A taxonomy of theorem-proving strategies

Maria Paola Bonacina [⋆]

Department of Computer Science
The University of Iowa
Iowa City, IA 52242-1419, USA
E-mail: bonacina@cs.uiowa.edu

Abstract. This article presents a taxonomy of strategies for fully-automated general-purpose first-order theorem proving. It covers forward-reasoning ordering-based strategies and backward-reasoning subgoal-reduction strategies, which do not appear together often. Unlike traditional presentations that emphasize logical inferences, this classification strives to give equal weight to the inference and search components of theorem proving, which are equally important in practice. For this purpose, a formal notion of search plan is given and shown to apply to all classes of strategies. For each class, the form of derivation is specified, and it is shown how inference system and search plan cooperate to generate it.

1 Introduction

The objective of fully-automated theorem proving is to have computer programs that, given a collection of assumptions H and a conjecture φ, determine whether φ is a logical consequence of H (in symbols, whether $H \models \varphi$). Assumptions and conjecture need to be written in some language; let Θ be a first-order signature, providing symbols for variables, constants, functions and predicates; for now, let $\mathcal{L}_\Theta$ denote ambiguously a Θ-language of sentences, or clauses, or equations, depending on the problem, and $\mathcal{P}(\mathcal{L}_\Theta)$ its powerset. The theorem-proving approach to the problem is to try to show that there is a *proof* of φ from H (in symbols, $H \vdash \varphi$, or, refutationally, that there is a proof of a contradiction from $H \cup \{\neg\varphi\}$, i.e., $H \cup \{\neg\varphi\} \vdash \bot$), or disprove φ by exhibiting a *model* of $H \cup \{\neg\varphi\}$. A proof is a sequence[1] of statements in $\mathcal{L}_\Theta$ logically connected by applications of *inference rules*, that is, rules in the form:

$$f\colon\ \frac{\psi_1 \ldots \psi_n}{\psi}$$

which says that the inference rule named f, if given premises in the form $\psi_1 \ldots \psi_n$, infers a consequence in the form ψ. For example, *binary clausal resolution* is defined as

[⋆] Supported in part by the National Science Foundation with grant CCR-97-01508.

[1] Proofs are read sequentially also when they are presented as trees or graphs.

$$\frac{L_1 \vee D, L_2 \vee C}{(C \vee D)\sigma}\, L_1\sigma = \neg L_2\sigma \quad (\sigma\ \mathit{most\ general\ unifier})$$

where L_1 and L_2 are literals and C and D are disjunctions of literals. As another example, the *T-rule for conjunction* in analytic tableaux[2] with signed formulae is defined as

$$\frac{T A \wedge B}{T A,\ T B}$$

where A and B are sentences and T is the sign for true.

In order to make sure that once a proof has been obtained, it really means that $H \models \varphi$, one needs to check that the inference rules are *sound*: the generic inference rule f above is sound if for all interpretations $\mathcal{I}$ of Θ, that is, for all ways to give meaning to the predicates, functions and constants in Θ, if $\mathcal{I}$ satisfies $\{\psi_1 \ldots \psi_n\}$, then $\mathcal{I}$ satisfies ψ. Symmetrically, one wishes that whenever $H \models \varphi$, the inference rules are sufficiently strong to ensure that there is a proof of φ from H: a set of inference rules – or *inference system* – with this property is said to be *complete*. Since most inference systems for mechanical theorem proving work refutationally rather than directly, the requirement is that whenever $H \models \varphi$, there is a proof of $\bot$ from $H \cup \{\neg\varphi\}$, or the system is *refutationally complete*.

The availability of a sound and complete inference system guarantees the existence of a proof. It remains the problem of how to compute one. The initial state of a proof attempt contains H and $\neg\varphi$, and the application of an inference rule to this state produces a new state. Thus, the problem can be seen in the terms, familiar to Artificial Intelligence, of a *search problem*, with the inference rules as *transformation rules*, or *production rules*, *states* containing partial proofs, *successful states* containing complete proofs, and a *search plan* – or *computation rule* in terminology influenced by logic programming – controlling the search.

Let *States* denote, ambiguously, for now, the set of all possible states. Given a set of inference rules I, a search plan Σ is made of at least three components:

- A *rule-selecting function* $\zeta\colon \mathit{States}^* \rightarrow I$, which selects the next rule to be applied based on the history of the search so far;
- A *premise-selecting function* $\xi\colon \mathit{States}^* \rightarrow \mathcal{P}(\mathcal{L}_\Theta)$, which selects the elements of the current state the inference rule should be applied to;
- A *termination-detecting function* $\omega\colon \mathit{States} \rightarrow \mathit{Bool}$, which returns *true* if the given state is successful, false otherwise.

If the current state is not successful, ζ selects rule f and ξ selects premises $\psi_1 \ldots \psi_n$, the next step will consist of inferring ψ from $\psi_1 \ldots \psi_n$. The sequence of states thus generated forms the *derivation* by I controlled by Σ from the given input. A derivation is *successful* if it terminates in a successful state.

[2] Analytic tableaux are a form of semantic tableaux: see [126] for an introduction.

It is important to appreciate that given an initial state with H and $\neg\varphi$, there are many derivations that an inference system I can generate from the initial state. In this sense, an inference system is *non-deterministic*. If I is coupled with a search plan Σ, there is one and only one derivation generated by I and Σ from the initial state. The combination of inference system and search plan forms a deterministic procedure called a *theorem-proving strategy*. While the inference system is required to be sound and refutationally complete, a search plan is expected to be *fair*: if there are proofs, or, equivalently, if there are successful states in the search space, one will be generated eventually.

In summary, a *theorem-proving problem* has the form $S = H \cup \{\neg\varphi\}$, where φ is called the *target theorem* and $\neg\varphi$ is called the *goal*; a *theorem-proving strategy* $\mathcal{C}$ is specified by an inference system and a search plan, $\mathcal{C} = \langle I, \Sigma\rangle$. If I is refutationally complete, whenever $H \models \varphi$, there exist proofs $H \cup \{\neg\varphi\} \vdash_I \bot$, and if Σ is fair, the unique derivation driven by Σ will generate one of these proofs.

There are many ways to classify theorem-proving strategies. From a proof-theoretical point of view, one may question whether the strategy is *analytic* (i.e., it only generates formulae that are subformulae of $H \supset \varphi$) or *generative* (i.e., not analytic). From the point of view of the language and its expressive power, one may be interested in whether the strategy works with equations, clauses, or sentences. From the point of view of the logic and its applicability, one may consider whether the strategy works for propositional logic, Horn logic, or first-order logic. The point of view of this taxonomy is to give a classification of strategies based on *how they search*.

A classification key for this purpose is to observe whether the strategy works from the assumptions – called *forward reasoning*, *forward chaining* or *bottom-up* reasoning – or from the goal – called *backward reasoning*, *backward chaining* or *top-down* reasoning. Since finding a proof generally requires some combination of the two, *forward-reasoning strategies* are strategies that work primarily, not exclusively, by forward reasoning, and *backward-reasoning strategies* are defined dually. This criterion alone, however, may not be sufficient. First, different strategies may not have the same notion of what is the goal: for instance, in resolution, $\neg\varphi$ is the goal, but in analytic tableaux one may consider the whole $\neg(H \supset \varphi)$ as the goal. Second, the same feature of a strategy can be used for either type of reasoning. For instance, in a strategy with set of support, one can get a backward-reasoning behaviour by putting in the set of support the goal clauses (i.e., those originated from the transformation of the goal into clausal form), and a forward-reasoning behaviour by putting in the set of support clauses originated from the transformation of the assumptions. Therefore, the distinction between forward and backward reasoning will be used in the following, but it will not be the only key.

The primary key will be to distinguish between those strategies that work on a set of objects (e.g., clauses) and develop implicitly many proof attempts, and those strategies that work on one object at a time (e.g., a goal clause, or a tableau) and develop one proof attempt at a time, backtracking when the current proof attempt cannot be completed into a proof. The strategies of the first type, on the other hand, never backtrack, because whatever they do may further one of the proof attempts. While other names may be chosen to emphasize other features, strategies in the first group will be called here *ordering-based strategies*, to emphasize that exactly because they work with a set of objects, they can use a *well-founded ordering* to order them, and possibly *delete* objects that are greater than and entailed by others. Thus, these strategies work by generating objects, *expanding* the set, and deleting objects, *contracting* the set. Since the set typically grow very large, they may employ *indexing techniques* to retrieve objects, and *eager-contraction* search plans to control the growth. Ordering-based strategies with an eager-contraction search plan are called *contraction-based strategies*. The strategies resulting from the merging of the resolution-paramodulation paradigm with the term-rewriting and Knuth-Bendix paradigm, as well as strategies based on generating instances instead of resolvents, belong to this class[3]. Theorem provers based on these strategies include Otter [97], RRL [76], Reveal [3], SNARK [134], EQP [98], Barcelona [104], CLIN-S [47], SPASS [142], Gandalf [137], OSHL [114], and daTac [138].

The strategies of the second type will be called *subgoal-reduction strategies*, because if one considers the single object they work on as the goal, each step consists in reducing the goal to subgoals. Since they do not generate a set of objects, subgoal-reduction strategies do not use an ordering to sort it, neither can they use an object to delete another one. Because they need backtracking, a typical choice of search plan is *depth-first search with iterative deepening*. Tableaux-based strategies, model elimination, linear resolution, and problem reduction format methods belong to this class. Theorem provers implementing these strategies include Setheo [87, 64], METEOR [4], Protein [17], *TAP* [23, 22] and Tatzelwurm [29]. More provers of both types can be found in the system descriptions in the CADE proceedings [123, 92, 135, 77, 41, 101, 100] and in [136].

Note that the notion of goal in "subgoal-reduction" does not necessarily coincide with the notion of goal based on the interpretation of the problem. For instance, model elimination can start with any input clause as the first goal, although it is natural to start with a goal clause. This, together with the above observation about strategies with set of support, which are ordering-based strategies, shows that it is not necessarily the case that ordering-based strategies do forward reasoning, and subgoal-reduction strategies do backward reasoning. However, most ordering-based strategies are forward-reasoning strategies, and most subgoal-reduction strategies are backward-reasoning strategies. Similarly, it is not necessarily the case that subgoal-reduction strategies work with tableaux and ordering-based strategies work with clauses, although this is true for many strategies. For instance, linear resolution strategies are subgoal-reduction strategies that work with clauses, and the disjunctive positive model elimination with subsumption of [16] is an example of an ordering-based strategy that works with

[3] E.g., a strategy that features only resolution is an ordering-based strategy with an empty ordering.

tableaux. Thus, the essential criterion to separate the strategies is the nature of the search. Table 1 summarizes these points.

	Ordering-based	Subgoal-reduction
data	set of objects	one goal-object at a time
proof attempts built	many implicitly	one at a time
backtracking	No	Yes
contraction	Yes	No

Table 1. Two main classes of strategies

The following sections cover first ordering-based strategies and their subclasses, and then subgoal-reduction strategies and their subclasses.

1.1 Remarks and further reading

Classical books in theorem proving are Chang and Lee [45], Loveland [91] and Bibel [24], while recent books include [25, 85, 128]. Wos et al. [145] emphasize experimentation with theorem provers. Books in logic useful for theorem proving include Smullyan [126], Gallier [63], Ramsay [115], and Fitting [60], while a classical reference for search in AI is Pearl [106].

Collections of research articles in theorem proving and related topics include [122], which makes early classical papers (1957-1970) available, and [83, 40, 62], while [28] emphasizes applications. A major forum for the presentation of results in theorem proving is CADE – the International Conference on Automated Deduction: recent issues are [123, 92, 135, 77, 41, 101, 100].

The inference+search paradigm may be as old as theorem proving itself; Kowalski emphasized search in theorem proving in [81]. The formalization of the inference+search paradigm and the taxonomy of strategies in this paper organize, improve and extend elements appeared in [36, 37, 35, 38, 39, 32]. This paper considers only sequential strategies: an extension to parallel and distributed strategies, continuing the work begun in [35], will be a subject of future work.

2 Ordering-based strategies

Since most ordering-based strategies work with clauses, in this section $\mathcal{L}_\Theta$ is the language of clauses on signature Θ and $\bot$ is the empty clause $\Box$. If $H \cup \{\neg\varphi\}$ is not already in clausal form, each element in $H \cup \{\neg\varphi\}$ is transformed into a set of clauses, whose union S is the clausal form of the theorem-proving problem (e.g., [45] for this transformation). The goal $\neg\varphi$ is considered as an additional assumption, and most ordering-based strategies do not distinguish the clauses coming from $\neg\varphi$ from those coming from H.

2.1 Inference systems for ordering-based strategies

Inference rules for ordering-based strategies operate on *sets*:

$$f\colon \frac{S}{S'}$$

where S and S' are sets of clauses. For instance, binary resolution is written

$$\frac{S \cup \{L_1 \vee D, L_2 \vee C\}}{S \cup \{L_1 \vee D, L_2 \vee C, (C \vee D)\sigma\}} \; L_1\sigma = \neg L_2\sigma \quad (\sigma \textit{ most general unifier})$$

Exactly because generated data are kept, it is possible to use them to establish that other generated data are not needed. What is not needed is determined by a *well-founded ordering* on clauses $\succ$: intuitively, if a clause φ is entailed by one or more smaller clauses, φ can be deleted. Thus, ordering-based strategies have two types of inference rules:

– *Expansion inference rules*, that generate and add clauses:

$$f\colon \frac{S}{S'} \; S \subset S'$$

where the condition $S \subset S'$ tells that something has been added, and implies $S \prec_{mul} S'$, where $\succ_{mul}$ is the multiset extension of $\succ$.

– *Contraction inference rules*, that delete clauses or replace them by smaller ones:

$$f\colon \frac{S}{S'} \; S \not\subseteq S'$$

where the condition $S \not\subseteq S'$ tells that something has been deleted; furthermore, $S' \prec_{mul} S$ needs to hold.

With this formulation of inference rules, the *soundness* requirement is written $Th(S') \subseteq Th(S)$, where $Th(S) = \{\varphi \mid S \models \varphi\}$, which means that whatever is added is a logical consequence of what was given. Since there are inference rules that delete elements, one needs also the dual property of *monotonicity*: $Th(S) \subseteq Th(S')$, which means that all theorems are preserved.

The ordering $\succ$ is fundamental for these strategies. In addition to being well-founded, it needs to be *monotonic* with respect to the term structure (i.e., $s \succ t$ implies $c[s] \succ c[t]$ for all terms s, t, c) and *stable* with respect to substitutions (i.e., $s \succ t$ implies $s\sigma \succ t\sigma$ for all substitutions σ). An ordering with these three properties is a *reduction ordering*. A *simplification ordering* has monotonicity, stability and the *subterm property* (i.e., a term is greater than any of its proper subterms), which together imply well-foundedness [52]. *Complete simplification orderings* – introduced in [69] – are also total on ground terms. Since complete simplification orderings are used most frequently, let $\succ$ be such an ordering. Once an ordering on terms and literals is given, it can be extended to equations and clauses in standard ways based on the multiset extension (e.g., [70, 12]).

In addition to *resolution*[4] [118], *paramodulation* (on clauses) [116, 70], and *superposition* (on rewrite rules [79] or equations [69, 10]) are expansion inference rules. Contraction rules include *tautology deletion*, *purity deletion* [50, 38], *subsumption* [118], *clausal simplification* (also called *unit simplification* or *unit deletion*, because a unit clause simplifies another clause), *functional subsumption* [69], and *simplification* [144, 79, 69, 10, 119]:

$$\frac{S \cup \{p \simeq q, L \vee D\}}{S \cup \{p \simeq q, L[q\sigma]_u \vee D\}} \; L|u = p\sigma \;\wedge\; L \succ L[q\sigma]_u$$

where $L|u$ denotes the subterm of L at position u and $L[q\sigma]_u$ denotes the literal obtained by replacing $L|u$ by $q\sigma$.

In order to describe unambiguously how a strategy generates a derivation, inference rules can be characterized as functions $f\colon \mathcal{P}(\mathcal{L}_\Theta) \to \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$, which take as argument a set of premises, and return a pair of sets, a set of generated clauses and a set of clauses to be deleted[5]. If f does not apply to a set X, $f(X) = (\emptyset, \emptyset)$. Let π_1 and π_2 be the projection functions $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$; then expansion and contraction rules (with respect to $\succ$) can be described in this form as follows:

- An inference rule f is an *expansion inference rule* if for all X, $\pi_2(f(X)) = \emptyset$.
- An inference rule f is a *contraction inference rule* if either $\pi_1(f(X)) = \pi_2(f(X)) = \emptyset$, or $\pi_2(f(X)) \neq \emptyset$ and $X - \pi_2(f(X)) \cup \pi_1(f(X)) \prec_{mul} X$.

Most inference rules generate and/or delete one clause at each application, so that $\pi_1(f(X))$ and $\pi_2(f(X))$ are singletons. In the following the same notation is used ambiguously to denote both the set and the single element it contains. The condition $X \models \pi_1(f(X))$ implies soundness, while $X - \pi_2(f(X)) \cup \pi_1(f(X)) \models \pi_2(f(X))$ implies monotonicity.

Contraction does more than deleting existing clauses. By deleting existing clauses, it also prevents the strategy from using those clauses to generate others. In order to understand better this deeper effect of contraction, the notion of *redundancy*, whereby clauses deleted by contraction are redundant, was developed (e.g., [119, 127, 12, 36]). The advantages of working with a notion of redundancy are several. First, not only clauses deleted by contraction are redundant, but also clauses that can be generated only by redundant clauses. Second, redundancy can be generalized from clauses to inferences, observing that an inference step that uses a redundant clause without deleting it is redundant. Third, restrictions to expansion inference rules such as *ordered inference rules* (e.g., [70, 12, 105]), *critical pair criteria* (e.g., [9, 73]), and *basic inferences* (e.g., [14]), can be explained as preventing the expansion rules from generating redundant clauses. Thus, the design of contraction rules to delete redundant clauses and the design of refinements of expansion rules to prevent the generation of redundant clauses are two sides of the same effort to contain redundancy.

[4] Here and in the rest of the paper resolution include factoring.

[5] In the notation, $\mathcal{P}(\ )$ means powerset, and if $X = \{\psi_1 \ldots \psi_n\}$ we may write $f(X)$ or $f(\psi_1, \ldots, \psi_n)$ instead of $f(\{\psi_1, \ldots, \psi_n\})$.

Redundancy depends on the well-founded ordering on clauses: like selecting different orderings may yield different contraction rules, selecting different orderings may yield different *redundancy criteria* [11]. A *redundancy criterion* is a mapping R on sets of clauses, such that $R(S)$ is the set of clauses that are redundant with respect to S according to R, and the following properties are satisfied: (1) $S - R(S) \models R(S)$, (2) if $S \subseteq S'$, then $R(S) \subseteq R(S')$, (3) if $(S' - S) \subseteq R(S')$, then $R(S') \subseteq R(S)$ [11].

A redundancy criterion R and a set of contraction rules I *correspond* if they are based on the same well-founded ordering $\succ$, and for all sets of clauses S:

- Whatever is deleted by I is redundant according to R: for all $f \in I$ and $X \in \mathcal{P}(S)$, $\pi_2(f(X)) \subseteq R(X - \pi_2(f(X)) \cup \pi_1(f(X)))$.
- If a clause in S is redundant with respect to S, I can delete it with no need to add other clauses (to make it redundant): for all $\varphi \in S \cap R(S - \{\varphi\})$, there exist $f \in I$ and $X \in \mathcal{P}(S)$, such that $\pi_1(f(X)) = \emptyset$ and $\pi_2(f(X)) = \{\varphi\}$.

Given an inference system I, I_E denotes the subset of expansion rules, and I_R denotes the subset of contraction rules, with R the corresponding redundancy criterion. The first property of redundancy criteria serves the purpose of implying that if $S \vdash_{I_R} S'$ then $Th(S) \subseteq Th(S')$, and it is equivalent in this respect to the condition $X - \pi_2(f(X)) \cup \pi_1(f(X)) \models \pi_2(f(X))$ formulated for all contraction rules in I_R. The second and third properties of redundancy criteria guarantee that $S \vdash_I S'$ implies $R(S) \subseteq R(S')$.

Redundancy control is fundamental for ordering-based strategies, exactly because they work by generating and keeping consequences. In first-order theorem proving, the search space of consequences that can be generated from a given $H \cup \{\neg\varphi\}$ is typically infinite: a strategy that searches this space by generating clauses without the possibility of deleting/avoiding redundant ones is not practical. In summary, ordering-based strategies need redundancy control, and at the same time they make it possible (e.g., one cannot use clauses to delete other clauses if clauses are not kept in the first place).

Remarks and further reading Much research on ordering-based strategies originated from works in *rewriting*, *orderings*, and *Knuth-Bendix completion*: a comprehensive treatment of this area can be found in [55], while theorem-proving oriented surveys include [111, 57, 108].

The essential role of orderings for these strategies can be appreciated by considering the history of *simplification*, *paramodulation* and *ordered resolution*. Simplification was introduced as *demodulation* in [144], but without a well-founded ordering to guarantee termination one had to impose a maximum number of rewriting steps. Paramodulation was introduced in [116], but the completeness proof required the functionally reflexive axioms and paramodulating into variables. The conjecture of [116] that these requirements are not necessary was proved in [107], but only postulating an ordering with a very rare property (order-isomorphism to ω), and finally in [70], with a complete simplification ordering as the fundamental ingredient. Ordered resolution is almost as old as resolution itself: the early research is summarized in [70], where prior references can be found. The main difference between the early formulations and the contemporary ones is the ordering on literals: the former treated clauses as lists, so that the ordering was arbitrary, whereas the latter use a complete simplification ordering. Further developments are treated in [13]; the references to previous work on *basic narrowing* and *basic paramodulation* can be found in [14].

While most ordering-based strategies work on clauses, the strategies that work with rewrite rules or equations in the *Boolean ring* [67] also belong to this class: their key characteristics include the uniqueness of normal forms in the Boolean ring representation, and simplification inferences that cannot be simulated by resolution and subsumption (see [102] for this comparison). These strategies began with the *N-strategy* of [67] and the *Gröbner basis method* of [74]. The strategy of [8] continued in the spirit of [74], while the N-strategy was extended to first-order logic with equality in [68], and to non-clausal input in [147]. Other developments can be found in [102], together with a comparison with clausal resolution methods continuing [56].

Other directions of growth for these inference systems have been *theory reasoning* and *constrained reasoning*. Theory resolution was pioneered in [131], while a recent overview of theory reasoning is available in Chapter 4 of [15]. In equational logic forms of theory reasoning may be obtained by replacing syntactic unification in the inference rules with semantic unification: two surveys of this large field are [72, 7]. Ordering-based strategies with constraints were studied in [78], where references to previous work can be found. A general treatment of constrained resolution was given in [42].

The RAMC method [44] and its successor EQMC [43] also work with constrained clauses. These methods combine searching for a refutation by generating consequences with searching for a model by generating *non-consequences*: if S were consistent, the fact that $S \not\models \psi$ means that there exists a model of S where ψ is false and $\neg\psi$ true, and therefore generating ψ is a step towards identifying such a model if it exists. The search for a refutation employs expansion rules, such as resolution, and contraction rules, such as subsumption, similar to other ordering-based strategies, while the search for a model employs dual *dis-inference rules* (e.g., dis-resolution, dis-subsumption), case analysis by splitting, generation of pure literals, and equational constraints to encode models.

2.2 Search plans for ordering-based strategies

Since ordering-based strategies work on sets of clauses, and multisets need to be used in order to apply the ordering, for these strategies *States* is the set of all multisets of $\mathcal{L}_\Theta$. The general notion of search plan given in Section 1 can be instantiated to a tuple $\Sigma = \langle \zeta, \xi_1, \xi_2, \omega \rangle$ where:

- $\xi_1\colon States^* \rightarrow \mathcal{L}_\Theta$ selects a *primary premise* from the current state: $\xi_1((S_0 \dots S_i)) = \psi_1 \in S_i$;
- $\zeta\colon States^* \times \mathcal{L}_\Theta \rightarrow I$ selects an inference rule, based on the history and the primary premise: $\zeta((S_0 \dots S_i), \psi_1) = f^n \in I$;

– $\xi_2 \colon States^* \times \mathcal{L}_\Theta \times I \rightarrow \mathcal{P}(\mathcal{L}_\Theta)$ selects one or more *secondary premises*, depending on the arity of the inference rule that ζ selected: $\xi_2((S_0 \dots S_i), \psi_1, f^n) = \{\psi_2 \dots \psi_n\} \subseteq S_i$;
– $\omega \colon States \rightarrow Bool$ returns *true* if and only if the given state contains the empty clause.

For example, if ξ_1 selects a clause ψ, and ζ selects a binary expansion rule, ξ_2 selects a second clause; if ζ selects simplification (normalization), ξ_2 selects the simplifier(s) to reduce ψ.

Concrete search plans may fit in this pattern in various ways. Consider the *given-clause algorithm* of Otter [97]: it works with a list of clauses to be selected, called `sos` for historical reasons (the Set of Support strategy of [143]), and a list of clauses already selected, called `usable`, because these clauses can be used for inferences. It selects a given clause from `sos`, makes all expansion inferences between the given clause and the clauses in `usable`, process and appends to `sos` the non-trivial normal forms of all clauses thus generated, moves the given clause from `sos` to `usable`, and repeats. Then ξ_1 is the mechanism that selects the given clause, ζ represents the order of the expansion rules in the code of the prover, and ξ_2 is the mechanism that selects the other premises from `usable`.

In the default configuration of Otter, the next given clause is the shortest clause in `sos`. Thus, ξ_1 performs a *best-first search*, with the length of the clause as heuristic evaluation function. Changing the heuristic evaluation function amounts to modifying the component ξ_1 of the search plan. For instance, Otter has a parameter, called *pick-given-ratio*, which allows one to to add some breadth-first search: if the value of *pick-given-ratio* is k, ξ_1 selects the oldest clause in `sos` (instead of the shortest) every k choices. The selection of suitable premises from `usable` is done by an *indexing technique* [133, 95, 46]. Then ξ_2 is the abstraction of the indexing technique, and choosing a different indexing technique amounts to changing the component ξ_2 of the search plan.

The equational prover EQP [98], which solved the Robbins conjecture [99], features search plans based on the given-clause algorithm, and search plans based on another algorithm, called the *pair algorithm.* This search plan works on an index of all possible pairs of equations in the database: it selects a pair from the index, performs all expansion inferences between the equations in the pair, if at least one of them belongs to `sos`, and repeats. Then there is no ξ_1, and ξ_2 is the mechanism that selects the next pair.

For the control of contraction, one needs to distinguish between *forward contraction*, which normalizes a newly generated clause with respect to the existing clauses, and *backward contraction*, which applies the normal form of a newly generated clause to reduce the existing clauses. For forward contraction, ξ_1 returns the newly generated clause, say ψ, ζ corresponds to the order of the contraction rules in the code of the prover, and ξ_2 to the indexing mechanism that selects the simplifers that match ψ. For backward contraction, ξ_2 corresponds to the indexing mechanism that selects the clauses matched by ψ.

It is possible to abstract from the relation between selection of the inference rule and selection of the premises, and use $\zeta \colon States^* \rightarrow I$ and $\xi \colon States^* \times I \rightarrow$

$\mathcal{P}(\mathcal{L}_\Theta)$, with the condition that $\xi((S_0 \ldots S_i), f^n) = \{\psi_1 \ldots \psi_n\}$ if and only if $\xi_1((S_0 \ldots S_i)) = \psi_1$ and $\xi_2((S_0 \ldots S_i), \psi_1, f^n) = \{\psi_2 \ldots \psi_n\}$.

Given a theorem-proving problem S, an ordering-based strategy $\mathcal{C} = \langle I, \Sigma \rangle$, with $\Sigma =< \zeta, \xi, \omega >$, generates the *derivation*

$$S_0 \underset{\mathcal{C}}{\vdash} \ldots S_i \underset{\mathcal{C}}{\vdash} \ldots$$

where $S_0 = S$ and for all $i \geq 0$, if $\omega(S_i) = false$, $\zeta((S_0 \ldots S_i)) = f$, and $\xi((S_0 \ldots S_i), f) = X$, then $S_{i+1} = S_i \cup \pi_1(f(X)) - \pi_2(f(X))$. An equivalent characterization can be given using ξ_1 and ξ_2.

Remarks and further reading Most theorem-proving search plans exist only in the code of theorem provers, with specifications in natural language in manuals or system descriptions. A definition of search plan was given by Kowalski in [81]: it was defined directly on the search space of resolution, and therefore did not account for the generation of the derivation. The definition given here allows to fit in concrete search plans; it is a refinement of the definition in [37], improved in [39, 32], which was the first one to account for the derivation in the context of general expansion and contraction.

The distinction between forward and backward subsumption was made when several authors working towards proving the completeness of resolution with subsumption discovered that unrestricted backward subsumption of variants can violate completeness: the solution was to perform forward subsumption before backward subsumption; Kowalski summarized this early work in the introduction of [81]. The problem was better understood later, when it was clarified that the difficulty is that the subsumption ordering is not well-founded. The solution is to label the clauses based on generation time and use a well-founded ordering given by the lexicographic combination of the proper subsumption ordering and a well-founded ordering on the labels of clauses, so that newly generated variants are forward-subsumed before they can back-subsume; Loveland presents problem and solution on pages 207–208 of [91]. An analysis and solutions of problems of the same nature with subsumption in distributed derivations can be found in [34]. McCune distinguishes between forward and backward subsumption, and forward and backward demodulation in the code of Otter [97] and EQP [98]; the distinction was generalized to any kind of contraction in [35].

2.3 Search spaces and proofs by ordering-based strategies

The *search space* of a theorem-proving problem S contains all the clauses that can be derived from S by using the inference system I: the *closure* of S with respect to I is the set $S^*_I = \bigcup_{k \geq 0} I^k(S)$, where $I(S) = S \cup \{\varphi | \varphi \in \pi_1(f(X)),\ f \in I,\ X \subseteq S\}$, $I^0(S) = S$, and for $k \geq 1$, $I^k(S) = I(I^{k-1}(S))$ for $k \geq 1$.

This search space can be modelled as a *search graph*, a *hypergraph* $G(S^*_I) = (V, E, l, h)$, where the vertices in V represent the clauses in the closure S^*_I, and the hyperarcs in E represent the inferences. The hypergraph is decorated by an *arc-labelling function* $h: E \to I$ from hyperarcs to inference rules, and an injective *vertex-labelling function* $l: V \to \mathcal{L}_\Theta / \doteq$ from vertices to equivalence classes of clauses, where $\doteq$ is equivalence up to variable renaming. Thus, all variants of a clause are associated to a unique vertex. For simplicity, $l(v)$ denotes a clause, meaning a representative of a class of variants.

If $f^n(\{\varphi_1 \dots \varphi_n\}) = (\{\psi_1 \dots \psi_m\}, \{\gamma_1 \dots \gamma_p\})$ for $f^n \in I$, E contains a hyperarc $e = (v_1 \dots v_k; w_1 \dots w_p; u_1 \dots u_m)$ where $h(e) = f^n$ and:

– $v_1 \dots v_k$ are the vertices labelled by those premises that are not deleted, i.e., $l(v_j) = \varphi_j$ and $\varphi_j \notin \{\gamma_1 \dots \gamma_p\}$, $\forall j$, $1 \leq j \leq k$, where $k = n - p$,
– $w_1 \dots w_p$ are the vertices labelled by the deleted clauses, i.e., $l(w_j) = \gamma_j$, $\forall j$, $1 \leq j \leq p$, and
– $u_1 \dots u_m$ are the vertices labelled by the generated clauses, i.e., $l(u_j) = \psi_j$, $\forall j$, $1 \leq j \leq m$.

Vertices and their labels can be used interchangeably, and without loss of generality one can consider hyperarcs in the form $(v_1 \dots v_n; w; u)$, where at most one clause is added or deleted. For instance, a *resolution* arc has the form $(v_1, v_2; u)$, where u is a resolvent of v_1 and v_2; a *simplification* arc has the form $(v; w; u)$, where v reduces w to u; and a *normalization* arc has the form $(v_1 \dots v_n; w; u)$, where u is a normal form of w with respect to the simplifiers $v_1 \dots v_n$. Contraction inferences that purely delete clauses are represented as replacement by *true*, where *true* is a dummy clause, such that *true* $\prec \varphi$ for all φ, and a special vertex T in $G(S_I^*)$ is labelled by *true*. The application of this representation to more inference rules and several examples of inference steps can be found in [39].

$G(S_I^*) = (V, E, l, h)$ represents the static structure of the search space. The dynamics of the search during a derivation is described by *marking functions* for vertices and arcs. A *marked search-graph* (V, E, l, h, s, c) is enriched with

– A *vertex-marking function* $s: V \to Z$ from vertices to integers, such that

$$s(v) = \begin{cases} m & \text{if } m \text{ variants } (m > 0) \text{ of } l(v) \text{ are present,} \\ -1 & \text{if all variants of } l(v) \text{ have been deleted,} \\ 0 & \text{otherwise.} \end{cases}$$

– An *arc-marking function* $c: E \to Z^+$ from hyperarcs to non-negative integers, such that $c(e) = n$ if the inference of arc e has been executed n times.

The vertex-marking function represents the dynamic effect of contraction (if a clause is deleted, its marking becomes negative), while the arc-marking function represents the selections of steps done by the search plan.

It is then possible to represent the evolution of the search space during a derivation. First, a hyperarc $e = (v_1 \dots v_n; w; u) \in E$ is *enabled* if its premises are present: $s^k(v_j) > 0$ for $1 \leq j \leq n$ and $s^k(w) > 0$ ($s^k(w) > 1$ if $w \in \{v_1 \dots v_n\}$, e.g., for a variant subsumption arc (v, v, T)).

A derivation induces a *succession of vertex-marking functions* $\{s_i\}_{i \geq 0}$ and a *succession of arc-marking functions* $\{c_i\}_{i \geq 0}$ initialized as follows: for all $v \in V$,

$s_0(v) = 0$, and for all $a \in E$, $c_0(a) = 0$. Then, $\forall i \geq 0$, if at stage i the strategy executes an enabled hyperarc $e = (v_1 \dots v_n; w; u)$:

$$s_{i+1}(v) = \begin{cases} s_i(v) - 1 & \text{if } v = w \wedge s_i(v) > 1 \text{ (decrease marking of deleted clause)}, \\ -1 & \text{if } v = w \wedge s_i(v) = 1, \\ s_i(v) + 1 & \text{if } v = u \wedge s_i(v) \geq 0 \text{ (increase marking of generated clause)}, \\ 1 & \text{if } v = u \wedge s_i(v) = -1, \\ s_i(v) & \text{otherwise.} \end{cases}$$

$$c_{i+1}(a) = \begin{cases} c_i(a) + 1 & \text{if } a = e \text{ (increase marking of executed arc)}, \\ c_i(a) & \text{otherwise.} \end{cases}$$

The initialization $s_0(v) = 0$ for all vertices, including input clauses, assumes that also the steps of reading the input clauses are included in the derivation (e.g., modelled as expansion steps). Alternatively, one can start with $s_0(v) = 1$, if $\varphi = l(v)$ is in S_0, and $s_0(v) = 0$ otherwise.

Each state S_i has its associated search graph $G_i = (V, E, l, h, s_i, c_i)$, and S_i is exactly the multiset of clauses with positive marking in G_i. The subgraph containing only these clauses, $G_i^+ = (V^+, E^+, l, h, s_i, c_i)$, where $V^+ = \{v \mid v \in V,\ s_i(v) > 0\}$ and E^+ is the restriction of E to V^+, represents the *active part of the search space* at stage i. The subgraph of all the clauses with non-zero marking, $G_i^* = (V^*, E^*, l, h, s_i, c_i)$, where $V^* = \{v \mid v \in V,\ s_i(v) \neq 0\}$ and E^* is the restriction of E to V^*, represents the *generated search space* up to stage i. If the derivation halts at some stage k, G_k^* is the search space generated by the strategy during the entire derivation.

It is important to emphasize that neither G_k^* nor G_k^+ represent the *proof* computed by an ordering-based strategy. The notion of *ancestor-graph* of a clause clarifies this point. Given a search graph $G = (V, E, l, h)$, for all $v \in V$:

- If v has no incoming hyperarcs, the *ancestor-graph* of v is the graph made of v itself.
- If $e = (v_1 \dots v_n; v_{n+1}; v)$ is a hyperarc in E and $t_1 \dots t_{n+1}$ are ancestor-graphs of $v_1 \dots v_{n+1}$, the graph with root v connected by e to the subgraphs $t_1 \dots t_{n+1}$ is an *ancestor-graph* of v, denoted by the triple $(v; e; (t_1 \dots t_{n+1}))$.

An ancestor-graph of v represents a sequence of inferences, or a *generation-path*, that generates its associated clause φ from the input clauses. If the strategy halts at stage k (i.e., $\Box \in S_k$), *the computed proof is the ancestor-graph of* $\Box$ that has been traversed to generate $\Box$ during the derivation.

It is clear in this model why ordering-based strategies *generate many proof attempts*: at each stage i each ancestor-graph $(v; e; (t_1 \dots t_{n+1}))$ in G_i^+ is a proof of $l(v)$ and an attempt at a proof of $\Box$, because it may be possible to continue it into an ancestor-graph of $\Box$. Of course, the strategy does not know which proof attempts (ancestor-graphs) can be extended into a proof (an ancestor-graph of $\Box$). This is equivalent to saying that the strategy does not know which clauses in S_i are ancestors of $\Box$. Also, the strategy works on S_i, not on G_i^+: hence the proof attempts are built *implicitly*.

After an empty clause has been generated, the prover engages in *proof reconstruction* to make the proof *explicit*. Proof reconstruction is the operation of extracting the ancestor-graph of $\Box$ from G_k^*. For instance, in Otter [97] and EQP [98], each clause is stored with its identifier and its "justification," that is, the name of the inference rule that generated it, and the identifiers of its parents. As soon as an empty clause is generated, the prover reconstructs the proof by listing first the empty clause, then its parents, then the parents of each parent and so on, until it reaches input clauses. Then, this list of clauses is printed with the input clauses first and the empty clause last.

The proof may include contraction steps and clauses deleted by contraction. Clauses deleted by forward contraction are not used as premises of other steps before deletion and therefore cannot occur in the proof. Clauses deleted by backward contraction may occur, because they may have been used as premises of other steps before being deleted. Therefore, provers such as Otter or EQP need to save the clauses deleted by backward contraction in a separate component of the database, which will be consulted only by the proof reconstruction algorithm.

Also, the proof is generally a graph, not a tree, because a clause may be used more than once in the proof, and all variants of a clause are associated to the same vertex. However, once the proof has been extracted, it is possible to transform it into a tree, by creating a distinct vertex (in the tree) for each occurrence of a clause in the proof. The resulting tree is a *deduction tree* [45], and since it is a deduction tree of $\Box$, it is a *refutation* of the initial set S.

Remarks and further reading A model of the search space of resolution as a search graph was given by Kowalski in [81]. The model given here appeared in [39]: it is compatible with the one in [81] for the representation of expansion inferences, it has been the first to model contraction inferences, and it has been extended to distributed search in [32]. A different representation of the search space is adopted by Plaisted and Zhu in [113] for other purposes. Comparisons of the model given here with those in [81] and [113] can be found in [39, 31].

2.4 Expansion-oriented and contraction-based strategies

Ordering-based strategies can be classified further based on usage of contraction and degree of goal-sensitivity, as shown in Figure 1.

Strategies that feature only expansion rules, and strategies that apply contraction rules only for forward contraction, are called *expansion-oriented strategies*[6]. If the strategy features contraction rules, it is convenient to separate the newly generated clauses from the others, because the former are subject to contraction, and the latter are not. Thus, the elements of *States* are pairs of sets $(S; N)$, where S is the set of clauses that may be used as premises of expansion or as simplifiers, whereas N is the set of *raw clauses*, or the newly generated clauses that need to be normalized before being inserted in S. If

[6] The names expansion-oriented and contraction-based appeared in [35].

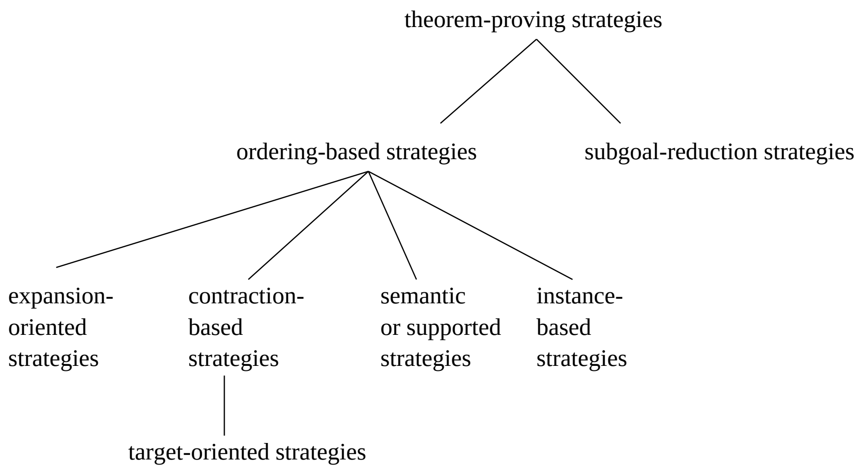
theorem-proving strategies
ordering-based strategies
subgoal-reduction strategies
expansion-oriented strategies
contraction-based strategies
semantic or supported strategies
instance-based strategies
target-oriented strategies

Fig. 1. Classes of ordering-based strategies

$\zeta((S_0;N_0)\dots(S_i;N_i)) = f \in I_E$, then $\xi((S_0;N_0)\dots(S_i;N_i),f) = X \subseteq S_i$, because clauses in N are not eligible to be premises for expansion until they are normalized and moved to S. If $\zeta((S_0;N_0)\dots(S_i;N_i)) = f \in I_R$, then $\xi((S_0;N_0)\dots(S_i;N_i),f) = X \subseteq S_i \cup N_i$, because contraction applies clauses in S to reduce clauses in N; and $\omega((S_i;N_i)) = \mathit{true}$ if $\Box \in S_i \cup N_i$.

Given a theorem-proving problem S, an expansion-oriented strategy $\mathcal{C} = \langle I, \Sigma\rangle$, with $\Sigma =< \zeta, \xi, \omega >$, generates the *derivation*

$$(S_0;N_0) \underset{\mathcal{C}}{\vdash} \dots (S_i;N_i) \underset{\mathcal{C}}{\vdash} \dots$$

where $S_0 = S$, $N_0 = \emptyset$, and $\forall i \geq 0$, if $\omega((S_i;N_i)) = \mathit{false}$,
$\zeta((S_0;N_0)\dots(S_i;N_i)) = f$, and $\xi((S_0;N_0)\dots(S_i;N_i),f) = X$, then

$$(S_{i+1};N_{i+1}) = \begin{cases} (S_i; N_i \cup \pi_1(f(X))) & \text{if } f \in I_E, \\ (S_i \cup \pi_1(f(X)); N_i - \pi_2(f(X))) & \text{if } f \in I_R \text{ and } \pi_1(f(X)) \text{ is the} \\ & S_i\text{-normal form of } \pi_2(f(X)), \\ (S_i; N_i \cup \pi_1(f(X)) - \pi_2(f(X))) & \text{otherwise.} \end{cases}$$

While expansion-oriented strategies allow a limited amount of contraction, at the other extreme there are *contraction-based strategies*, which not only allow both forward and backward contraction, but require contraction to be *eager*:
A derivation $S_0 \vdash_{\mathcal{C}} \dots S_i \vdash_{\mathcal{C}} \dots$ has *eager contraction*, if for all $i \geq 0$ and $\varphi \in S_i$, if there are $f \in I_R$ and $X \subseteq S_i$, such that $\pi_2(f(X)) = \{\varphi\}$, then there exists an $l \geq i$ such that $S_l \vdash S_{l+1}$ deletes φ, and $\forall j, i \leq j \leq l$, $S_j \vdash S_{j+1}$ is not an expansion inference, unless the derivation succeeds sooner.
A search plan Σ is an *eager-contraction search plan*, if all derivations controlled by Σ have eager contraction. A strategy $\mathcal{C}$ is *contraction-based*, if its inference system includes contraction rules and its search plan is eager-contraction.

The component of the search plan which is mostly responsible for eager contraction is the rule-selecting function ζ. For instance, the given-clause search plan of Otter that was described in Section 2.2 has eager forward contraction, because each raw clause ψ is reduced by forward contraction to its normal form ψ' immediately after generation, but it does not have eager backward-contraction, because ψ' is not used to contract other clauses, until after all clauses that can be generated by the current given clause have been generated, forward-contracted and appended to `sos`. The search plans of EQP also have eager backward-contraction: regardless of whether ψ was generated by the given-clause algorithm or the pair algorithm (see Section 2.2), ψ' is applied to contract other clauses right after its generation. If ψ' backward-simplifies an existing clause φ to a new form φ', also φ' is applied to do backward contraction as soon as possible. Thus, the cycle of expansion inferences does not restart until all applicable backward contraction has been performed. It may happen that this prevents the generation of clauses generated by the Otter's search plan.

Remarks and further reading Early forward-reasoning strategies were typically expansion-oriented (e.g., based primarily on expansion by resolution). The merging of the resolution-paramodulation paradigm with the term-rewriting and Knuth-Bendix paradigm has led to contraction-based strategies. Most ordering-based provers developed in recent years (e.g., Otter [97], RRL [76], Reveal [3], SNARK [134], EQP [98], SPASS [142], Barcelona [104], and daTac [138]) are based on contraction-based strategies, and also thanks to them succeeded in solving challenge problems (e.g., [3, 2, 75, 134, 99, 138]).

2.5 Target-oriented strategies

The question of how to make contraction-based strategies more goal-sensitive has long been a challenge to the automated deduction community. The situation of equational reasoning has been peculiar in this respect: the problem of goal-sensitivity was ignored for a long time, because the strategies based on the term-rewriting and Knuth-Bendix paradigm were regarded as completion procedures to generate confluent (or saturated) rewrite systems, rather than theorem-proving strategies. This view is not practical, because most theories have infinite saturated systems, so that it is impossible to compile first the theory into a saturated system and then use the latter as a decision procedure for the theory. Furthermore, since completion procedures do not have a goal, the issue of goal-sensitivity for theorem-proving applications was not considered. Nonetheless, working with a simpler logic has some advantages in this respect, and some progress was made.

In the purely equational case, H is a set of equations E, and φ has the form $\forall \bar{x}\ s \simeq t$, so that $\neg\varphi$ has the form $\hat{s} \not\simeq \hat{t}$, where the hat denotes that all variables have been replaced by Skolem constants. Since the negation of the target theorem is the only negative clause, it is trivial for the strategy to identify it. Furthermore, the negation of the target theorem is ground, so that unification involving a target term reduces to matching, and inferences on the target are simpler than general inferences. Therefore, it is possible to characterize contraction-based strategies for equational theories as *target-oriented strategies*. *States* is the set of pairs in the form $(E;\varphi)$, where E is the *presentation* and φ is the *target theorem*, which may be a ground equality $\hat{s} \simeq \hat{t}$ (e.g., in strategies based on *Unfailing Knuth-Bendix completion* and some of its extensions [69, 10, 36]), or a disjunction of ground equalities (e.g., in the extension of UKB of [3], called *Inequality Ordered Saturation* or *IOS-strategy* in [36]), or a disjunction of equalities with existentially quantified variables (e.g., in the *S-strategy* of [69]).

Accordingly, one can distinguish inference rules that apply to the presentation (forward reasoning) and inference rules that apply to the target (backward reasoning):

- *Presentation inference rules*:
  - *Expansion inference rules*: $f\colon\ \dfrac{(E;\varphi)}{(E';\varphi)}$ where $E \subset E'$ and $E \prec_{mul} E'$.
  - *Contraction inference rules*: $f\colon\ \dfrac{(E;\varphi)}{(E';\varphi)}$ where $E \not\subseteq E'$ and $E' \prec_{mul} E$.
- *Target inference rules*:
  - *Expansion inference rules*: $f\colon\ \dfrac{(E;\varphi)}{(E;\varphi')}$ where φ implies φ'.
  - *Contraction inference rules*: $f\colon\ \dfrac{(E;\varphi)}{(E;\varphi')}$ where φ does not imply φ'.

An example of target contraction inference rule is simplification of the target, where the old target alone obviously does not imply its reduced form. An example of target expansion inference rule is the *ordered saturation* of the IOS-strategy:

$$\frac{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}\})}{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}, \hat{s}' \simeq \hat{t}'\})} \quad \begin{array}{l} \hat{s}|u = l\sigma \quad \hat{s}[r\sigma]_u \rightarrow^*_E \hat{s}' \quad \hat{t} \rightarrow^*_E \hat{t}' \\ \{\hat{s}', \hat{t}'\} \not\succeq_{mul} \{\hat{g}, \hat{d}\} \ \ \forall \hat{g} \simeq \hat{d} \in N \cup \{\hat{s} \simeq \hat{t}\} \end{array}$$

where E is a set (meaning a conjunction) of equations, N is a set (meaning a disjunction) of ground equalities, and clearly the old target logically implies the new one. Ordered saturation applies if $\hat{s} \prec \hat{s}[r\sigma]_u$, since if $\hat{s} \succ \hat{s}[r\sigma]_u$ held, simplification would apply. The target equality $\hat{s}' \simeq \hat{t}'$ might have a shorter proof than the other target equalities: the strategy keeps multiple target equalities to broaden the chance of reaching a proof as soon as possible.

The characterization of expansion and contraction of Section 2.1 applies to these rules as well: it is sufficient to negate the target and move it into the main set (e.g., if the disjunction of ground equalities that form the target of the IOS-strategy is negated and moved to the main set, it becomes a conjunction, or a set, of ground inequalities).

Since the negation of the target is not added to the presentation, success is not marked by the generation of the empty clause, but by the reduction of the target to *true*, as in the *target deletion* rule of the IOS-strategy:

$$\frac{(E; N \cup \{\hat{s} \simeq \hat{s}\})}{(E; true)}$$

If the target were negated and added to E, $\hat{s} \neq \hat{s}$ would generate an empty clause by resolving with $x = x$ (e.g., as done by the *unit conflict* in Otter, which needs $x = x$ in the input [97]), or because the theorem prover detects that the two sides of the inequality are equal (e.g., as done by the *unit conflict* in EQP which does not need $x = x$ in the input [98]). Accordingly, $\omega((E_i; \varphi_i)) = \mathit{true}$ if and only if φ_i is *true*.

Given a theorem-proving problem $(E_0; \varphi_0)$, a target-oriented strategy $\mathcal{C} = \langle I, \Sigma \rangle$, with $\Sigma =< \zeta, \xi, \omega >$, generates the *derivation*

$$(E_0; \varphi_0) \underset{\mathcal{C}}{\vdash} \ldots (E_i; \varphi_i) \underset{\mathcal{C}}{\vdash} \ldots$$

such that $\forall i \geq 0$, if $\omega((E_i, \varphi_i)) = \mathit{false}$, $\zeta((E_0; \varphi_0) \ldots (E_i; \varphi_i)) = f$, and $\xi((E_0; \varphi_0) \ldots (E_i; \varphi_i), f) = X$, then

$$(E_{i+1}; \varphi_{i+1}) = \begin{cases} (E_i; \varphi') & \text{if } \varphi_i \in X,\ \pi_1(f(X)) = \{\varphi'\} \\ & \text{and } \pi_2(f(X)) = \{\varphi_i\}, \\ (E_i \cup \pi_1(f(X)) - \pi_2(f(X)); \varphi_i) & \text{otherwise.} \end{cases}$$

A target-oriented strategy does not need monotonicity, but only *relevance*: if $(E; \varphi) \vdash_{\mathcal{C}} (E'; \varphi')$, then $\varphi' \in Th(E')$ if and only if $\varphi \in Th(E)$.

Remarks and further reading Equational contraction-based strategies have been called rewriting-based, Knuth-Bendix-based, or completion-based. Their full characterization as target-oriented strategies was given in [36]. These strategies may take advantage of target-oriented heuristics, such as those of [3, 1, 51]. For instance, the combination of target inference rules, target-oriented heuristics, and inference rules for *cancellation* [71] in the IOS-strategy made possible to obtain results – the proofs of the Moufang identities [3] – that do not seem to have been reproduced by other theorem provers.

2.6 Semantic and supported strategies

In resolution-based theorem proving, the question of adding some backward reasoning to forward-reasoning strategies has been intertwined with the issue of limiting the generative power of expansion inference rules. In addition to restrictions based on redundancy criteria, restrictions that take knowledge about the problem into account have been investigated.

Since knowledge about the problem is semantic knowledge, *semantic resolution* [124] controls resolution by an interpretation $\mathcal{I}$: the given set of clauses S is partitioned into the subset T of all clauses in S that are satisfied by $\mathcal{I}$ and its complement $S - T$. Resolution is restricted in such a way that the consistent subset T is not expanded; only resolution steps with at most one premise from T are allowed: a clause in either T or $S - T$, called *nucleus*, resolves with one or more clauses in $S - T$, called *electrons*, until a resolvent that is false in $\mathcal{I}$, and therefore belongs to $S - T$, is generated. Semantic resolution may also assume an ordering on predicate symbols, and then require that the literal resolved upon in an electron contains the greatest predicate symbol in the electron.

If the interpretation $\mathcal{I}$ is defined based on sign, one obtains *hyperresolution* [117]: in *positive* hyperresolution, $\mathcal{I}$ contains all the negative literals, T contains the non-positive clauses, $S - T$ contains the positive clauses, and the electrons are positive clauses (from $S - T$) that resolve away all the negative literals in the nucleus (from T) to generate a positive hyperresolvent. Dually, in *negative* hyperresolution, $\mathcal{I}$ contains all the positive literals, T contains the non-negative clauses, $S-T$ contains the negative clauses, and the electrons are negative clauses (from $S-T$) that resolve away all the positive literals in the nucleus (from T) to generate a negative hyperresolvent. Hyperresolution can be more restrictive than semantic resolution with other interpretations, because hyperresolution excludes steps where both nucleus and electron are in $S - T$ (e.g., two negative clauses cannot resolve).

The intention of orienting the strategy towards backward-reasoning is more explicit in *resolution with set of support* [143]: a *set of support* (SOS) is a subset of S such that $S - SOS$ is consistent; only resolution steps with at most one premise from $S - SOS$ are allowed and all resolvents are added to SOS. The *set-of-support strategy* of [143] prescribed to put in SOS the goal clauses (those obtained from the transformation into clausal form of the negation of the target theorem), while $S-SOS$ contains the input assumptions, which form a consistent set, barring errors. Thus, the effect of working with a set of support is that most of the work done by the strategy is backward reasoning from the goal clauses.

Resolution with set of support fits in the paradigm of semantic resolution, with $T = S-SOS$ as the consistent subset and $SOS = S-T$ as its complement. Accordingly, the inferences allowed by the strategy, i.e., those with the electrons in SOS, are called *supported inferences*. However, resolution with set of support is less restrictive than semantic resolution, because it has the same condition on the choice of the premises (at most one from T), but it does not require that only resolvents that are false in $\mathcal{I}$ are generated. For instance, if SOS initially contains the positive clauses, resolution with set of support will generate and add to SOS also non-positive clauses, whereas positive hyperresolution will generate only positive clauses. Resolution with set of support can be seen as semantic resolution assuming an ad-hoc interpretation that makes all clauses in T true, and the clauses in SOS and all their descendants false.

Positive resolution [117] is binary resolution where one of the premises must be a positive clause (one where all literals are positive). Dually, *negative resolution* requires that a premise is a negative clause (one where all literals are negative). These strategies are considered sometime supported strategies where SOS contains the positive or negative clauses, respectively. Actually, positive resolution is more restrictive than resolution with set of support where SOS originally contains the positive clauses, because the former only allows steps with a positive premise, whereas the latter also allows resolutions between generated non-positive premises, as long as at least one of them is in SOS. On the other hand, positive hyperresolution is more restrictive than positive resolution, because the latter does not guarantee that only positive resolvents are generated. In essence, positive resolution and negative resolution are not semantic strategies, because they do not assume an interpretation with the provision that only clauses false in the interpretation are derived.

Since these *semantic* (or *supported*) strategies work by generating and keeping clauses, it is natural to enhance them with contraction rules (e.g., tautology deletion, subsumption, clausal simplification). For instance this combination is available in Otter [97]. In order to preserve completeness, however, it is necessary to apply contraction rules that replace clauses by other clauses, such as clausal simplification, in a way that respects the partition based on the interpretation: reduced forms of SOS-clauses stay in SOS, whereas reduced forms of T-clauses can stay in T only if they are true in the interpretation, and move to SOS otherwise [38].

A derivation by these strategies can be described as follows: $States$ is a set of pairs $(T;SOS)$, ξ_1 selects the nucleus, ξ_2 selects the electrons from SOS, and $\omega((T;SOS)) = true$ if SOS contains $\Box$. Given a theorem-proving problem S, and an interpretation $\mathcal{I}$, let $T_0 = \{\psi \mid \psi \in S,\ \mathcal{I} \models \psi\}$ and $SOS_0 = S - T_0$. The *derivation* generated by a semantic (or supported) strategy $\mathcal{C} = \langle I, \Sigma \rangle$, with $\Sigma =< \zeta, \xi, \omega >$, is the sequence

$$(T_0;SOS_0) \underset{\mathcal{C}}{\vdash} \ldots (T_i;SOS_i) \underset{\mathcal{C}}{\vdash} \ldots$$

such that $\forall i \geq 0$, if $\omega((T_i,SOS_i)) = false$, $\zeta((T_0;SOS_0)\ldots(T_i;SOS_i)) = f$, and $\xi((T_0;SOS_0)\ldots(T_i;SOS_i), f) = X$, then

$$(T_{i+1};SOS_{i+1}) = \begin{cases} (T_i;SOS_i \cup \pi_1(f(X))) & \text{if } f \in I_E; \\ (T_i;SOS_i \cup \pi_1(f(X)) - \pi_2(f(X))) & \text{if } f \in I_R \text{ and} \\ & \pi_2(f(X)) \in SOS_i; \\ (T_i \cup \pi_1(f(X)) - \pi_2(f(X));SOS_i) & \text{if } f \in I_R \text{ and } X \subseteq T_i; \\ (T_i - \pi_2(f(X));SOS_i \cup \pi_1(f(X))) & \text{if } f \in I_R,\ \pi_2(f(X)) \in T_i, \\ & X - \pi_2(f(X)) \subseteq SOS_i, \\ & \text{and } \mathcal{I} \not\models \pi_1(f(X)); \\ (T_i \cup \pi_1(f(X)) - \pi_2(f(X));SOS_i) & \text{if } f \in I_R,\ \pi_2(f(X)) \in T_i, \\ & X - \pi_2(f(X)) \subseteq SOS_i, \\ & \text{and } \mathcal{I} \models \pi_1(f(X)). \end{cases}$$

The general definitions of semantic resolution and resolution with set of support imply neither backward reasoning nor forward reasoning. The type of reasoning produced by the strategy depends on the interpretation and the form of the problem. For instance, if the assumptions are non-negative clauses and the goal clauses are negative clauses, positive hyperresolution is a forward-reasoning strategy and negative hyperresolution is a backward-reasoning strategy compatible with the set-of-support strategy of [143]; this is the case in Horn logic. In general, the partition of S into T and SOS based on the distinction between assumptions and goal clauses may not agree with the partition based on sign (e.g., the goal clauses may not be negative clauses), so that hyperresolution and the set-of-support strategy are not always compatible.

In essence, if the set of support contains assumptions, supported inferences are forward inferences, and the semantic strategy is a forward strategy; if the set of support contains goal clauses, supported inferences are backward inferences, and the semantic strategy is a backward strategy. For instance in Otter or EQP [98], the `sos` list can be seen as SOS and the `usable` list as T. If one puts in `sos` only the goal clauses, the resulting strategy is a backward-reasoning strategy (i.e., the set-of-support strategy of [143]). By putting more input clauses in `sos`, one increases the forward character of the resulting strategy: for example, if the formulation of the problem includes fundamental axioms, special assumptions and the goal clauses, one can put the axioms in `usable` and the rest in `sos`. If all input clauses are in `sos`, the outcome is a pure forward-reasoning strategy. The latter is often the best choice for purely equational problems: based on this experience, the `auto` mode of Otter (the mechanism for automated choice of the strategy) places all input clauses in `sos` if equality is the only predicate in the input. Table 2 summarizes the considerations made so far on the issue of forward and backward reasoning.

	Forward reasoning	Backward reasoning
Expansion-oriented	all inferences	
Contraction-based	all inferences	
Target-oriented	inferences on the presentation	inferences on the target
Supported	if SOS contains assumptions	if SOS contains goals

Table 2. Classification of ordering-based strategies in terms of forward and backward reasoning

Semantic strategies can be enriched with controlled forms of *lemmatization*, where lemmas are the product of selected unsupported inferences [38]. If the set of support contains assumptions, lemmatization adds backward reasoning to a forward strategy; if the set of support contains goal clauses, lemmatization adds forward reasoning to a backward strategy. Thus, lemmatization is a general technique to combine forward and backward reasoning.

Remarks and further reading Much work has been done on trying to combine restrictions of resolution. The semantic resolution of [124] is compatible with assuming an ordering on predicate symbols, and establishing that the literal resolved upon in an electron contains the greatest predicate symbol in the electron. In propositional logic, an ordering on predicate symbols is also an ordering on literals; in first-order logic the two are different. Thus, semantic or supported strategies for ordered resolution have been investigated. The combination of semantic resolution with the early formulations of ordered resolution (with clauses treated as lists, hence an arbitrary ordering on literals) was not complete: this early work is presented in Section 6.6 of [45], which we refer to for references. With ordered resolution based on a complete simplification ordering on literals, the *positive ordered strategy* (positive ordered resolution and paramodulation) and *positive unit strategy* (for Horn logic) were proved complete in [70]. The *maximal-literal unit strategy* of [54] combines the unit restriction for Horn logic with the ordering. Many refinements of resolution are revisited in [13], including *positive ordered resolution*, *ordered resolution with maximal selection* (similar to positive ordered hyperresolution), and *ordered resolution with set of support*. Hyperresolution and the positive ordered strategy are used to design model-building methods in [58, 59].

2.7 Instance-based strategies

The principle of instance-based strategies is to implement directly the Herbrand Theorem (e.g., [45]): prove the unsatisfiability of S by generating sets of ground instances of its clauses, and applying an algorithm for propositional satisfiability to detect that one such set is unsatisfiable. Different methods differ in how they generate instances and test ground unsatisfiability.

The method of [84] interleaves instance generation by *hyperlinking* and unsatisfiability test by the *Davis-Putnam algorithm* [50]. A *hyperlink* involves a clause $N_1 \vee \ldots N_k$, called *nucleus*, and clauses $E_1, \ldots E_k$, called *electrons*, to generate an instance of the nucleus:

$$\frac{S \cup \{N_1 \vee \ldots N_k, E_1, \ldots E_k\}}{S \cup \{N_1 \vee \ldots N_k, E_1, \ldots E_k, (N_1 \vee \ldots N_k)\sigma\}} \quad \begin{array}{l} \forall i, 1 \leq i \leq k, \exists L_i \in E_i \\ N_i\sigma = \neg L_i\sigma \qquad (\sigma\ mgu) \end{array}$$

Variants of a same clause may be used in a hyperlink, and all literals of the nucleus are linked, since the purpose is not to generate a resolvent, but to instantiate the nucleus as much as possible. Contraction is limited to unit subsumption and clausal simplification, because unrestricted subsumption would delete all instances and defeat the purpose of the strategy. In this regard, instance-based strategies are expansion-oriented strategies (see Section 2.4), with state $(S; N)$, where N contains the generated instances. After all hyperlinks in S_i have been considered, and contraction has been applied, all clauses in $S_i \cup N_i$ are made ground, by replacing all variables by a constant, and the Davis-Putnam algorithm is applied to the resulting ground set: if it is unsatisfiable, the procedure halts successfully; otherwise, the next phase of hyperlinking starts on $S_{i+1} = S_i \cup N_i$.

The Davis-Putnam algorithm [50] decides satisfiability of a set of ground clauses by trying all possible interpretations by case analysis (implemented as *splitting*) [45]. The case analysis is enhanced with *tautology deletion*, *purity deletion* and *unit propagation* (equivalent to unit resolution and unit subsumption), and these operations can be made very efficient by using fast data structures (e.g., [149]).

Remarks and further reading The idea of instance-based strategies dates back to the first implementations of Herbrand theorem, and has regained popularity as the efficiency of propositional methods has improved: a summary of the early work can be found in Section 7 of [84]. Hyperlinking has been applied also to model generation [48]; it can be augmented with interpretations to produce semantic strategies [47], and combined with orderings, and forms of paramodulation and simplification, to handle equality (e.g., [114]).

While instance-based strategies may be considered radically different than other ordering-based strategies, because they generate instances instead of resolvents, from the point of view of this classification they belong to the same class, because they work on a set of objects (clauses), build many proof attempts (the ground sets), do not backtrack, and feature some contraction. The analysis in [113] emphasizes the difference between generating resolvents and generating instances, and compares instance-based strategies, other ordering-based strategies and subgoal-reduction strategies in terms of measures of *duplication* in the total search space. Intuitively, a strength of instance-based strategies is that they do not duplicate literals by combining the literals of the parents in each resolvent.

The Davis-Putnam algorithm is the basis for efficient theorem provers or model finders for ground problems, with extensions for finding small finite models of first-order inputs[7]. Two such systems are SATO [146, 148] and MACE [96], while FINDER is related to Davis-Putnam but better understood in terms of tableaux [125], and SEM combines Davis-Putnam with other techniques [150].

3 Subgoal-reduction strategies

Supported strategies may be more goal-sensitive than general ordering-based strategies, but they belong to the same category, because they work by generating and keeping clauses, so that their *States* are multisets of clauses like for general ordering-based strategies. *Subgoal-reduction strategies* single out a goal object and work by reducing the goal to subgoals: one can distinguish classes of subgoal-reduction strategies as in Figure 2.

The rest of this section covers first linear and linear-input clausal strategies, and then tableau-based strategies.

3.1 Linear clausal strategies

Ordering-based strategies generate a portion of the search space (e.g., G^*_k if the derivation succeeded at stage k), and then extract the generated proof (the ancestor-graph of $\Box$ contained in G^*_k). The idea of *linear strategies* is to restrict the search to search only for ancestor-graphs of $\Box$ in a certain form, with the advantage of keeping in memory only the current proof attempt.

Linear resolution (e.g., [82, 90]) starts with a set of clauses $S = T \cup \{\varphi_0\}$, where clause φ_0 has been selected as the *top clause.* At each step i of the derivation, the strategy generates clause φ_{i+1} by resolving the *center clause* φ_i with a *side clause*, either a clause in T (an *input clause*), or a clause φ_j such that $j < i$

[7] Finite/infinite model means model with finite/infinite domain.

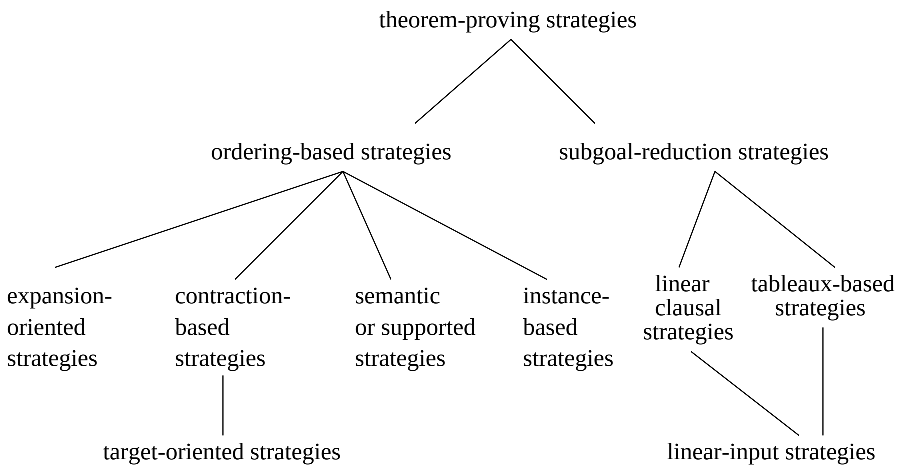
theorem-proving strategies
ordering-based strategies
subgoal-reduction strategies
expansion-oriented strategies
contraction-based strategies
semantic or supported strategies
instance-based strategies
linear clausal strategies
tableaux-based strategies
target-oriented strategies
linear-input strategies

Fig. 2. Classes of strategies

(an *ancestor clause*). The strategy succeeds at stage k if φ_k is the empty clause, and the $\varphi_0 \dots \varphi_k$ together with the side clauses form a comb-like ancestor-graph of $\Box$. Such an ancestor-graph is a *linear deduction tree*, and furthermore a *linear refutation*, because it is a deduction tree of $\Box$. Linear resolution is refutationally complete, in the sense that if $S = T \cup \{\varphi_0\}$ is unsatisfiable and T is consistent, there exists a linear refutation of S with φ_0 as the top clause. In other words, there exists in the search space a comb-like ancestor-graph of $\Box$ made of resolution steps with φ_0 as top clause and side clauses defined as above. It is sufficient to consider the center clauses as goals to see that linear resolution is a subgoal-reduction strategy: the most recently generated center clause is the current goal, each step consists in reducing the current goal to a subgoal, and the previous center clauses are ancestors of the current goal.

Refutational completeness guarantees the existence of a linear refutation, but the strategy needs to search for one. An ordering-based strategy is not looking for a proof of a specific form, and therefore it accumulates whatever it generates that is not redundant. A linear strategy, on the other hand, is looking for a linear refutation, and if it emerges that the deduction tree built so far cannot become a linear refutation, it needs to *backtrack*, that is, undo the last step and try a different continuation of the deduction tree. Therefore, search plans for linear strategies work by backtracking.

The set of *States* for these strategies contains triples $(T; \varphi; A)$, where φ is the current center clause, and A is the set of its ancestors. The search plan $\Sigma = \langle \zeta, \xi_1, \xi_2, \omega \rangle$ operates as follows:

- Since the primary premise is the current goal, the task of $\xi_1 \colon States^* \rightarrow \mathcal{L}_\Theta$ is to select a literal in the current goal:
  $\xi_1((T; \varphi_0; A_0) \dots (T; \varphi_i; A_i)) = L \in \varphi_i$;

– $\zeta\colon \mathit{States}^* \times \mathcal{L}_\Theta \to I \cup \{\mathit{backtrack}\}$ accounts for the selection of the inference rule, and the decision to backtrack;
– $\xi_2\colon \mathit{States}^* \times \mathcal{L}_\Theta \times I \to \mathcal{L}_\Theta$ chooses the secondary premise among clauses in T or ancestors of the current goal:
$\xi_2((T;\varphi_0;A_0)\dots(T;\varphi_i;A_i), L, f) = \psi \in T \cup A_i$;
– $\omega((T;\varphi;A)) = \mathit{true}$ if and only if $\varphi = \Box$.

Note how the definition of search plan for subgoal-reduction strategies fits in the same $\langle\zeta,\xi_1,\xi_2,\omega\rangle$ template used for ordering-based strategies.

Given a theorem-proving problem $S = T \cup \{\varphi_0\}$, a linear strategy $\mathcal{C} = \langle I, \Sigma\rangle$, with $\Sigma =< \zeta, \xi_1, \xi_2, \omega >$, generates the *derivation*

$$(T_0;\varphi_0;A_0) \underset{\mathcal{C}}{\vdash} \dots (T_i;\varphi_i;A_i) \underset{\mathcal{C}}{\vdash} \dots$$

such that $T_0 = T$, $A_0 = \emptyset$, and for all $i \geq 0$, if $\omega((T_i;\varphi_i;A_i)) = \mathit{false}$, $\xi_1((T_0;\varphi_0;A_0)\dots(T_i;\varphi_i;A_i)) = L$, then

$$(T_{i+1};\varphi_{i+1};A_{i+1}) = \begin{cases} (T_i;\pi_1(f(\varphi_i,\psi));A_i \cup \{\varphi_i\}) \\ \text{if } \zeta((T_0;\varphi_0;A_0)\dots(T_i;\varphi_i;A_i),L) = f \in I \\ \text{and } \xi_2((T_0;\varphi_0;A_0)\dots(T_i;\varphi_i;A_i),L,f) = \psi \\ \\ (T_{i-1};\varphi_{i-1};A_{i-1}) \\ \text{if } \zeta((T_0;\varphi_0;A_0)\dots(T_i;\varphi_i;A_i),L) = \mathit{backtrack}. \end{cases}$$

A characterization of inference rules other than the one in Section 2.1 is not necessary: the differences are that there is *no contraction* ($\pi_2(f(X)) = \emptyset$), and the generated clause ($\pi_1(f(X))$) is *not added to a set*, but used to *replace* its predecessor. The depth-first search of the subgoal-reduction strategy is captured by the form of the derivation itself, where the current goal is the most recently generated goal. Depth-first search with backtracking, however, is not fair, so that *depth-first search with backtracking and iterative deepening (DFID)* [80, 130] is used instead.

Linear resolution can be regarded as a refinement of resolution with set of support: the center clauses form the set of support, and the only needed resolution steps between clauses in SOS are the resolutions with ancestors. Linear resolution is obviously compatible with the set-of-support strategy of [143], if one chooses as top clause a goal clause. In such a case, linear resolution performs backward reasoning. However, like resolution with set of support performs forward or backward reasoning depending on what one puts in the set of support, a linear strategy may perform forward or backward reasoning depending on the choice of the top clause. Selecting a goal clause is natural and common, although not necessary. Any input clause φ_0 can be chosen as top clause as long as $T = S - \{\varphi_0\}$ is consistent.

Remarks and further reading Many authors contributed to linear resolution: see Section 7.1 of [45] for a summary and the relevant references. Other independently developed subgoal-reduction strategies include the *problem reduction format strategies* of [109], which also yield semantic strategies [103].

3.2 Linear-input clausal strategies

Linear resolution requires to keep the ancestors around; this is not necessary in *linear input strategies*, where all side clauses are input clauses. This class includes *linear input resolution*, which is complete for Horn logic, and *model elimination*, which is complete for first-order logic.

Inference rules *Model elimination* [89] proves that $S = T \cup \{\varphi_0\}$ is unsatisfiable by showing that no model of T satisfies φ_0 as follows. It works on *chains*, that can be seen as clauses made of plain literals, called *B-literals*, and framed literals, called *A-literals*. A chain encodes a stage of model construction: the A-literals are those that are true in the current candidate model, whereas the B-literals are those that still need to be considered. A model elimination strategy picks an input clause φ_0 to be the initial chain, and $T = S - \{\varphi_0\}$ contains all other input clauses. If T contains $L_1 \vee D$, φ_0 is $L_2 \vee C$, and the literals L_1 and L_2 have opposite sign and unify, the *ME-extension* rule applies:

$$\frac{(T \cup \{L_1 \vee D\}; L_2 \vee C)}{(T \cup \{L_1 \vee D\}; (D \vee [L_2] \vee C)\sigma)}\ L_1\sigma = \neg L_2\sigma \quad (\sigma\ most\ general\ unifier)$$

With this step, the procedure tries to build a T-model that makes $L_2\sigma$ true; since $L_1\sigma = \neg L_2\sigma$, such a model makes $L_1\sigma$ false. Therefore, in order to satisfy $(L_1 \vee D)\sigma$ it is necessary to satisfy $D\sigma$ by more ME-extension steps. Should this fail, it will be necessary to remove $L_2\sigma$ from the candidate model and try to satisfy $C\sigma$. The literals in $D\sigma$ are *subgoals* of $L_2\sigma$, because in order to satisfy $L_2\sigma$, the procedure needs to satisfy $D\sigma$, or, dually, in order to refute $L_2\sigma$ (and exclude it from the candidate model), the procedure needs to refute $D\sigma$.

If satisfying a subgoal would make the current candidate model inconsistent, the subgoal is eliminated by the *ME-reduction* rule:

$$\frac{(T; L \vee D \vee [L'] \vee C)}{(T; (D \vee [L'] \vee C)\sigma)}\ L\sigma = \neg L'\sigma$$

If $L\sigma = \neg L'\sigma$ and L' is already in the model, L cannot be part of it.

If the candidate model that makes an A-literal true fails to satisfy its subgoals (i.e., it is not a T-model), the A-literal must be removed from the candidate model. This is detected by the *ME-contraction* rule when all literals on the left of an A-literal have been eliminated:

$$\frac{(T; [L] \vee C)}{(T; C)}$$

This means that L has been refuted (no T-model includes it), or, equivalently, $\neg L$ has been proved. The inference system is completed by *factoring* on B-literals. If the current chain becomes empty, it means that no T-model satisfies φ_0, or $T \cup \{\varphi_0\}$ is unsatisfiable.

Because it works on one chain at a time, and has a natural notion of subgoaling, model elimination fits in the template of subgoal-reduction strategies with the current chain as the current goal. Furthermore, model elimination may also be presented as a refinement of linear resolution (e.g., [90, 82, 45]): the succession of chains corresponds to the succession of center clauses. ME-extension inferences can be interpreted as input-resolution inferences, modified by saving the literal resolved upon in the goal as an A-literal in the successor goal. ME-reduction inferences can be interpreted as what replaces ancestor-resolution inferences. In this interpretation, the mechanism of saving literals resolved upon as A-literals is the refinement that makes ancestor-resolution inferences unnecessary: the A-literals are precisely the ancestor literals that is necessary to keep to complete the refutation.

Search plans Since each step involves either the current goal and an input clause or the current goal only, the set of *States* for these strategies contains pairs $(T;\varphi)$, where φ is the current goal, and the component A of the ancestors is no longer needed. The search plan $\Sigma = \langle\zeta,\xi_1,\xi_2,\omega\rangle$ works as for linear strategies, except that ξ_2 selects the secondary premise from T only: $\xi_2((T;\varphi_0)\ldots(T;\varphi_i),L,f) = \psi \in T$.

In the first formulation of model elimination, ξ_1 selected literals in right to left order [89]. The typical choice for ξ_1 became left to right order (implictly assumed in the above presentation of the inference rules), in *Prolog Technology Theorem Proving* [130, 132], when it was discovered that model-elimination strategies can be implemented efficiently on top of a Prolog engine, such as the *Warren Abstract Machine* [140]. Since the set T of "axioms" – from a theorem proving point of view – or "program rules" – from a logic programming point of view – is static, it can be compiled at compile-time, and the strategy works on a stack of goals, operating at each step on the current goal, on top of the stack. In logic programming terminology, ξ_1 corresponds to the *AND computation rule*, and ξ_2 to the *OR computation rule*.

Given a theorem-proving problem $S = T \cup \{\varphi_0\}$, a linear input strategy $\mathcal{C} = \langle I,\Sigma\rangle$, with $\Sigma =< \zeta,\xi_1,\xi_2,\omega >$, generates the *derivation*

$$(T_0;\varphi_0)\underset{\mathcal{C}}{\vdash}\ldots(T_i;\varphi_i)\underset{\mathcal{C}}{\vdash}\ldots$$

such that $T_0 = T$, and $\forall i \geq 0$, if $\omega((T;\varphi_i)) = false$, $\xi_1((T_0;\varphi_0)\ldots(T_i;\varphi_i)) = L$, then

$$(T_{i+1};\varphi_{i+1}) = \begin{cases} (T_i;\pi_1(f(\varphi_i,\psi))) & \text{if } \zeta((T_0;\varphi_0)\ldots(T_i;\varphi_i),L) = f^2 \in I \\ & \text{and } \xi_2((T_0;\varphi_0)\ldots(T_i;\varphi_i),L,f^2) = \psi; \\ (T_i;\pi_1(f(\varphi_i))) & \text{if } \zeta((T_0;\varphi_0)\ldots(T_i;\varphi_i),L) = f^1 \in I; \\ (T_{i-1};\varphi_{i-1}) & \text{if } \zeta((T_0;\varphi_0)\ldots(T_i;\varphi_i),L) = \mathit{backtrack}. \end{cases}$$

The three cases cover, respectively, inferences involving the goal and an input clause, inferences on the goal only, and backtracking.

If the search plan is depth-first search with iterative deepening, ζ maintains the information of what is the depth bound of the current round of iterative deepening, say k, and associates to each goal φ_i a depth bound n_i $(0 \leq n_i \leq k_i)$, meaning that $k - n_i$ steps were used to reduce φ_0 to φ_i, and n_i more steps are allowed to try to reduce φ_i to $\Box$. If $n_i = 0$, ζ orders to backtrack, because no more steps are available for φ_i. If at stage $j-1$ the search space down to depth k has been exhausted, ζ resets the depth bound to $k+m$, for some $m > 0$, and starts the next round of iterative deepening in state $(T_j; \varphi_j)$, where $T_j = T$ and $\varphi_j = \varphi_0$, with $n_j = k + m$.

In addition to strategies for theorem proving and logic programming, strategies for functional programming and term rewriting can be seen as linear-input subgoal-reduction strategies. For instance, in term rewriting, T is a set of rewrite rules, φ_0 is the input term to be normalized, ξ_1 selects the subterm to be rewritten, and ξ_2 selects the rewrite rule to be applied. A main difference is that in term rewriting or functional programming there is only reduction, not search, and therefore no need for backtracking.

Refinements Subgoal-reduction strategies concentrate only on the current goal, and have no memory of previously solved goals. If the same subgoals, or instances thereof, are generated at different stages, the strategy solves them independently, repeating the same steps. While ordering-based strategies may run out of memory if they keep too many clauses, so that their set of clauses becomes too large, subgoal-reduction strategies may run out of memory if they repeat too many subgoals, so that their stack of goals becomes too large. Ordering-based strategies use contraction rules and restrictions to expansion rules to try to avoid the space explosion. Subgoal-reduction strategies try to avoid repetitions by using *pruning rules* and *lemmatization*.

Pruning rules affect backtracking, and therefore are part of the search plan. *Identical ancestor pruning* causes the procedure to backtrack if the current goal has the form $L \vee D \vee [L] \vee C$: if L has been inserted already in the candidate model, or, dually, if the procedure is trying already to refute L, it is useless to do it again.

Lemmatization is an extension of the inference system: when ME-contraction removes $[L]$, it means that no model of T satisfies L, hence $T \models \neg L$, and $\neg L$ can be added to T as a *lemma*. This holds, however, only if all subgoals of L were eliminated without recurring to ME-reduction by ancestors of L (i.e., A-literals on the right of $[L]$). If an ME-reduction step with ancestor $[A]$ was used, the lemma being proved is $\neg L \vee \neg A$ (no model of T satisfies $L \wedge A$, hence $T \models \neg L \vee \neg A$). If a subgoal of L was eliminated by factoring[8] with a B-literal B (on the right of $[L]$), the lemma being proved is $\neg L \vee B$ (no model of T satisfies $L \wedge \neg B$, hence $T \models \neg L \vee B$).

For reasons of efficiency, lemmatization may be restricted to the generation of unit lemmas, which are advantageous because a step with a unit lemma reduces the length of the goal. In Horn logic, all lemmas are unit lemmas, because ME-reduction is not necessary. Therefore, unit lemmatization can be replaced by *caching*, including *success caching*, where solutions are stored in a fast cache,

[8] In a typical ME-derivation ME-reduction applies more frequently than factoring.

rather than being turned into lemmas, and *failure caching*, which saves the information that a goal failed in order to avoid trying to solve it again (e.g., [110, 6]). Caching has the same logical justification as lemmatization, but it is different operationally: lemmas are used as premises for the regular inference mechanism of the subgoal-reduction strategy; the information stored in the cache is used to solve/fail a subgoal literal L without further inferences, based on the fact that the cache contains a more general subgoal L' that was solved or failed already.

Since it assumes that all lemmas are unit lemmas, caching is not consistent with inference systems for first-order logic, which need ME-reduction and factoring. It is not correct to solve a subgoal literal L by matching it with a cached solution $L'\rho$ of L', if the generation of this solution involved eliminating subgoals of L' by ME-reduction or factoring, because in such a case not $\neg L'\rho$ but some non-unit lemma $(\neg L' \vee \neg A_1 \ldots \vee \neg A_n)\rho$ was proved.

Both lemmatization and caching are enhancements of strategies that are already complete. Since cache retrieval replaces the regular inference mechanism, it is necessary to cache *all the solutions* in order to retain completeness. Also, caching is *incomplete* in the presence of identical ancestor pruning, because if identical ancestor pruning was used to prune the search of solutions of L', there is no guarantee that all solutions of L are instances of the cached solutions of L' [6]. Lemmatization obviously does not affect completeness.

In a strategy with lemmatization the T component is no longer static, because lemmas are added to T. Since clauses are generated and kept, it becomes possible to apply forms of contraction such as *lemma subsumption* or *cache subsumption* (e.g., [6, 129, 38]). Similar to linear resolution, model elimination does backward or forward reasoning depending on whether φ_0 is a goal clause or not. Assume the natural choice of picking as φ_0 a goal clause: then lemmatization adds forward reasoning to a backward-reasoning subgoal-reduction strategy. If the subgoal-reduction mechanism is interpreted as eliminating models, dually lemmatization can be interpreted as generating models, since a lemma is a logical consequence of T. Systems such as SATCHMO [93] and MGTP [66] use Prolog technology theorem proving as a basis for model generation.

Remarks and further reading Lemmatization was introduced with model elimination in [89]. At the time of its first implementation [61], unrestricted lemmatization generated more lemmas than the procedure could handle efficiently. This led to investigate weaker forms of lemmatization, such as *C-reduction* [121]. After the inception ([130, 132] and the MESON strategy in [91]) and maturation (e.g., [4, 19]) of Prolog technology theorem proving, lemmatization has been reintroduced, as in the METEOR theorem prover [5]. The analysis in [113] discusses how lemmaizing and caching may reduce from exponential to quadratic, or, at best, linear, certain measures of duplication in the search spaces of model elimination for problems in propositional Horn logic. A general treatment of lemmatization in supported strategies, covering lemmatization in model elimination as a special case, was given in [38]: it includes a formalization of caching and depth-dependent caching in iterative deepening strategies, and a discussion of contraction and caching as ways of reducing redundancy. Improvements of depth-first search with iterative deepening are proposed in [65].

The research on linear input resolution contributed to the invention of Prolog and logic programming: a theory-oriented introduction that emphasizes the connections with automated deduction can be found in [88]. A main conceptual difference is that in theorem proving one is interested in a refutation, whereas in logic programming one is interested in all the answer substitutions. This affects termination: in a theorem-proving problem, if the search space contains linear refutations, depth-first search with iterative deepening is guaranteed to find one and halt; in a logic programming problem, depth-first search with iterative deepening will reach all solutions eventually, but may still fail to terminate if the search space is infinite, because it deepens forever to look for more solutions. *Subsumption-based techniques* to enhance termination in Prolog were studied in [30]. *Linear Completion* is a linear strategy for logic programming with rewrite systems (e.g., [53, 33]). The effect of *simplification* on the termination of these programs was studied in [33], whose Linear Completion strategy also features a form of lemmatization (answer rules are added to the program to act as simplifiers). Section 8 in [33] contains comparisons with the subsumption-based techniques of [30], and the earlier work on Linear Completion. The relation between model elimination and computing answers in logic programming is investigated in [20]. Techniques similar to caching, called *memoing*, *tabling*, or *OLDT-resolution*, have been developed independently in logic programming to add a bottom-up component (logically, lemmas from the program) to top-down evaluations. Symmetrically, *magic sets* add a top-down component (logically, lemmas from the goal) to bottom-up evaluations in deductive databases. A survey of these areas was presented in [141], where more references can be found.

3.3 Tableaux-based strategies

While ordering-based strategies build implicitly many proof attempts, linear strategies build a proof attempt at a time, and backtrack to try another one. However, in clausal linear strategies the proof attempt is still implicit, because the strategy works on a stack of clauses and reconstructs at the end the produced linear refutation. *Tableau-based strategies* are subgoal-reduction strategies that inherit from natural deduction methods (e.g., analytic tableaux [126]) the property of working explicitly on the proof attempt. The inference system of the strategy is used to build a *tableau*: in essence, a tableau is a survey of the possible interpretations of S, with each branch representing an interpretation. If a branch contains a contradiction, the branch is *closed*, because it cannot be a viable interpretation. The purpose of the strategy is to develop the tableau to close all its branches, and show that S is unsatisfiable; the resulting *closed* tableau is the proof. The refutational completeness of the inference system guarantees the existence of a closed tableau if S is unsatisfiable. The search plan tries one tableau at a time: during the search an open tableau is a proof attempt, and if the current tableau cannot be closed, the search plan backtracks to try a different tableau. A fair search plan guarantees that a closed tableau will be generated if one exists. In propositional logic complete tableaux are finite, so that if the strategy terminates with a complete open tableaux its open branches represent models of S. In first-order logic the process of completing a tableau is infinite in general, but in some cases it is possible to extract a finite model from an open tableau.

Inference rules In *model elimination tableaux* [87, 86], the tableaux are built by using the inference rules of model elimination. A tableau is a tree with nodes labelled by literals. Given S, the strategy selects a clause $\varphi_0 = L_1 \vee \ldots \vee L_n$ to form the initial tableau $\mathcal{X}_0$, while $T = S - \{\varphi_0\}$ contains the remaining input clauses. $\mathcal{X}_0$ is a tree with no label at the root and n leaves labelled by the literals $L_1 \ldots L_n$, so that the branches represent all the ways of satisfying φ_0. If T contains a clause $F_1 \vee \ldots \vee F_k$ such that $F_1\sigma = \neg L_1\sigma$, ME-extension expands node L_1 with children $F_1 \ldots F_k$, closes node F_1 and applies σ to all literals in the resulting tableau. ME-reduction closes a leaf L if it has an ancestor L' such that $L\sigma = \neg L'\sigma$, and applies the unifier to all literals in the tableau. Factoring is also called *merging*, or *forward merging*, in tableaux terminology [139]: it closes a leaf L if there is an open node L' (e.g., sibling of an ancestor of L) such that $L\sigma = L'\sigma$, and applies σ to the tableau. When all the children of a node are closed, the node itself is closed (this corresponds to ME-contraction), and a tableau is closed when all its nodes are. Roughly speaking, open leaves in a tableau correspond to B-literals in a chain, while open internal nodes correspond to A-literals. A structural difference between the two is that ME-reduction and ME-contraction on chains remove literals, whereas in a tableau nodes are closed but not removed. The precise correspondence between model elimination operating on chains and model elimination operating on tableaux can be found in [18].

Assuming that $\mathcal{L}_\Theta$ is the language of clauses and tableaux on signature Θ, the characterization of inference rules as functions $f\colon \mathcal{P}(\mathcal{L}_\Theta) \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$ still holds. For instance, ME-extension takes a tableau and a clause and generates a tableau, while the other rules take a tableau and produce another tableau. Putting clauses and tableaux in $\mathcal{L}_\Theta$ is acceptable, considering that clauses and literals are terms, hence trees, and tableaux are trees. The rules in a basic subgoal-reduction strategy are expansion rules (regardless of whether the strategy works on clauses, or chains or tableaux), and what they generate is used to *replace* the previous goal, rather than being *added* to a set as in ordering-based strategies.

Search plans The elements of *States* have the form $(T; \mathcal{X})$, where $\mathcal{X}$ is the current tableau. The search plan $\Sigma = \langle \zeta, \xi_1, \xi_2, \omega \rangle$ works similarly to those for clausal linear-input strategies: ξ_1 selects an open leaf in the current tableau (i.e., if $open(\mathcal{X})$ denotes the open leaves in $\mathcal{X}$, $\xi_1((T_0; \mathcal{X}_0) \ldots (T_i; \mathcal{X}_i)) = L \in open(\mathcal{X}_i)$); ζ selects the inference rule and decides backtracking; ξ_2 selects a premise from T if needed; and $\omega((T; \mathcal{X})) = true$ if $\mathcal{X}$ is closed. Chains induce to think of left-to-right or right-to-left as the natural choices for ξ_1. Working with tableaux, on the other hand, ξ_1 can be any rule to visit a tree. The rule corresponding to selecting the leftmost literal in a chain is to select the leftmost open node in a tableau.

Given a theorem-proving problem $S = T \cup \{\varphi_0\}$, a tableau-based strategy $\mathcal{C} = \langle I, \Sigma \rangle$, with $\Sigma = <\zeta, \xi_1, \xi_2, \omega>$, generates the *derivation*

$$(T_0; \mathcal{X}_0) \underset{\mathcal{C}}{\vdash} \ldots (T_i; \mathcal{X}_i) \underset{\mathcal{C}}{\vdash} \ldots$$

such that $T_0 = T$, $\mathcal{X}_0$ is the tableau for φ_0, and $\forall i \geq 0$, if $\omega((T; \mathcal{X}_i)) = false$, $\xi_1((T_0; \mathcal{X}_0) \ldots (T_i; \mathcal{X}_i)) = L$, then

$$(T_{i+1}; \mathcal{X}_{i+1}) = \begin{cases} (T_i; f(\mathcal{X}_i, \psi)) & \text{if } \zeta((T_0; \mathcal{X}_0) \ldots (T_i; \mathcal{X}_i), L) = f^2 \in I \\ & \text{and } \xi_2((T_0; \mathcal{X}_0) \ldots (T_i; \mathcal{X}_i), L, f) = \psi \\ (T_i; f(\mathcal{X}_i)) & \text{if } \zeta((T_0; \mathcal{X}_0) \ldots (T_i; \mathcal{X}_i), L) = f^1 \in I \\ (T_{i-1}; \mathcal{X}_{i-1}) & \text{if } \zeta((T_0; \mathcal{X}_0) \ldots (T_i; \mathcal{X}_i), L) = backtrack. \end{cases}$$

Refinements The refinements of Section 3.2 apply also in the context of tableaux. Identical ancestor pruning is replaced by *equal predecessor fail*, or *regularity*: if a node is identical to one of its ancestors, the tableau is said to be *irregular*; if the strategy generates an irregular tableau, it discards it and backtracks.

When all the children of a node L are closed, L itself is closed, and a lemma $\neg L$ is proved. Since closed nodes are not removed, it is not necessary to add $\neg L$ to T: the information about the lemma is encoded in the tableau. Lemmatization in tableaux [87, 86] is also known as *regressive merging* or *backward merging* [139], because in the context of tableaux, and from an operational point of view, it has the appearance of the dual operation of merging. Applying a merging step consists in closing an open leaf L that unifies with another open leaf L'. Since L' is an open leaf, the strategy has not selected it yet, and this gives an idea of merging forward (collapsing a node on a node that will be selected). Applying a lemma consists in closing an open leaf L that unifies with a closed node L' (i.e., a lemma $\neg L'$). Since L' is closed, it has been selected already, and this gives an idea of merging backward (collapsing a node on an already selected node).

If the problem is first-order, not all lemmas are unit lemmas: if nodes below L were closed by ME-reduction steps with ancestors of L (e.g., $A_1 \ldots A_n$), the lemma attached to the closing of L is a non-unit lemma (e.g., $\neg L \vee \neg A_1 \ldots \vee \neg A_n$). Non-unit lemmatization can be implemented as *folding-up* [86, 139], which is a way of encoding the non-unit lemma in the tableau. Symmetrically, *folding-down* implements merging [64].

If lemmas are not generated explicitly, it may be more complicated to use them for contraction (e.g., subsumption) within T. The tableau-based prover Setheo [87, 64] uses subsumption, tautology deletion and purity deletion during the pre-processing of the input. If the Delta pre-processor [120] is invoked, the subgoal-reduction phase in Setheo is preceded by a phase where *UR-resolution* (Unit-Resulting resolution: unit electrons resolve against all but one literal in the nucleus to produce a unit resolvent [94]) and other restricted forms of resolution are applied to expand the set T, and the contraction rules are applied throughout this phase. The principle that makes expansion by resolution useful is essentially the same that makes lemmatization useful: it makes T more powerful for the subgoal-reduction phase, and it provides an integration of *forward reasoning* (hence *contraction*) and *backward reasoning* (hence *goal-sensitivity*). A difference is that unit lemmas are often less general than UR-resolvents, exactly because they are generated during the subgoal-reduction phase. On some problems unit lemmas are advantageous, because intuitively in terms of search they are "closer" to the solution. On other problems, unit-resolvents are more useful: exactly because they are more general, they are more powerful for unit subsumption and the subgoal-reduction inferences themselves. Another feature of Setheo are *anti-lemmas*, which is basically a form of depth-dependent failure caching.

A different approach to enhancing tableau-based methods with contraction is *tableau subsumption* [86, 16], which is based on defining a subsumption ordering among tableaux, and using tableaux to subsume others, similar to clausal subsumption. There are two main difficulties with defining a practical notion of tableau subsumption [16]. First, comparing entire tableaux seems inefficient. Thus, one would like to compare only open leaves (e.g., $\mathcal{X}_1$ subsumes $\mathcal{X}_2$ if $open(\mathcal{X}_1)$ subsumes $open(\mathcal{X}_2)$). However, a subsumption rule that compares only leaves destroys completeness, because completeness requires remembering ancestors for ME-reduction. Roughly speaking, for $\mathcal{X}_1$ to subsume $\mathcal{X}_2$, it is also necessary that all ancestors of open leaves in $\mathcal{X}_2$ appear as ancestors of open leaves in $\mathcal{X}_1$. Therefore, one would like a notion of tableau subsumption that preserves completeness without imposing to compare entire tableaux. The approach of [16] consists in defining a complete restriction of model elimination, called *disjunctive positive model elimination (DPME)*, which is a refinement of the *positive model elimination* of [112]: in DPME the only ancestors needed for ME-reduction are *disjunctive positive ancestors*, that is, positive ancestors coming from non-Horn clauses.

Then, a subsumption relation among tableaux is defined based on open leaves and disjunctive positive ancestors only. The potential downside of this approach is that the advantage of tableau subsumption may be outweighted by the disadvantage of restricting ME-reduction to disjunctive positive ancestors. In practice, ME-reduction is very important to keep the stack of goals from growing too large. The solution of the Mission prover [16], where disjunctive positive model elimination with tableau subsumption is implemented, is to use unrestricted ME-reduction, while considering only disjunctive positive ancestors in tableau subsumption.

The second issue is that in order to apply tableau subsumption the strategy needs to generate and save tableaux. In other words, one needs to abandon the subgoal-reduction framework, where only one tableau is kept in memory, and adopt the style of ordering-based strategies, envisioning a strategy that generates all possible tableaux, and applies forward and backward subsumption among tableaux [16]. Such a strategy is an ordering-based strategy that works with a set of tableaux, instead of a set of clauses. It is quite natural, however, to translate tableaux into clauses, and define such a strategy to work on clauses of A-literals and B-literals.

A less radical option is to maintain the subgoal-reduction framework, but save the predecessors of the current tableau, and use them for forward tableau-subsumption. The result is a linear tableau-based strategy with derivations in the form $(T_0; \mathcal{X}_0; A_0) \vdash_{\mathcal{C}} \ldots (T_i; \mathcal{X}_i; A_i) \vdash_{\mathcal{C}} \ldots$, where A_i contains the predecessor tableaux $\mathcal{X}_0 \ldots \mathcal{X}_{i-1}$. If a newly generated tableau is forward-subsumed by one of its predecessors, the strategy backtracks.

Table 3 summarizes some refinements of subgoal-reduction strategies (the distinction between clausal and tableaux model elimination in the table is mostly one of terminology, since almost everything that can be done in one can be done in the other).

	Combination of forward and backward reasoning	Contraction	Pruning
Model elimination	lemmatization C-reduction success caching	lemma subsumption cache subsumption	identical ancestor pruning failure caching
Prolog Datalog	tabling/memoing magic sets		cut
Tableaux	regressive merging folding up UR-resolution hyperlinking	tableau subsumption subsumption tautology deletion purity deletion	irregularity anti-lemmas

Table 3. Refinements of subgoal-reduction strategies

Remarks and further reading General treatments of tableaux-based strategies and their relations with other strategies can be found in [27, 49].

Equality has long been a weak point of subgoal-reduction strategies, because for reasoning with equalities it is natural to generate and keep equations, and use them to rewrite other equations. In [21], approaches to equip analytic tableaux to handle equality include adding expansion rules for equality, or using forms of E-unification. Exactly because forward-reasoning with equalities is not a native feature of tableaux, in this context equality reasoning is considered as a form of *theory reasoning*, to be handled by a specialized component of the theorem prover: a general treatment of this topic can be found in [15]. E-Setheo is a version of Setheo with equality, continuing in the spirit of [120] of combining forward-reasoning and subgoal-reduction.

In addition to Setheo and Mission, other model-elimination tableau provers include Protein [17] and KoMeT [26], while provers based on analytic tableaux include TAP [23, 22] and Tatzelwurm [29]. Protein extends Prolog technology theorem proving with *theory reasoning*; KoMeT has lemmatization together with lemma subsumption, depth-dependent failure caching and some theory reasoning; Tatzelwurm enhances tableau-based strategies with UR-resolution and instance generation by hyperlinking [84].

4 Discussion

An advantage of subgoal-reduction strategies is that at each stage of the derivation, they need to keep in memory only the current proof attempt (e.g., the current goal and its ancestors, or the current tableau), whereas ordering-based strategies need to keep in memory all generated clauses not deleted by contraction (e.g., G_i^+ at stage i). Thus, if we call *active search space* what is held in memory, subgoal-reduction strategies tends to have a smaller active search space than ordering-based strategies. It is a fallacy, on the other hand, to conclude that the search space generated by subgoal-reduction strategies is also small. This fallacy is due to a confusion of active search space and *generated search space*. Because the subgoal-reduction strategy searches by backtracking, its generated search space is equal to the union of all the partial proofs it has attempted. Thus, the generated search space may be large, even if the active search space is small.

Another misconception is to say that ordering-based strategies search for a clause – the empty clause – whereas tableau-based strategies search for a proof. All theorem-proving strategies search for a proof. The difference is that ordering-based strategies build their proof attempts implicitly, and when an empty clause is generated, extract the completed proof from the generated search space (e.g., G_k^*). On the other hand, tableau-based strategies generate explicitly one proof attempt at a time, backtrack to modify it, and succeed when it is completed. A related error is to blame ordering-based strategies for generating huge proofs: this is based on mistaking the generated search space for the computed proof. Table 4 clarifies these points.

	Ordering-based	Subgoal-reduction
Generated search space	all generated clauses	all tried tableaux
Active search space	all kept clauses	the current tableau
Generated proof	the ancestor-graph of $\Box$	the closed tableau

Table 4. Two main classes of strategies (revisited)

If small active search space is an advantage of subgoal-reduction strategies, a main advantage of ordering-based strategies is contraction, which not only deletes existing redundant clauses, but also prevents their descendants in the search space from being generated. The study in [39] analyzes this behavior in term of *bounded search spaces* defined over the infinite search spaces of theorem-proving problems, and shows that in a contraction-based derivation the bounded search spaces are monotonically decreasing. In summary, the generated search space is typically large and the computed proof represents a small portion of it for both classes of strategies. The difference is in how the search proceeds, as reflected by small active search space on one hand, and monotonically decreasing bounded search spaces on the other.

References

1. Siva Anantharaman and Nirina Andrianarivelo. Heuristical criteria in refutational theorem proving. In Alfonso Miola, editor, *Proceedings of the 1st DISCO*, volume 429 of *LNCS*, pages 184–193. Springer Verlag, 1990.
2. Siva Anantharaman and Maria Paola Bonacina. An application of automated equational reasoning to many-valued logic. In Stéphane Kaplan and Mitsuhiro Okada, editors, *Proceedings of CTRS-90*, volume 516 of *LNCS*, pages 156–161. Springer Verlag, 1990.
3. Siva Anantharaman and Jieh Hsiang. Automated proofs of the Moufang identities in alternative rings. *Journal of Automated Reasoning*, 6(1):76–109, 1990.
4. Owen L. Astrachan and Don W. Loveland. METEORs: High performance theorem provers using model elimination. Pages 31–60 in [40].
5. Owen L. Astrachan and Don W. Loveland. The use of lemmas in the model elimination procedure. *Journal of Automated Reasoning*, 19(1):117–141, 1997.
6. Owen L. Astrachan and Mark E. Stickel. Caching and lemmaizing in model elimination theorem provers. Pages 224–238 in [77].
7. Franz Baader and Jörg H. Siekmann. Unification theory. In Vol. 2 of [62].
8. Leo Bachmair and Nachum Dershowitz. Inference rules for rewrite-based first-order theorem proving. In *Proceedings of LICS-87*, pages 331–337. IEEE Computer Society Press, 1987.
9. Leo Bachmair and Nachum Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.
10. Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. Completion without failure. In Hassam Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume II: Rewriting Techniques, pages 1–30. Academic Press, 1989.
11. Leo Bachmair and Harald Ganzinger. Non-clausal resolution and superposition with selection and redundancy criteria. In Andrei Voronkov, editor, *Proceedings of LPAR-92*, volume 624 of *LNAI*, pages 273–284. Springer Verlag, 1992.
12. Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
13. Leo Bachmair and Harald Ganzinger. A theory of resolution. Technical Report MPI-I-97-2-005, Max Planck Institut für Informatik, 1997. To appear in J. Alan Robinson and Andrei Voronkov, eds., *Handbook of Automated Reasoning.*
14. Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation. *Information and Computation*, 121(2):172–192, 1995.
15. Peter Baumgartner. *Theory Reasoning in Connection Calculi*, volume 1527 of *LNAI*. Springer, 1998.

16. Peter Baumgartner and Stefan Brüning. A disjunctive positive refinement of model elimination and its application to subsumption deletion. *Journal of Automated Reasoning*, 19:205–262, 1997.
17. Peter Baumgartner and Ulrich Furbach. PROTEIN: a PROver with a Theory Extension INterface. Pages 769–773 in [41].
18. Peter Baumgartner and Ulrich Furbach. Consolution as a framework for comparing calculi. *Journal of Symbolic Computation*, 16(5), 1993.
19. Peter Baumgartner and Ulrich Furbach. Model elimination without contrapositives and its application to PTTP. *Journal of Automated Reasoning*, 13:339–359, 1994.
20. Peter Baumgartner, Ulrich Furbach, and Frieder Stolzenburg. Model elimination, logic programming and computing answers. *Artificial Intelligence*, 90(1–2):135–176, 1997.
21. Bernhard Beckert. Semantic tableaux with equality. *Journal of Logic and Computation*, 7(1):39–58, 1997.
22. Bernhard Beckert, Reiner Hähnle, P. Oel, and M. Sulzmann. The tableau-based theorem prover 3^{T^AP}, version 4.0. Pages 303–307 in [101].
23. Bernhard Beckert and Joachim Posegga. leanTAP: lean tableau-based theorem proving. Pages 793–797 in [41].
24. Wolfgang Bibel. *Automated Theorem Proving*. Friedr. Vieweg & Sohn, 2nd edition, 1987.
25. Wolfgang Bibel. *Deduction: Automated Logic*. Academic Press, 1993.
26. Wolfgang Bibel, Stefan Brüning, Uwe Egly, and T. Rath. KoMeT. Pages 783–788 in [41].
27. Wolfgang Bibel and E. Eder. Methods and calculi for deduction. Pages 68–183 in Vol. 1 of [62].
28. Wolfgang Bibel and P. H. Schmitt, Eds. *Automated Deduction – A Basis for Applications*. Kluwer, 1998.
29. Carsten Bierwald and Thomas Käufl. Tableau prover Tatzelwurm: hyper-links and UR-resolution. In Maria Paola Bonacina and Ulrich Furbach, editors, *Proceedings of the 1st FTP*, number 97-50 in Technical Reports of RISC, pages 22–28. Johannes Kepler Universität, 1997.
30. Roland N. Bol, Krzysztof R. Apt, and J. W. Klop. An analysis of loop checking mechanisms in logic programming. *Theoretical Computer Science*, 86:35–79, 1991.
31. Maria Paola Bonacina. A note on the analysis of theorem-proving strategies. AAR Newsletter, No. 36, pages 2–8, April 1997. Full version available as Technical Report, Department of Computer Science, University of Iowa, May 1996.
32. Maria Paola Bonacina. Analysis of distributed-search contraction-based strategies. In Jürgen Dix, Luis Fariñas del Cerro, and Ulrich Furbach, editors, *Proceedings of the 6th JELIA*, volume 1489 of *LNAI*, pages 107–121. Springer, 1998. Full version available as Tech. Rep., Dept. of Comp. Sci., Univ. of Iowa, April 1998.
33. Maria Paola Bonacina and Jieh Hsiang. On rewrite programs: semantics and relationship with Prolog. *Journal of Logic Programming*, 14(1 & 2):155–180, 1992.
34. Maria Paola Bonacina and Jieh Hsiang. On subsumption in distributed derivations. *Journal of Automated Reasoning*, 12:225–240, 1994.
35. Maria Paola Bonacina and Jieh Hsiang. Parallelization of deduction strategies: an analytical study. *Journal of Automated Reasoning*, 13:1–33, 1994.
36. Maria Paola Bonacina and Jieh Hsiang. Towards a foundation of completion procedures as semidecision procedures. *Theoretical Computer Science*, 146:199–242, 1995.

37. Maria Paola Bonacina and Jieh Hsiang. A category-theoretic treatment of automated theorem proving. *Journal of Information Science and Engineering*, 12(1):101–125, 1996.
38. Maria Paola Bonacina and Jieh Hsiang. On semantic resolution with lemmaizing and contraction and a formal treatment of caching. *New Generation Computing*, 16(2):163–200, 1998.
39. Maria Paola Bonacina and Jieh Hsiang. On the modelling of search in theorem proving – towards a theory of strategy analysis. *Information and Computation*, 147:171–208, 1998.
40. Robert S. Boyer, Ed. *Automated Reasoning – Essays in Honor of Woody Bledsoe.* Kluwer, 1991.
41. Alan Bundy, Ed. *Proceedings of the 12th CADE*, volume 814 of *LNAI*. Springer, 1994.
42. H.-J. Bürckert. *A Resolution Principle for a Logic with Restricted Quantifiers*, volume 568 of *LNAI*. Springer Verlag, 1991.
43. Ricardo Caferra and Nicolas Peltier. Model building in the cross-roads of consequence and non-consequence relations. In Maria Paola Bonacina and Ulrich Furbach, editors, *Proceedings of the 1st FTP*, number 97-50 in Technical Reports of RISC, pages 40–44. Johannes Kepler Universität, 1997.
44. Ricardo Caferra and N. Zabel. A method for simultaneous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13:613–641, 1992.
45. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.
46. Jim Christian. Flatterms, discrimination nets and fast term rewriting. *Journal of Automated Reasoning*, 10:95–113, 1993.
47. Heng Chu and David A. Plaisted. CLINS-S: a semantically guided first-order theorem prover. In [136].
48. Heng Chu and David A. Plaisted. Model finding in semantically guided instance-based theorem proving. *Fundamenta Informaticae*, 21(3):221–235, 1994.
49. M. D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga, Eds. *Handbook of Tableau Methods.* Kluwer, 1998.
50. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
51. Jörg Denzinger and M. Fuchs. Goal-oriented equational theorem proving using team-work. In *Proceedings of the 18th KI*, volume 861 of *LNAI*, pages 343–354. Springer, 1994.
52. Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
53. Nachum Dershowitz. Computing with rewrite systems. *Information and Control*, 65:122–157, 1985.
54. Nachum Dershowitz. Canonical sets of Horn clauses. In J. Leach Albert, B. Monien, and Mario Rodríguez Artalejo, editors, *Proceedings of the 18th ICALP*, volume 510 of *LNCS*, pages 267–278. Springer Verlag, 1991.
55. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, 1990.
56. Roland Dietrich. Relating resolution and algebraic completion for Horn logic. Pages 62–78 in [123].
57. Norbert Eisinger and Hans Jürgen Ohlbach. Deduction systems based on resolution. Pages 184–273 in Vol. 1 of [62].

58. Christian Fermüller and Alexander Leitsch. Hyperresolution and automated model-building. *Journal of Logic and Computation*, 6(2):173–203, 1996.
59. Christian Fermüller and Alexander Leitsch. Decision procedures and model-building in equational clause logic. *Journal of the IGPL*, 6(1):17–41, 1998.
60. Melvin Fitting. *First-order Logic and Automated Theorem Proving*. Springer, 1990.
61. S. Fleisig, Don W. Loveland, A. Smiley, and D. Yarmush. An implementation of the model elimination proof procedure. *Journal of the ACM*, 21:124–139, 1974.
62. Dov M. Gabbay, Christopher J. Hogger, and J. Alan Robinson, Eds. *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 1 & 2)*. Oxford University Press, 1993.
63. Jean Gallier. *Logic for Computer Science – Foundations of Automatic Theorem Proving*. Harper & Row, 1986.
64. Chr. Goller, Reinhold Letz, K. Mayr, and Johann Schumann. SETHEO v3.2: recent developments. Pages 778–782 in [41].
65. John Harrison. Optimizing proof search in model elimination. Pages 313–327 in [101].
66. Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. MGTP: a parallel theorem prover based on lazy model generation. Pages 776–780 in [77].
67. Jieh Hsiang. Refutational theorem proving using term rewriting systems. *Artificial Intelligence*, 25:255–300, 1985.
68. Jieh Hsiang. Rewrite method for theorem proving in first order theories with equality. *Journal of Symbolic Computation*, 3:133–151, 1987.
69. Jieh Hsiang and Michaël Rusinowitch. On word problems in equational theories. In Th. Ottman, editor, *Proceedings of the 14th ICALP*, volume 267 of *LNCS*, pages 54–71. Springer Verlag, 1987.
70. Jieh Hsiang and Michaël Rusinowitch. Proving refutational completeness of theorem proving strategies: the transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, 1991.
71. Jieh Hsiang, Michaël Rusinowitch, and Ko Sakai. Complete inference rules for the cancellation laws. In *Proceedings of the 10th IJCAI*, pages 990–992, 1987.
72. Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. Pages 257–321 in [83].
73. Deepak Kapur, Dave Musser, and Paliath Narendran. Only prime superposition need be considered in the Knuth-Bendix completion procedure. *Journal of Symbolic Computation*, 6:19–36, 1988.
74. Deepak Kapur and Paliath Narendran. An equational approach to theorem proving in first order predicate calculus. In *Proceedings of the 9th IJCAI*, pages 1146–1153, 1985.
75. Deepak Kapur and Hantao Zhang. A case study of the completion procedure: proving ring commutativity problems. Pages 360–394 in [83].
76. Deepak Kapur and Hantao Zhang. An overview of RRL: rewrite rule laboratory. In Nachum Dershowitz, editor, *Proceedings of the 3rd RTA*, volume 355 of *LNCS*, pages 513–529. Springer Verlag, 1989.
77. Deepak Kapur, Ed. *Proceedings of the 11th CADE*, volume 607 of *LNAI*. Springer, 1992.
78. Claude Kirchner, Hélène Kirchner, and Michaël Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990.
79. Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Proceedings of the Conf. on Computational Problems in Abstract Algebras*, pages 263–298. Pergamon Press, 1970.

80. R. E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
81. Robert Kowalski. Search strategies for theorem proving. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 181–201. Edinburgh University Press, 1969.
82. Robert Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2:227–260, 1971.
83. Jean-Louis Lassez and Gordon Plotkin, Eds. *Computational Logic – Essays in Honor of Alan Robinson*. The MIT Press, 1991.
84. Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyperlinking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.
85. Alexander Leitsch. *The Resolution Calculus*. Springer, 1997.
86. Reinhold Letz, K. Mayr, and Chr. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13(3):297–338, 1994.
87. Reinhold Letz, Johann Schumann, S. Bayerl, and Wolfgang Bibel. SETHEO: a high performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.
88. John W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 2nd edition, 1987.
89. Don W. Loveland. A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):349–363, 1969.
90. Don W. Loveland. A unifying view of some linear Herbrand procedures. *Journal of the ACM*, 19(2):366–384, 1972.
91. Don W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, 1978.
92. Ewing Lusk and Ross Overbeek, Eds. *Proceedings of the 9th CADE*, volume 310 of *LNCS*. Springer Verlag, 1988.
93. Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. Pages 415–434 in [92].
94. J. McCharen, Ross Overbeek, and Larry Wos. Complexity and related enhancements for automated theorem proving programs. *Computers and Mathematics with Applications*, 2(1):1–16, 1976.
95. William W. McCune. Experiments with discrimination tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.
96. William W. McCune. A Davis-Putnam program and its application to finite first-order model search: quasigroup existence problems. Unpublished manuscript, May 1994.
97. William W. McCune. Otter 3.0 reference manual and guide. Technical Report 94/6, MCS Division, Argonne National Laboratory, 1994.
98. William W. McCune. 33 Basic test problems: a practical evaluation of some paramodulation strategies. In Robert Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, pages 71–114. MIT Press, 1997.
99. William W. McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.
100. William W. McCune, Ed. *Proceedings of the 14th CADE*, volume 1249 of *LNAI*. Springer, 1997.
101. Michael McRobbie and John Slaney, Eds. *Proceedings of the 13th CADE*, volume 1104 of *LNAI*. Springer, 1996.

102. Jürgen Müller and Rolf Socher-Ambrosius. Topics in completion theorem proving. Technical Report SEKI SR-88-13, Fachbereich Informatik, Univ. Kaiserslautern, 1988.
103. Xumin Nie and David A. Plaisted. A complete semantic back chaining proof system. Pages 16–27 in [135].
104. Robert Niewenhuis, José Miguel Rivero, and Miguel Angel Vallejo. The Barcelona prover. In [136].
105. Robert Niewenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computation*, 19(4):321–351, 1995.
106. Judea Pearl. *Heuristics – Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley, 1984.
107. Gerald E. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal of Computing*, 12(1):82–100, 1983.
108. David A. Plaisted. Equational reasoning and term rewriting systems. Pages 273–364 in Vol. 1 of [62].
109. David A. Plaisted. A simplified problem reduction format. *Artificial Intelligence*, 18:227–261, 1982.
110. David A. Plaisted. Non-Horn clause logic programming without contrapositives. *Journal of Automated Reasoning*, 4(3):287–325, 1988.
111. David A. Plaisted. Mechanical theorem proving. In Ranan B. Banerji, editor, *Formal Techniques in Artificial Intelligence*. Elsevier, 1990.
112. David A. Plaisted. A sequent-style model elimination strategy and a positive refinement. *Journal of Automated Reasoning*, 6(4):389–402, 1990.
113. David A. Plaisted and Yunshan Zhu. *The Efficiency of Theorem Proving Strategies*. Friedr. Vieweg & Sohns, 1997.
114. David A. Plaisted and Yunshan Zhu. Ordered semantic hyper linking. In *Proceedings of AAAI-97*, 1997.
115. Allan Ramsay. *Formal Methods in Artificial Intelligence*. Cambridge University Press, 1988.
116. G. Robinson and Larry Wos. Paramodulation and theorem-proving in first-order theories with equality. In D. Michie and R. Meltzer, editors, *Machine Intelligence*, volume IV, pages 135–150. Edinburgh Univ. Press, 1969.
117. J. Alan Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.
118. J. Alan Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
119. Michaël Rusinowitch. Theorem-proving with resolution and superposition. *Journal of Symbolic Computation*, 11(1 & 2):21–50, 1991.
120. Johann Schumann. Delta: a bottom-up pre-processor for top-down theorem provers. Pages 774–777 in [41].
121. Robert E. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.
122. Jörg H. Siekmann and Graham Wrightson, Eds. *Automation of reasoning – Classical Papers on Computational Logic*. Springer Verlag, 1983.
123. Jörg H. Siekmann, Ed. *Proceedings of the 8th CADE*, volume 230 of *LNCS*. Springer, 1986.
124. James R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697, 1967.
125. John Slaney. FINDER: finite domain enumerator. Pages 798–801 in [41].
126. Raymond M. Smullyan. *First-Order Logic*. Dover, 1995. (Republication of the work first published as "Band 43" Series *Ergebnisse der Mathematik und ihrer Grenzgebiete*, Springer Verlag, 1968).

127. Rolf Socher-Ambrosius. How to avoid the derivation of redundant clauses in reasoning systems. *Journal of Automated Reasoning*, 9(1):77–98, 1992.
128. Rolf Socher-Ambrosius and Patricia Johann. *Deduction systems.* Springer, 1997.
129. Mark E. Stickel. PTTP and linked inference. Pages 283–296 in [40].
130. Mark E. Stickel. A Prolog technology theorem prover. *New Generation Computing*, 2(4):371–383, 1984.
131. Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.
132. Mark E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.
133. Mark E. Stickel. The path-indexing method for indexing terms. Technical Report 473, SRI International, 1989.
134. Mark E. Stickel, Richard Waldinger, Michael Lowry, Thomas Pressburger, and Ian Underwood. Deductive composition of astronomical software from subroutine libraries. Pages 341–355 in [41].
135. Mark E. Stickel, Ed. *Proceedings of the 10th CADE*, volume 449 of *LNAI*. Springer, 1990.
136. Geoff Sutcliffe and Christian Suttner, Eds. The CADE-13 ATP system competition. *Journal of Automated Reasoning*, 18(2), 1997.
137. Tanel Tammet. Gandalf. Pages 199–204 in [136].
138. Laurent Vigneron. Automated deduction techniques for studying rough algebras. *Fundamenta Informaticae*, 33:85–103, 1998.
139. Kevin Wallace and Graham Wrightson. Regressive merging in model elimination tableau-based theorem provers. *Journal of the IGPL*, 3(6):921–937, 1995.
140. David H. D. Warren. An abstract Prolog instruction set. Technical Report 309, SRI International, 1983.
141. David S. Warren. Memoing for logic programs. *Communications of the ACM*, 35(3):94–111, 1992.
142. Christoph Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER, version 0.42. Pages 141–145 in [101].
143. Larry Wos, D. Carson, and G. Robinson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12:536–541, 1965.
144. Larry Wos, G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *Journal of the ACM*, 14(4):698–709, 1967.
145. Lary Wos, Ross Overbeek, Ewing Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications.* McGraw-Hill, 2nd edition, 1992.
146. Hantao Zhang. SATO: an efficient propositional prover. Pages 272–275 in [100].
147. Hantao Zhang. A new method for the boolean ring based theorem proving. *Journal of Symbolic Computation*, 17(2):189–211, 1994.
148. Hantao Zhang, Maria Paola Bonacina, and Jieh Hsiang. PSATO: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21:543–560, 1996.
149. Hantao Zhang and Mark E. Stickel. Implementing the Davis-Putnam algorithm by tries. Technical Report 94-12, Department of Computer Science, University of Iowa, 1994.
150. Jian Zhang and Hantao Zhang. Generating models by SEM. Pages 308–312 in [101].