kluwer Academic Publishers, Dordrecht (1991)
152. Rodríguez, A.J., Torrens, A., Verdú, V.: Łukasiewicz logic and Wajsberg algebras. Bull. Polish Acad. Sci., Sect. Logic **19**(2), 51–55 (1990)
153. Rose, A., Rosser, J.B.: Fragments of many-valued statement calculi. Trans. Am. Math. Soc. **87**, 1–53 (1958)
154. Rusinowitch, M.: Démonstration automatique par des techniques de réécriture. Ph.D. thesis, Université de Nancy 1 (1987). Published in the series Collection Science Informatique, InterEdition, Paris, France, 1989
155. Rusinowitch, M.: Theorem-proving with resolution and superposition. J. Symb. Comput. **11**(1–2), 21–50 (1991)
156. Schlichtkrull, A., Banchette, J., Traytel, D., Waldmann, U.: Formalizing Bachmair's and Ganzinger's ordered resolution prover. J. Autom. Reason. **64**, 1169–1195 (1991)

157. Schulz, S.: E – A brainiac theorem prover. AI Commun. **15**(2–3), 111–126 (2002)
158. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In: M.P. Bonacina, M.E. Stickel (eds.) Automated Reasoning and Mathematics: Essays in Memory of William W. McCune, *Lecture Notes in Artificial Intelligence*, vol. 7788, pp. 45–67. Springer, Berlin (2013)
159. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: P. Fontaine (ed.) Proceedings of 27th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 11716, pp. 495–507. Springer, Berlin (2019)
160. Schulz, S., Möhrmann, M.: Performance of clause selection heuristics for saturation-based theorem proving. In: N. Olivetti, A. Tiwari (eds.) Proceedings of 8th International Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 9706, pp. 330–345. Springer, Berlin (2016)
161. Slagle, J.R.: Automatic theorem proving with renamable and semantic resolution. J. ACM **14**(4), 687–697 (1967)
162. Slaney, J., Lusk, E., McCune, W.W.: SCOTT: Semantically constrained Otter. In: A. Bundy (ed.) Proceedings of 12th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 814, pp. 764–768. Springer, Berlin (1994)
163. Socher-Ambrosius, R., Johann, P.: Deduction systems. Springer, Berlin (1997)
164. Sofronie-Stokkermans, V.: Hierarchic reasoning in local theory extensions. In: R. Nieuwenhuis (ed.) Proceedings of 20th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 3632, pp. 219–234. Springer, Berlin (2005)
165. Sofronie-Stokkermans, V., Ihlemann, C.: Automated reasoning in some local extensions of ordered structures. J. Multiple-Valued Log. Soft Comput. **13**(4–6), 397–414 (2007)
166. Stickel, M.E.: Automated deduction by theory resolution. J. Autom. Reason. **1**, 333–355 (1985)
167. Stickel, M.E.: PTTP and linked inference. In: R.S. Boyer (ed.) Automated Reasoning: Essays in Honor of Woody Bledsoe, Automated Reasoning Series, pp. 283–296. Kluwer Academic Publishers, Dordrecht (1991)
168. Stuber, J.: Superposition theorem proving for Abelian groups represented as integer modules. Theoret. Comput. Sci. **208**(1–2), 149–177 (1998)
169. Sutcliffe, G.: A heterogeneous parallel deduction system. In: R. Hasegawa, M.E. Stickel (eds.) Proceedings of FGCS Workshop on Automated Deduction: Logic Programming and Parallel Computing Approaches, pp. 5–13 (1992)
170. Tammet, T.: GKC: a reasoning system for large knowledge bases. In: P. Fontaine (ed.) Proceedings of 27th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 11716, pp. 538–549. Springer, Berlin (2019)
171. Tarski, A., Łukasiewicz, J.: Investigations into the sentential calculus. In: A. Tarski (ed.) Logic, Semantics and Metamathematics, Lecture Notes in Artificial Intelligence, chap. 4, pp. 38–56. Clarendon Press, Oxford (1956)
172. Urban, J., Vyskocil, J.: Theorem proving in large formal mathematics as an emerging AI field. In: M.P. Bonacina, M.E. Stickel (eds.) Automated Reasoning and Mathematics: Essays in Memory of William W. McCune, *Lecture Notes in Artificial Intelligence*, vol. 7788, pp. 240–257. Springer, Berlin (2013)
173. Veenker, G.: Beweisalgorithmen für die prädikatenlogik. Computing **2**(3), 263–283 (1967)
174. Vukmirović, P., Bentkamp, A., Blanchette, J., Cruanes, S., Nummelin, V., Tourret, S.: Making higher-order superposition work. J. Autom. Reason. **in press** (2022). Available at http://doi.org/10.1007/s10817-021-09613-z
175. Vukmirović, P., Blanchette, J., Cruanes, S., Schulz, S.: Extending a brainiac prover to lambda-free higher-order logic. In: T. Vojnar, L. Zhang (eds.) Proceedings of 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), *Lecture Notes in Computer Science*, vol. 11427, pp. 192–210. Springer, Berlin (2019)
176. Vukmirović, P., Nummelin, V.: Boolean reasoning in a higher-order superposition prover. In: P. Fontaine, K. Korovin, I.S. Kotsireas, P. Rümmer, S. Tourret (eds.) Proceedings of 7th Workshop on Practical Aspects in Automated Reasoning (PAAR), *CEUR Workshop Proceedings*, vol. 2752, pp. 148–166 (2020)

177. Waldmann, U.: Superposition for divisible torsion-free Abelian groups. In: C. Kirchner, H. Kirchner (eds.) Proceedings of 15th International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 1421, pp. 144–159. Springer, Berlin (1998)
178. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: N. Peltier, V. Sofronie-Stokkermans (eds.) Proceedings of 10th International Joint Conference on Automated Reasoning (IJCAR), *Lecture Notes in Artificial Intelligence*, vol. 12166, pp. 316–334. Springer, Berlin (2020)
179. Warren, D.H.D.: An abstract Prolog instruction set. Tech. Rep. 309, SRI International (1983)
180. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: R.A. Schmidt (ed.) Proceedings of 22nd International Conference on Automated Deduction (CADE), *Lecture Notes in Artificial Intelligence*, vol. 5663, pp. 140–145. Springer, Berlin (2009)
181. Weidenbach, C., Wischnewski, P.: Contextual rewriting in SPASS. In: B. Konev, R.A. Schmidt, S. Schulz (eds.) Proceedings of 1st Workshop on Practical Aspects in Automated Reasoning (PAAR), *CEUR Workshop Proceedings*, vol. 373, pp. 115–124 (2008)
182. Wos, L.: New challenge problem in sentential calculus. Newsletter of the Association for Automated Reasoning, No. 16, pages 7–8 (1990). Available at http://aarinc.org/Newsletters/016-1990-11.pdf
183. Wos, L.: The legacy of a great researcher. In: M.P. Bonacina, M.E. Stickel (eds.) Automated Reasoning and Mathematics: Essays in Memory of William W. McCune, pp. 1–14. Springer, Berlin (2013)
184. Wos, L., Carson, D.F., Robinson, G.A.: The unit preference strategy in theorem proving. In: Proceedings of AFIPS Fall Joint Computer Conference, pp. 615–621. Spartan Books, New York (1964)
185. Wos, L., Henschen, L.: Automated theorem proving 1965-1970. In: J. Siekmann, G. Wrightson (eds.) Automation of Reasoning 2 – Classical Papers on Computational Logic 1967-1970, pp. 1–24. Springer, Berlin (1983)
186. Wos, L., Overbeek, R., Lusk, E.: Subsumption, a sometimes undervalued procedure. In: J.L. Lassez, G. Plotkin (eds.) Computational Logic – Essays in Honor of Alan Robinson, pp. 3–40. MIT Press, Cambridge (1991)
187. Wos, L., Robinson, G.A., Carson, D.F.: Efficiency and completeness of the set of support strategy in theorem proving. J. ACM **12**, 536–541 (1965)
188. Wos, L., Robinson, G.A., Carson, D.F., Shalla, L.: The concept of demodulation in theorem proving. J. ACM **14**(4), 698–709 (1967)
189. Zhang, H.: Contextual rewriting in automated reasoning. Fundam. Inf. **24**(1–2), 107–123 (1995)