On the modelling of search in theorem proving – Towards a theory of strategy analysis*

Maria Paola Bonacina †
Department of Computer Science
The University of Iowa
Iowa City, IA 52242-1419, USA
bonacina@cs.uiowa.edu

Jieh Hsiang ‡
Department of Computer Science
National Taiwan University
Taipei, Taiwan
hsiang@csie.ntu.edu.tw

Abstract

We present a model for representing search in theorem proving. This model captures the notion of *contraction*, which has been central in some of the recent developments in theorem proving. We outline an approach to measuring the complexity of search which can be applied to analyze and evaluate the behaviour of theorem-proving strategies. Using our framework, we compare contraction-based strategies of different contraction power and show how they affect the evolution of the respective search spaces during the derivation.

1 Introduction

Theorem-proving problems usually have infinite search spaces. Theorem-proving strategies with *contraction inference rules* – rules that delete clauses, such as simplification and subsumption – are known to have in general better performance than strategies without such rules. The intuitive explanation of this experimental phenomenon is that contraction inference rules *prune the search space*, thus allowing the strategy to find a solution in shorter time. However, there has not been any formal mathematical analysis to explain how contraction rules affect the complexity of search in theorem proving. The main reason for this is the lack of formal tools to analyze the complexity of strategies involving search in an infinite search space. Indeed, the absence of such a formal method for assessing the effectiveness of inference strategies has been one of the serious weak points of artificial intelligence.

Traditional algorithm analysis is concerned mainly with the asymptotic analysis of *finite* objects. The two dominating measures are *time* and *space*, which are tightly related to each other. For instance, if a problem has a lower bound of n^2 space complexity, one cannot expect to find

*This paper is a significantly extended version of "On the representation of dynamic search spaces in theorem proving," C.-S. Yang, ed., *Proceedings of the International Computer Symposium*, 85–94, December 1996. An abstract entitled "On the notion of complexity of search in theorem proving" was presented at the *Logic Colloquium 1996*, San Sebastiàn, Spain, and appeared in Bulletin of Symbolic Logic 3(2):253–254, June 1997.

†Supported in part by the National Science Foundation with grants CCR-94-0866 and CCR-97-01508.

‡Supported in part by grant NSC 85-2221-E-002-009 of the National Science Council of the Republic of China.

an algorithm that runs in time less than n^2. Such a methodology is not suitable for analyzing search strategies in theorem proving (or in artificial intelligence in general), since the search space of a typical theorem-proving problem is infinite. Given that the search space is infinite, it is no longer meaningful to discuss average case analysis, much less worst case. However, contemporary theorem provers from different approaches (e.g., [39, 27, 23, 2, 3, 26, 44, 28, 30]) are still capable of solving problems of practical interest (e.g., [2, 1, 40, 29]). This is mainly because the existence of an infinite search space may not require an infinite amount of computation time, since it is not necessary to traverse the entire search space to find a proof. Thus, one needs a new way of analysis in order to reason about and to compare theorem-proving strategies.

The difficulty of analyzing the complexity of search in an infinite space appears in many ways. One is the absence of a relationship between the complexity of the computation and the input of the problem. In a finite problem the time and space complexities can usually be treated as functions of a measure of the input. But for first-order logic, for instance, the difficulty of finding a proof is not related to the size of the input set of clauses. A source of the problem is that a set of first-order clauses represents more than itself; it represents the infinite set of the ground instances of the clauses. Thus, any measure based on the input alone is not sufficient.

Neither is the complexity of a search strategy related to its output. In theorem proving, the output is the computed proof, if a proof is produced. The size of the proof is generally not indicative of the difficulty of *finding* the proof, since one may find a simple proof after an extensive traversal of the search space. Thus, a more accurate notion of complexity should be how difficult the *process* of finding the proof is, rather than either the input or the output of the computation.

In this paper we present an approach for the analysis and comparison of theorem proving strategies. To demonstrate how it is used, we apply our framework to analyze the effect of contraction in *forward-reasoning* strategies, such as those originated from rewriting-based methods on one hand, and the resolution-paramodulation paradigm on the other (e.g., [14, 16, 18] for general reference). We should mention that very little is known about the analysis of infinite search problems. Therefore the majority of the definitions in this paper are new.

In the rest of this section, we briefly outline the ingredients of our approach.

Representation of search space The first task is to provide an appropriate representation of the search space of a theorem-proving strategy. This task is made difficult by the existence of contraction inferences. If one considers only inference rules that generate clauses from existing clauses – called *expansion rules*, such as resolution – the well-known approach of Kowalski [24] should be sufficient. In [24], a search space is represented as an infinite graph with vertices representing the clauses and arcs representing the inferences. When the strategy generates a clause the corresponding vertex is reached. The search graph is *static* and the action of the strategy consists in *visiting* the search graph.

If the strategy contains contraction rules, the behaviour of the strategy cannot be described solely in terms of visiting a graph, because whenever the strategy performs a contraction inference a clause is deleted. The execution of the strategy *modifies* the search space. Furthermore, the modification is not merely local, because the deletion of one clause may affect the reachability

of others. It follows that the search space for a strategy with contraction is essentially *dynamic*. The main ideas in our solution of this problem are:

- we distinguish between a static part and a dynamic part and we represent them by two different components;
- we point out that the proper representation of contraction needs to employ both the static component and the dynamic component.

The static part depends only on the inference system, that is, all the applicable inferences in the search space. We represent it by a search graph with vertices labelled by clauses and arcs labelled by inference rules. The dynamic part depends also on the search plan, which decides the actual inference steps chosen during a derivation.[1] We capture it by a *marking* of the vertices of the graph. Thus, the search space is represented as a *marked search-graph*. The generation and deletion of clauses are performed on the graph by incrementing/decrementing the marking of the vertices representing the clauses involved. This approach provides a natural way of reflecting a theorem-proving derivation on the search graph, by associating a marking of the search graph to each state of the derivation.

Complexity measures The second task is to define a proper notion of complexity measures. A complexity measure is essentially a *well-founded ordering*. More precisely, one needs to take into consideration

1. the entities, such as search strategies or proofs, whose complexity is being investigated;
2. mathematical objects that are *representatives* of the aforementioned entities;
3. a well-founded ordering to compare the objects.

The phrase complexity measure in complexity theory often refers only to (2). This is because the conventional complexity measures implicitly assume the natural (well-founded) ordering on $\mathbb{N}$ (the non-negative integers). The well-founded ordering for our purpose may not be based on the ordering on $\mathbb{N}$, and therefore we need to work with a notion of complexity measure that comprises (2) and (3). For our problem, the entities are the search spaces of theorem-proving strategies, and we need to find appropriate complexity measures.

The domains In conventional analysis of algorithms, the complexity measures of time and space refer to the *history* of the computation by the algorithm from the initial state to the final state. The history of the computation is the *domain* over which the measures have meaning. A computation by a forward-reasoning theorem-proving strategy is a derivation

$S_0 \vdash S_1 \vdash \ldots S_i \vdash \ldots,$

[1] A *derivation* is the computation by a theorem-proving strategy.

where S_0 is the initial set of clauses and S_i is the set of clauses after i inference steps. Since theorem-proving derivations may not halt (the search space is infinite), the computation is open-ended and we cannot reason in terms of history from the initial to the final state. We reason in a different way: at each stage of a derivation a finite portion of the search space has been generated and an infinite portion remains to be explored. We call the former the *present* and the latter the *future* of the derivation. Pictorially, one can think of the search space as an iceberg, where the present is the emerging part and the future the part under the water. Another metaphor is the labyrinth: the present is where we are, the future is the rest of the labyrinth. In order to capture the complexity of the search problem in the *global* search space, the analysis needs to involve *both domains*, the present and the future. In more concrete terms, we need to measure the effects of the inference steps performed by the strategy up to stage k not only on the clauses generated up to stage k (the present), but also on the search space that remains to be explored after stage k (the future).

Complexity measures for theorem proving Having identified the appropriate domains for the analysis, we need to define mathematical objects (point 2 of above) that are representatives of the behaviour of a strategy on such domains. For the *present*, we consider the multiset of existing clauses at any stage k of the derivation. For the *future*, we partition the infinite search space into an infinite succession of finite *bounded search spaces*. A *bounded search space of bound* j is the space of clauses that can be reached from the input by at most j steps. Unlike a static notion such as a path length, this notion of *reachability* reflects the past actions of the strategy. This is because the inference steps may affect the reachability of clauses that have not been generated. In particular, a step that deletes a clause may make other clauses unreachable. Thus, the succession of bounded search spaces changes at every step in the derivation. In this way we can capture the effect of inferences on the future. Bounded search spaces are also characterized as multisets of clauses.

For theorem-proving strategies which assume a well-founded ordering on the set of clauses, e.g., [35], it is natural to use the multiset extension of the same ordering, which is also well-founded, as the ordering in the complexity measures for theorem proving.

Analysis of strategies For the last task of the paper, we demonstrate how our framework is applied to the analysis and comparison of different strategies with contraction. We analyze first the effects of expansion and contraction on the bounded search spaces. We prove that contraction steps contract the bounded search spaces by making redundant clauses unreachable. Thus, while expansion inferences allow the strategy to *visit* the search space, contraction inferences also enable the strategy to *prune* it. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two strategies with the same search plan and the same set of expansion inference rules, but $\mathcal{C}_2$ has a higher degree of contraction power than $\mathcal{C}_1$.[2] We compare the derivations generated by $\mathcal{C}_1$ and $\mathcal{C}_2$ from the same input. Note that the two strategies start with different search spaces, since contraction inferences are part of the search space, and $\mathcal{C}_2$ has more. We show that $\mathcal{C}_2$ eventually induces a higher reduction of the bounded search spaces. Therefore, contraction reduces the complexity of the search process according to our measures.

[2]The case where $\mathcal{C}_1$ has no contraction rules is included as a special case.

Comparison with other work The classical representation of the search space of clauses for a theorem-proving problem was given in [24]. Our work is essentially compatible with [24] for the representation of expansion inferences, and adds the representation of contraction. Most works on search strategies emphasized the design of heuristics (e.g., [32]), while studies of the complexity of theorem proving analyzed provability or the length of computed proofs (e.g., [17, 19, 20, 21, 25, 31, 34, 38, 41, 42] and [43] for a survey; see also Section 7.1 for further discussion of these works). Our problem is the complexity of search. This is also the interest of [33], that analyzed the *duplication* in the search spaces generated by most of the known theorem-proving strategies applied to propositional Horn logic. Unlike ours, the analyzed search spaces are finite, and therefore the approach in [33] may apply worst-case analysis and classical counting techniques.

Organization of the paper Section 2 contains the basic definitions. Section 3 is dedicated to the representation of the search space: the search graph, the marking and the association of a search graph to each stage of a derivation. The section is closed by a comparison with previous work on the representation problem. Section 4 is devoted to the complexity measures for theorem proving, and it includes the definition of bounded search spaces. In Section 5 we analyze the effects of inferences on the search space and in 6 we apply our complexity measures to compare the behaviour of different strategies. Section 7 contains a comparison with other approaches, a summary, and a discussion of problems. A preliminary version has been available in [13].

2 Inference rules, search plan, and derivation

In this section we give the basic definitions and notations that will be used in the paper. Given a signature Θ, we denote by $\mathcal{L}_\Theta$ the language of all the clauses on signature Θ and by $\mathcal{P}(\mathcal{L}_\Theta)$ its powerset. A *theorem-proving problem* $(S_0; \varphi_0)$ is the problem of deciding whether $S_0 \models \varphi_0$, or, refutationally, whether $S = S_0 \cup \{\neg\varphi_0\}$ is inconsistent. We consider theorem-proving problems in clausal form, where $\varphi_0 \in \mathcal{L}_\Theta$, $S_0 \in \mathcal{P}(\mathcal{L}_\Theta)$, and inconsistency is shown by deriving the empty clause $\Box$ from S.

2.1 The inference component of a strategy

Theorems are proved by means of *inference rules*. Because we are interested in the effects of inference rules on the search space, we classify them into *expansion inference rules* and *contraction inference rules*. Expansion inference rules, such as resolution and paramodulation [16], derive new clauses from the existing ones and add the new to the old. Contraction inference rules, such as tautology deletion, proper subsumption and equational simplification [16, 18], delete clauses and possibly replace them by smaller ones, according to a well-founded ordering $\succ$ on clauses. We define formally an inference rule as a function[3] that takes a tuple of premises and yields two sets, the set of clauses to be added and the set of clauses to be deleted.

[3]For simplicity, we assume that a signature is fixed and therefore an inference rule can be defined as a function. A more general definition that makes an inference rule independent from the signature by defining it as a natural transformation can be found in [15].

Definition 2.1 *Given a signature Θ, an inference rule f^n of arity n is a function $f^n: \mathcal{L}_\Theta^n \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$.*

If f^n does not apply to a tuple $\bar{x} \in \mathcal{L}_\Theta^n$, both output sets are empty. A set of inference rules I forms an *inference system* or *inference mechanism*. Let π_1 and π_2 be the projection functions $< x, y > \rightarrow x$ and $< x, y > \rightarrow y$. Given a set of clauses S, the set obtained by adding to S the clauses generated by I in one step is

$$I(S) = S \cup \{\varphi |\ \varphi \in \pi_1(f(\varphi_1, \ldots, \varphi_n))\ for\ f \in I,\ \varphi_1, \ldots, \varphi_n \in S\}.$$

We can then distinguish between expansion and contraction as follows:

Definition 2.2 *Given a well-founded ordering on clauses $(\mathcal{L}_\Theta, \succ)$, an inference rule f^n is an* expansion inference rule *if for all premises $\bar{x} \in \mathcal{L}_\Theta^n$, $\pi_2(f^n(\bar{x})) = \emptyset$; it is a* contraction inference rule with respect to $\succ$ *if $\pi_2(f^n(\bar{x})) \neq \emptyset$ for some $\bar{x}$, and whenever $\pi_1(f^n(\bar{x})) \neq \emptyset$, for all $\varphi \in \pi_1(f^n(\bar{x}))$ there exists a $\psi \in \pi_2(f^n(\bar{x}))$ such that $\psi \succ \varphi$.*

The ordering on clauses is often based on a complete simplification ordering [22] on the atoms, and then extended to clauses via the multiset ordering. Take the inference rule of (equational) *simplification* as an example. The inference step which uses the rewrite rule $f(x) \rightarrow x$ to simplify clause $P(f(f(0)))$ to $P(0)$ is formulated as

$$simplification(P(f(f(0))), f(x) \rightarrow x) = (\{P(0)\}, \{P(f(f(0)))\}).$$

This is legitimate since $P(f(f(0))) \succ P(0)$ in any complete simplification ordering.

Clauses deleted by contraction are said to be *redundant*, in the sense that they are not necessary for proving the theorem (e.g., [9, 14]). Accordingly, it is possible to justify a contraction rule by a redundancy criterion [9]. 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:

Definition 2.3 (Bachmair and Ganzinger 1992) [9] *A mapping R from sets of clauses to sets of clauses is called a* redundancy criterion *if*

1. *$S - R(S) \models R(S)$ (the criterion is sound),*
2. *if $S \subseteq S'$, then $R(S) \subseteq R(S')$ (the criterion is monotonic) and*
3. *if $(S' - S) \subseteq R(S')$, then $R(S') \subseteq R(S)$ (redundant clauses are irrelevant for establishing other redundancies).*

The first property complements the soundness of expansion (all generated clauses are logical consequences of their premises), by asking that clauses that are redundant, and therefore can be deleted by contraction, are logical consequences of the remaining clauses. The other two properties will be used in the following. Given a redundancy criterion R, a set of contraction rules I_R is associated to R, if all clauses that can be deleted by I_R with a set S are in $R(S)$,

and whenever φ is in $R(S) \cap S$, I_R can delete φ. Both R and I_R assume the same well-founded ordering. If we need to distinguish expansion and contraction rules in the inference system, we write $I = I_e \cup I_R$, meaning that I_e contains the expansion rules, I_R contains the contraction rules and the contraction rules are justified by the redundancy criterion R.

A main motivation for defining redundancy criteria and not only contraction rules is that a redundancy criterion also captures the redundancy of clauses that are not in the existing set. Indeed, note that since a clause generated from a redundant clause in S may also be redundant, $R(S)$ may not be a subset of S. An inference step that uses a redundant clause is a *redundant inference step.*[4] Keeping with the above example, since clause $P(f(f(0)))$ is redundant in any set that contains $P(0)$ and $f(x) \rightarrow x$, all inferences that use $P(f(f(0)))$ are also redundant. In addition to contraction rules, redundancy criteria may be used to refine expansion rules (e.g., [5]) to further prevent the generation of redundant clauses.

2.2 The search component of a strategy

An inference system I is usually nondeterministic in nature, since typically more than one rule in I applies to different tuples of clauses in the set. Therefore, a *theorem-proving strategy* $\mathcal{C} =< I, \Sigma >$ is given by an inference system I and a *search plan* Σ. From an operational point of view, a search plan is a mechanism that controls the application of an inference system: given a set of clauses and the inference rules, the search plan chooses the step to be executed among all possible candidates.

We define a *state* of a theorem-proving problem to be a multiset of clauses, and use *States* for the set of all possible states of the problem. A search plan chooses the next step based on the current state and the previous history, represented as a sequence of states (set $States^*$). More precisely, a search plan Σ is given by three components, a function ζ to choose the inference rule, a function ξ to choose the tuple of premises, and a function ω to detect termination. If ζ selects an inference rule of arity n, ξ selects a tuple of n premises in the database of the current state:

Definition 2.4 *A search plan Σ for a set of inference rules I is a triple $\Sigma =< \zeta, \xi, \omega >$ where*

1. $\zeta: States^* \rightarrow I$ *is the* rule-selecting function*,*
2. $\xi: States^* \times I \rightarrow \mathcal{L}^*_\Theta$ *is the* premise-selecting function*, such that if $\zeta((S_0, S_1, \ldots, S_i)) = f^n$, then $\xi((S_0, S_1, \ldots, S_i), f^n) \in S_i^n$ $(i \geq 0)$, and*
3. $\omega: States \rightarrow \{true, false\}$ *is the* termination-detecting function*, such that $\omega(S_k) = true$ $(k \geq 0)$ if and only if S_k contains the empty clause.*

Premise-selecting functions often work by sorting clauses based on some well-founded ordering, e.g. by age, by size, by a simplification ordering, and selecting minimal clauses. A theorem prover may implement more than one search plan and therefore different premise-selecting functions. The

[4] We remark that a clause generated by a redundant inference step may not be redundant since the same clause may also be generated by a non-redundant step.

evaluation function of [24] is similar to the premise-selecting function, but it is defined directly on the search space, rather than on the derivation. There is no rule-selecting function in [24], because the only inference rule is resolution.

If ζ chooses the inference rule f^n and ξ chooses the tuple of premises $\bar{x}$, the application of f^n to $\bar{x}$ gives two sets of sentences, $\pi_1(f^n(\bar{x}))$ and $\pi_2(f^n(\bar{x}))$, with the meaning that the clauses in $\pi_1(f^n(\bar{x}))$ should be added, whereas the clauses in $\pi_2(f^n(\bar{x}))$ should be deleted:

Definition 2.5 *Given a theorem-proving problem S, the* derivation *generated by a strategy $\mathcal{C} =< I, \Sigma >$, with $\Sigma =< \zeta, \xi, \omega >$, on input S is the sequence $S = S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} S_i \vdash_{\mathcal{C}} \ldots$ such that for all $i \geq 0$, if*

- $\omega(S_i) = false$,
- $\zeta((S_0, S_1, \ldots, S_i)) = f^n$, $f^n \in I$ *and*
- $\xi((S_0, S_1, \ldots, S_i), f^n) = \bar{x}$, $\bar{x} \in S_i^n$,

then $S_{i+1} = S_i \cup \pi_1(f^n(\bar{x})) - \pi_2(f^n(\bar{x}))$.

A derivation has the monotonicity property $R(S_0) \subseteq R(S_1) \subseteq \ldots$. That is, clauses that are redundant at a given stage of a derivation are also redundant at all the following stages. (This property is implied by the second and third requirements in Definition 2.3). A derivation is *successful* if $\omega(S_k) = true$ for some k, and a strategy $\mathcal{C} =< I, \Sigma >$ is *complete* if it is guaranteed to succeed whenever the input set is inconsistent. A complete strategy is a semidecision procedure which computes the partial function that maps the input state S_0 to S_k if $\omega(S_k) = true$ for some k, and is undefined otherwise. If I is *refutationally complete* (successful derivations in I exist for all inconsistent inputs) and Σ is *fair* (whenever successful derivations exist, the one constructed by Σ is successful), then $\mathcal{C}$ is complete [14]. A sufficient condition for fairness is *uniform fairness*:

Definition 2.6 (Bachmair and Ganzinger 1992) [9] *Given a set of inference rules I and a redundancy criterion R, a derivation $S_0 \vdash S_1 \vdash \ldots S_i \vdash \ldots$ is* uniformly fair *with respect to I and R if $I(S_\infty - R(S_\infty)) \subseteq \bigcup_{j \geq 0} S_j$, where $S_\infty = \bigcup_{j \geq 0} \bigcap_{i \geq j} S_i$ is the set of persistent clauses.*

In other words, all clauses that can be generated by I from persistent, non-redundant premises are generated eventually. A search plan Σ is *uniformly fair* with respect to I and R if all the derivations controlled by Σ are uniformly fair. Let $\mathcal{C} =< I_e \cup I_R, \Sigma >$ be a strategy such that I_R contains only contraction rules that delete clauses without generating any, Σ is uniformly fair with respect to I_e and R, and $\mathcal{C}$ is complete. If I'_R is a set of contraction rules that replace clauses redundant according to R by smaller clauses, the strategy $\mathcal{C}' =< I_e \cup I_R \cup I'_R, \Sigma >$ is also complete. This means that it is sufficient that Σ is uniformly fair with respect to I_e and R [9]. A search plan that always gives priority to contraction rules is an *eager-contraction* search plan, and a strategy that features contraction inference rules and an eager-contraction search plan is a *contraction-based* strategy.

3 Representation of the search space

The *search space* of a theorem-proving problem contains all the clauses that can be derived from the problem by using the inference rules:

Definition 3.1 *Given a theorem-proving problem S and an inference system I, the* closure *of S by I is the set $S_I^* = \bigcup_{k\geq 0} I^k(S)$, where $I^0(S) = S$ and $I^k(S) = I(I^{k-1}(S))$ for all $k \geq 1$.*

If the system I contains both expansion and contraction rules, the closure contains all the clauses that can be generated by rules of either type. If S is inconsistent and I is refutationally complete, then S_I^* contains the empty clause.

We represent the search space as a *search graph* with vertices labelled by clauses in S_I^* and arcs labelled by inference rules. Since inference steps generally have multiple premises, and possibly multiple consequences, the search graph will be a *hypergraph*. For the labels of the vertices we choose not to distinguish between *variants*: all variants of a clause will be associated to the same vertex. Given a signature Θ, we denote by $\mathcal{L}_\Theta/\stackrel{\bullet}{=}$ the quotient set of $\mathcal{L}_\Theta$ with respect to the relation $\stackrel{\bullet}{=}$ of equivalence up to variable renaming:

Definition 3.2 *Given a signature Θ, a theorem proving problem S on Θ, a set of inference rules I and a graph (V, E), an* arc-labelling function *is a function $h\colon E \to I$, and a* vertex-labelling function *is an injective function $l\colon V \to \mathcal{L}_\Theta/\stackrel{\bullet}{=}$, whose restriction $l\colon V \to S_I^*/\stackrel{\bullet}{=}$ is a bijection.*

Thus, $l(v)$ is a representative of an equivalence class of variants. In the following we refer to $l(v)$ as a clause, meaning implicitly a representative of a class of variants. A vertex-labelling function is required to be injective, so that only one vertex corresponds to each clause. It is surjective only on the closure S_I^*, since all the clauses in the search space have a vertex, but this is not necessarily true for all the clauses in the language.

The search graph will have a hyperarc for every applicable inference:

Definition 3.3 *Given a theorem-proving problem S on signature Θ and a set of inference rules I, the* search space *induced by S and I is represented by the* search graph *$G(S_I^*) = (V, E, l, h)$, where V is the set of vertices, l is a vertex-labelling function $l\colon V \to \mathcal{L}_\Theta/\stackrel{\bullet}{=}$, h is an arc-labelling function $h\colon E \to I$ and if*

$$f^n(\varphi_1, \ldots, \varphi_n) = (\{\psi_1, \ldots, \psi_m\}, \{\alpha_1, \ldots, \alpha_p\})$$

E contains a hyperarc

$$e = (v_1, \ldots, v_k; w_1, \ldots, w_p; u_1, \ldots, u_m),$$

where

- *$h(e) = f^n$,*
- *$v_1, \ldots, v_k$ are the vertices labelled by those premises that are not deleted, i.e. $l(v_j) = \varphi_j$ and $\varphi_j \notin \{\alpha_1, \ldots, \alpha_p\}$, for all j, $1 \leq j \leq k$, where $1 \leq k \leq n$,*

- *$w_1, \ldots, w_p$ are the vertices labelled by the deleted clauses, i.e. $l(w_j) = \alpha_j$, for all j, $1 \leq j \leq p$, and*
- *$u_1, \ldots, u_m$ are the vertices labelled by the generated clauses, i.e. $l(u_j) = \psi_j$, for all j, $1 \leq j \leq m$.*

In order to simplify the notation, we use interchangeably vertices and their labels, e.g., v and φ if $\varphi = l(v)$. Without loss of generality, we consider hyperarcs in the form $(v_1, \ldots, v_n; w; u)$, where at most one clause is added or deleted. A hyperarc in the general form can be replaced by multiple hyperarcs of the simpler type. If the inference rule f^n is an expansion rule, no clause is deleted, and therefore the hyperarc is written $(v_1, \ldots, v_n; u)$. For instance, in resolution, $\psi = l(u)$ is the resolvent resulting from parents $\varphi_1, \ldots, \varphi_n$. This representation applies similarly to other expansion inference rules such as *hyperresolution* and *paramodulation* [16]. Thus, our representation of expansion is basically compatible with the classical representation of resolution (e.g., [24]). Figure 1 shows examples of hyperarcs. Arrows are used for the contraction hyperarcs.

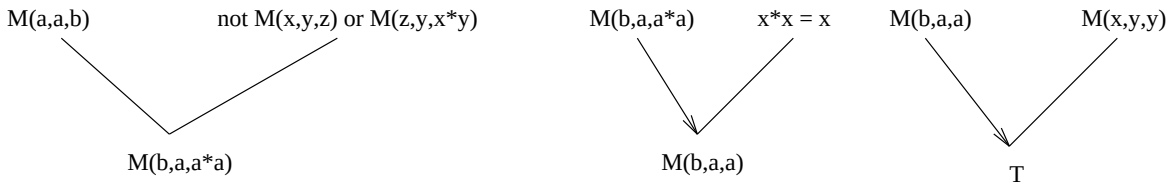

Figure 1: From left to right: a binary resolution hyperarc, a simplification hyperarc, and a subsumption hyperarc.

3.1 Contraction inferences in the search graph

The general form of a contraction hyperarc is $(v_1, \ldots, v_n; w; u)$, where $\varphi = l(w)$ is replaced by $\varphi' = l(u)$, $\varphi \succ \varphi'$ in the ordering on clauses, and the clauses of $v_1, \ldots, v_n$ justify the step. We show how this scheme applies to some of the most frequently used contraction rules. We consider first simplification inference rules:

- The *simplification* of a clause φ to φ' by an equation $s \simeq t$ is represented by the hyperarc $(v; w; u)$, where v, w and u are the vertices of $s \simeq t$, φ and φ', respectively, and $\varphi \succ \varphi'$. The center picture in Figure 1 is an example. *Clausal simplification* of φ to φ' by a unit clause L is represented similarly.
- The *normalization* of a clause φ, by repeated applications of simplification, is represented by a contraction hyperarc $(v_1, \ldots, v_n; w; u)$, where $v_1, \ldots, v_n$ are labelled by the equations applied as simplifiers, w by φ, u by its normal form φ', and $\varphi \succ \varphi'$.

For contraction inference rules that merely delete clauses, we assume that there is a dummy clause *true*, such that $true \prec \varphi$ for all φ, and a vertex T in the search graph labelled by *true*. Deletion of a clause is represented as replacement by *true*, meaning that a clause is deleted because it is trivially true in the context of the other clauses in the set:

- The *subsumption* of a clause φ by another clause ψ is represented by the hyperarc $(v; w; \mathsf{T})$, where $l(v) = \psi$ and $l(w) = \varphi$. The right picture in Figure 1 is an example. If ψ and φ are variants, the hyperarc has the form $(v; v; \mathsf{T})$, because variants are associated to the same vertex. The same representation applies to *functional subsumption* of equations [22].

- The *deletion* of a *tautology* φ is visualized as a hyperarc $(\_; v; \mathsf{T})$, which can be abbreviated as $(v; \mathsf{T})$, where v is the vertex of φ. *Deletion* of a trivial equation $s \simeq s$ is represented in the same way.

Figure 2 shows how inference hyperarcs of different types combine in a graph.

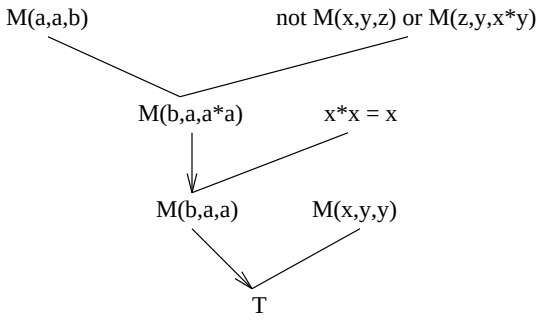

Figure 2: The combination of the hyperarcs of Figure 1 in a graph.

3.2 Representation of deletions: the marking function

The representation of contraction inferences by contraction hyperarcs captures the capability of contraction rules to generate new clauses, e.g. by simplification, and allows us to have a uniform representation of all the inferences. It is not sufficient, however, to represent contraction, because *only clauses that have been generated can be deleted*, and *only those deletions that are actually selected by the search plan need to be represented*. In other words, the representation of contraction cannot be separated from the representation of the selection of inferences by the search plan.

In order to represent the selection of inferences by the search plan, we enrich the search graph with a *marking*: a marking of arcs and a marking of vertices. The marking of an arc tells how many times the inference represented by the arc has been executed. The marking of a vertex represents its *status*: the status of a vertex in the search graph is positive, if its clause has been generated and is being kept, is 0 if it has not been generated, and is negative if it has been generated and then deleted. Generations and deletions of clauses will be represented by setting the statuses of their vertices accordingly. Since different variants of a clause may be generated during a derivation, we also use the marking to indicate the number of multiple occurrences:

Definition 3.4 *A* marked search-graph (V, E, l, h, s, c) *is given by*

- *a search graph* (V, E, l, h),

- *a* vertex-marking function $s: V \rightarrow Z$ *from vertices to integers, defined as follows:*

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

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

The search graph represents the *static* structure of all the possible inferences in the search space. Such structure depends only on the logic of the problem, i.e., the input clauses and the inference rules. The marking, on the other hand, represents the *dynamic* behaviour of the search space. While expansion inference rules can be represented by using only the static structure, both components of the representation are necessary for the proper representation of contraction.

3.3 The evolution of the search space during the derivation

The marking allows us to represent a derivation on the search graph. We define first the *preconditions* for the execution of an inference step:

Definition 3.5 *Given a marked search-graph* (V, E, l, h, s, c)*, the inference step represented by the hyperarc* $(v_1, \ldots, v_n; v_{n+1}; v_{n+2}) \in E$ *is* enabled*, if* $s(v_j) > 0$ *for all* $j \leq n+1$*, and* $s(v_{n+1}) > 1$ *if* $v_{n+1} \in \{v_1, \ldots, v_n\}$*.*

The *post-conditions* of the execution of an inference step are the following:

Definition 3.6 *Given a marked search-graph* (V, E, l, h, s, c)*, the* successor marking *induced by the execution of an enabled hyperarc* $e = (v_1, \ldots, v_n; v_{n+1}; v_{n+2})$ *in* E *is given by:*

- *the successor vertex-marking function* $succ_e(s)$ *defined as:*

$$succ_e(s)(v) = \begin{cases} s(v) - 1 & \text{if } v = v_{n+1} \text{ and } s(v) > 1 \\ -1 & \text{if } v = v_{n+1} \text{ and } s(v) = 1 \\ 1 & \text{if } v = v_{n+2} \text{ and } s(v) = -1 \\ s(v) + 1 & \text{if } v = v_{n+2} \text{ and } s(v) \geq 0 \\ s(v) & \text{otherwise.} \end{cases}$$

- *the successor arc-marking function* $succ_e(c)$ *defined as:*

$$succ_e(c)(a) = \begin{cases} c(a) + 1 & \text{if } a = e, \\ c(a) & \text{otherwise.} \end{cases}$$

An actual derivation can be reproduced by starting from the marking associated with the initial state and modifying the marking according to the derivation steps. In other words, changes in the marking on the search graph mirror actual generations and deletions of clauses. Since the steps in the derivation are chosen by the search plan Σ, the corresponding transformations of the marking of the search graph represent the effect of the search plan on the search graph:

Definition 3.7 *Let S be a theorem-proving problem and $\mathcal{C} =< I, \Sigma >$ be a theorem-proving strategy. Given the search graph $G(S_I^*) = (V, E, l, h)$, the* succession of markings associated to the derivation $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} S_i \vdash_{\mathcal{C}} \ldots$, *where $S_0 = S$, is the succession $\{(s_i, c_i)\}_{i \geq 0}$ such that:*

- *for all $e \in E$, $c_0(e) = 0$; for all $v \in V$, $s_0(v) = 1$ if clause $\varphi = l(v)$ is in S_0, $s_0(v) = 0$ otherwise, and*
- *for all $i \geq 0$, if e is the enabled hyperarc selected by Σ at stage i, then $s_{i+1} = succ_e(s_i)$ and $c_{i+1} = succ_e(c_i)$.*

It follows that each state S_i of a derivation 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. If we consider only these clauses, we obtain a subgraph:

Definition 3.8 *Given a marked search-graph $G = (V, E, l, h, s, c)$, the* active part of the search space *in G is the marked search-graph $G^+ = (V^+, E^+, l, h, s, c)$, where $V^+ = \{v \mid v \in V,\ s(v) > 0\}$ and E^+ is the restriction of E to V^+.*

If we consider all the clauses with non-zero marking, we obtain the portion of the search space generated by the strategy:

Definition 3.9 *Given a marked search-graph $G = (V, E, l, h, s, c)$, the* generated search space *in G is the marked search-graph $G^* = (V^*, E^*, l, h, s, c)$, where $V^* = \{v \mid v \in V,\ s(v) \neq 0\}$ and E^* is the restriction of E to V^*.*

Then, G_i^* is the search space generated by a strategy $\mathcal{C}$ *up to* stage i. For simplicity, we shall use G_i^* also to denote the multiset of clauses with non-zero marking in G_i. We remark that G_i^* is finite for all $i \geq 0$, regardless of whether the derivation terminates. If the derivation halts at some stage k, the search space generated up to stage k is the search space generated by the strategy during the entire derivation.

3.4 Discussion and comparison with related work

In this section we compare our representation of the search space with other related notions. Before we proceed, we need to outline some of the better known classical approaches.

In artificial intelligence, search space is usually described by a *graph of states*. The nodes are labelled by elements in *States*, where the root is labelled by the initial state and a node with label S has a child with label S', if S' can be derived from S by one application of an inference rule in I. In this framework, a path, or *I-path*, represents a derivation. The nodes labelled by a successful state (such as a multiset containing the empty clause) are *successful nodes*, and a path from the root to a successful node is a *successful path*. Since different sequences of inference steps may lead to the same state, this structure is usually treated as a graph, although it can also be represented as a tree if distinct nodes are allowed to have the same label. We called it *I-tree*

in [14] to emphasize that its structure is determined by the inference system. Indeed, the I-tree with root S represents all the possible derivations by the inference system I from S. The search plan enters in the picture as the mechanism that selects at each step the successor of the current node. The repeated application of the search plan Σ selects an I-path in the I-tree, the unique derivation generated by the strategy $< I; \Sigma >$.

A different framework was proposed in [33], in which the search space is represented as a *tree of states*. In this approach, what constitutes a state depends on the strategy. For *subgoal-reduction* strategies, such as those based on *model elimination*, a state is a goal clause. An arc connects two states S and S' if the goal clause of S' can be derived from the goal clause of S by a subgoal-reduction step. Thus, the search space is basically the tree of subgoals that is the typical choice for representing search by subgoal-reduction strategies. For strategies based on (forward) resolution, a state is a set of clauses, and there is an arc from state S to state S' if S' is the union of S and *all* the resolvents in S. This representation has two major problems. First, since each arc (representing an atomic action) generates and adds all resolvents, one cannot deal with search plans which select resolvents one at a time. Second, and for the same reason, one cannot consider contraction inference rules since the contractions of certain clauses may prevent some resolvents from being generated.

As a consequence, all strategies studied in [33] are either expansion-only strategies or subgoal-reduction strategies. Contraction is considered only after the main analysis has been completed. Because the type of analysis is worst-case analysis (in propositional logic), one is only interested in determining whether contraction rules, such as subsumption and clausal simplification, may apply to the sets of clauses used to establish the upper bounds for the expansion-only strategies.

The representation using tree of states is useful in describing global properties of strategies. In [14], we used it to study refutational completeness of the inference mechanism and fairness of the search plan. In [33], it is used to study the total size of the finite search spaces of Horn propositional problems. On the other hand, "searching for paths in trees is not general enough to represent the searches needed in automatic theorem proving" (quoted from [24] where it is attributed to [36]). This assessment is even more valid for theorem proving with contraction. The tree of states does not show the effect of contraction, because in such a representation the effect of contraction remains hidden in the states, e.g., the sets of clauses, that are the labels of the nodes of the tree. Thus, it does not emerge that contraction makes the search space dynamic.

In order to capture the effects of contraction and the role of the search plan, we use a *search graph* of clauses in the style of [24]. Our search graph is motivated by [24], and is intended to further represent the behaviour of deletions by contraction. At the first glance, one may incline to think that the deletion of a clause can be done within the search graph representation of [24] by simply deleting its associated vertex, all its arcs, and the portion of graph that is made disconnected as a consequence. This naive approach fails to recognize that while the generation of clauses by the inference system can be completely represented independently of the search plan, the representation of contraction is intertwined with the representation of the dynamic behaviour determined by the search plan. Therefore, in our approach we represent the generative power of the inference system by the static structure of the search graph (including generation by contraction rules, e.g., by simplification), and represent deletions by using the marking function.

The difficulty of representing contraction purely at the graph level is concretely visible if one considers the case of a clause that is generated, deleted, and then generated again as a different variant. If deletion of a clause were represented by deleting its vertex, it would be necessary to have a distinct vertex for each variant of a clause. This appears to be a cumbersome representation, since there are infinitely many variants of a clause. Also, it would not capture the basic knowledge that all variants are logically the same clause. In our approach, all variants of a clause are associated to the same vertex, and deletions are represented by changing the marking. In this way we capture both the logical knowledge that all variants are the same clause and the theorem-proving knowledge that theorem provers generate multiple variants and delete them by subsumption.

The choice of having a unique vertex for all occurrences and variants of a clause makes our search graph different from [24] also in terms of the representation of the generation of clauses. The search graph in [24] captured the fact that a clause can be derived in different ways by allowing distinct vertices to be labelled by the same clause (or variants thereof). In our search graph, different generations of a clause are represented by different paths to the same vertex. The representation of inferences is modified accordingly. In [24], an inference step was represented by a set of arcs, each arc connecting a premise to the consequence. The set of arcs forming a step was identified as the set of arcs pointing to the consequence of the step. If all occurrences of a clause are associated to the same vertex, it is not possible to identify a step by its consequence, because different steps have the same consequence. Our solution is to use hyperarcs, so that the premises of each step are grouped together, and each step is uniquely represented by a hyperarc.

Although the main motivation for our choices is the representation of contraction, there are other additional advantages. First, our model allows us to establish a formal relationship between the macro-level of the I-tree and the micro-level of the search graph: since Definition 3.7 could apply to any path in the I-tree, a marked search-graph can be associated to any node in the I-tree. Intuitively, if one could look at a node in the I-tree with a magnifying lense, one would see the corresponding marked search-graph. The marked search-graph is a "blow-up" of a node in the I-tree. Second, our approach appears suitable for extensions to parallel search. By keeping all information about a clause in one vertex, it makes it easier to keep track in the presence of several deductive processes which may generate/delete the same clause in different ways and at different times. Structural deletions in the graph, on the other hand, seem quite difficult in the parallel case, since a clause may be deleted by a deductive process and kept by another.

4 Measures of complexity of search

In this section, we define complexity measures for search in an infinite search space.

The first issue we emphasize is that the search space in consideration is usually infinite. Therefore the conventional measures such as the number of vertices or maximum path length are no longer sufficient to accurately represent the behaviour of the strategies.

A *complexity measure* is a pair $(\mathcal{A}, >)$, where $\mathcal{A}$ is the set of mathematical objects that are representatives of the entities whose complexity is being measured, and $>$ is a well-founded ordering

to compare the elements in $\mathcal{A}$.

Let $G_i = (V, E, l, h, s_i, c_i)$ be the marked search-graph associated to stage i during a derivation. In order to capture the complexity of the search process, one needs to reason on two domains: the portion of the search graph that has been visited, and the portion of the search graph that is still undiscovered. We call the former the *present*, because it represents the present state of the process, and the latter the *future*, because it contains all the possible continuations of the process. The present is completely characterized by G_i itself, from which we can extract measures such as the multiset S_i of existing clauses. Multisets can be compared by the multiset extension $\succ_{mul}$ of the given well-founded ordering $\succ$ on clauses. The crucial, and difficult, point is to define complexity measures for the future. This entails two parts. First we need to identify metrics for the future space. Such metrics should be related to the behaviour of the strategies that we aim at describing. Since the purpose of a (refutational) strategy is to generate eventually the empty clause, this metric will be a notion of *distance* of clauses in a search graph. The clauses in the future will be characterized by their distance relative to the current state. This notion of distance will be *dynamic*: the inference steps executed by the strategy modify the distances of the clauses in the future. In this way we can capture the effect of the inferences on the portion of the search graph that has not been generated. The second part is to identify a proper *finite* portion of the infinite unexplored search graph, where the behaviour of the strategies can be observed.

4.1 Dynamic distance

A notion of distance in graphs is usually defined in terms of path length. Since a search graph is a hypergraph, the notion of path is replaced by a notion of *ancestor-graph* of a vertex v:

Definition 4.1 *Let $G = (V, E, l, h)$ be a search graph. For all $v \in V$,*

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

An ancestor-graph of v represents a sequence of inferences, or a *generation-path*, that generates its associated clause φ from the input clauses. The clauses associated to the vertices of an ancestor-graph t of φ are its ancestors on the generation-path represented by t. Since the ancestor-graph of a clause in a search graph is not unique, we use $at_G(v)$ (or $at_G(\varphi)$) to denote the set of the ancestor-graphs of v in G.

Given $G_i = (V, E, l, h, s_i, c_i)$, a clause φ that has not been generated, and an ancestor-graph t of φ, we are interested in measuring the *distance that has been covered* and the *distance that has not been covered*, to reach φ on the generation-path represented by t. The *global distance* to φ will be the sum of these two distances. As a first approximation, the *distance that has not been covered* is given by the number of clauses in t that have not been generated (zero marking). Dually, the *distance that has been covered* is given by the number of clauses in t that have been generated

(non-zero marking). We use *p-distance* for the "distance that has been covered," *f-distance* for the "distance that has not been covered," and *g-distance* for the "global distance." In order to define these notions accurately, we need to distinguish those ancestors that are *relevant* to the generation of φ in the current marking:

Definition 4.2 *Let $G = (V, E, l, h, s, c)$ be a marked search-graph, v be a vertex in V such that $l(v) = \varphi$, and $t = (v; e; (t_1, \ldots, t_n, t_{n+1}))$ be an ancestor-graph of v, where $e = (v_1, \ldots, v_n; v_{n+1}; v)$. Then a vertex $w \in t$, $w \neq v$, is* relevant to v in t *if*

- *either $w \in \{v_1, \ldots, v_n, v_{n+1}\}$ and $c(e) = 0$,*
- *or w is relevant to v_i in t_i for some i, $1 \leq i \leq n+1$.*

We denote by $Rev_G(t)$ the set of relevant vertices of t in G. Figures 3 and 4 give a few examples of ancestor-graphs, showing how the execution of inference steps affects the markings and the relevance of ancestors. Boldface numbers attached to clauses and hyperarcs respectively represent their markings.

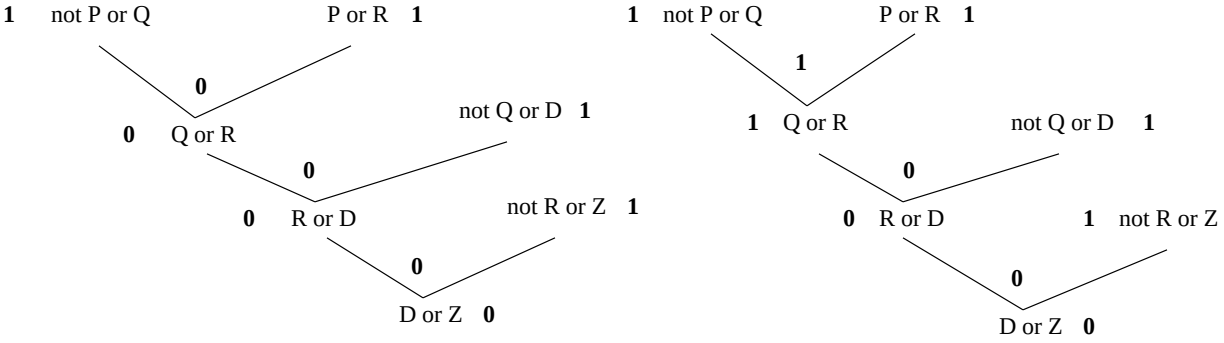

Figure 3: Ancestor-graph t_1 before and after the generation of $Q \vee R$.

Example 4.1 *The left-hand side of Figure 3 shows the ancestor-graph t_1 of $D \vee Z$. Initially all ancestors are relevant. If the resolution arc that generates $Q \vee R$ is executed, the marking is modified as shown in the right-hand side. After the step, the ancestors $\neg P \vee Q$ and $P \vee R$ are no longer relevant.*

Example 4.2 *The left-hand side in Figure 4 contains two ancestor-graphs, t_2 of $f(e, f(e, y)) \simeq f(e, y)$ and t_3 of $f(g(g(x)), y) \simeq f(x, y)$. The ancestor-graph t_2 of $f(e, f(e, y)) \simeq f(e, y)$ includes, from the bottom up, $f(e, f(e, y)) \simeq f(e, y)$, $g(e) \simeq e$ and $f(g(x), f(x, y)) \simeq f(e, y)$. The ancestor-graph t_3 of $f(g(g(x)), y) \simeq f(x, y)$ includes, from the bottom up, $f(g(g(x)), y) \simeq f(x, y)$, $f(g(x), f(x, y)) \simeq y$, $f(g(x), f(x, y)) \simeq f(e, y)$ and $f(e, x) \simeq x$. Equation $f(g(g(x)), y) \simeq f(x, y)$ is generated by superposing $f(g(x), f(x, y)) \simeq y$ into itself. Initially all ancestors are relevant in both graphs. On the right we have the two ancestor-graphs after the step that simplifies $f(g(x), f(x, y)) \simeq f(e, y)$ to $f(g(x), f(x, y)) \simeq y$ using $f(e, x) \simeq x$ is executed. Note that the ancestors $f(g(x), f(x, y)) \simeq f(e, y)$ and $f(e, x) \simeq x$ are no longer relevant to $f(g(g(x)), y) \simeq f(x, y)$.*

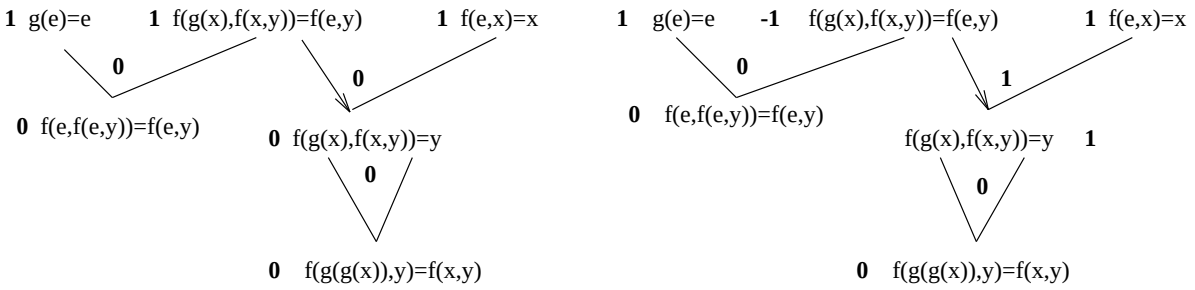

Figure 4: Ancestor-graphs t_2 and t_3 before and after the simplification of $f(g(x), f(x, y)) \simeq f(e, y)$.

We can now define the distance of a clause φ on an ancestor-graph $t \in at_G(\varphi)$. The f-distance will be given by the number of ancestors in t that have not been generated. Note that all such ancestors are relevant, because if w is an ancestor of v and $s(w) = 0$, the arc connecting w to v cannot have been executed. If some relevant ancestor of φ in t is deleted, φ is not reachable via t and therefore the f-distance of φ on t is infinite. The distance of the deleted clause itself obviously also becomes infinite:

Definition 4.3 *Given a marked search-graph $G = (V, E, l, h, s, c)$ and a clause φ, for all $t \in at_G(\varphi)$, we define:*

- *The* p-distance of φ on t *is* $pdist_G(t) = \mid \{w \mid w \in t, s(w) \neq 0\} \mid$.
- *The* f-distance of φ on t *is*

$$fdist_G(t) = \begin{cases} \infty & \text{if } s(\varphi) < 0 \text{ or } \exists w \in Rev_G(t),\ s(w) < 0, \\ \mid \{w \mid w \in t, s(w) = 0\} \mid & \text{otherwise.} \end{cases}$$

- *The* g-distance of φ on t *is* $gdist_G(t) = pdist_G(t) + fdist_G(t)$.

The f-distance of φ in G *is* $fdist_G(\varphi) = min\{fdist_G(t) \mid t \in at_G(\varphi)\}$. *The* g-distance of φ in G *is* $gdist_G(\varphi) = min\{gdist_G(t) \mid t \in at_G(\varphi)\}$.

This notion of distance is *dynamic* because it depends on the marking, and therefore it evolves with the marking during a derivation. The p-distance measures the portion of an ancestor-graph that has been *visited*. The f-distance measures *reachability*. If the f-distance of a clause φ is infinite on all its ancestor-graphs, then $fdist_G(\varphi) = \infty$, and φ is *unreachable*. If $fdist_G(\varphi)$ is finite, then φ is reachable. A positive f-distance measures the distance between the current state and φ. In particular, if $0 < fdist_G(t) < \infty$, $fdist_G(t)$ measures the amount of work that needs to be done to reach φ from the current state by traversing the path t. For instance, if φ is a *candidate* for generation at the next step, its f-distance is 1 on at least one ancestor-graph t of φ. This means that all ancestors of φ in t have already been generated and the hyperarc that generates φ in t is enabled. If a clause φ has been generated (reached), then $fdist_G(\varphi) = 0$. Remark that a generated clause may have positive or even infinite f-distance on ancestor-graphs other than the one traversed to reach it. Also, a deleted clause may be generated again, making

its f-distance and those of its descendants finite again. The g-distance keeps into account both the work that has been already done and the work that needs to be done. If the f-distance becomes infinite because of deletions of needed ancestors, the g-distance also becomes infinite.

We conclude this section by showing how the above definitions apply to several examples.

Example 4.3 *In the left-hand side of Figure 3 the p-distance of* $D \vee Z$ *is* 4 *and the f-distance is* 3. *After* $Q \vee R$ *is generated, the p-distance becomes* 5 *and the f-distance becomes* 2. *This signifies that one step has been made towards reaching* $D \vee Z$. *If* $\neg P \vee Q$ *is deleted later, the distance to* $D \vee Z$ *is unaffected, because* $\neg P \vee Q$ *is not relevant after the generation of* $Q \vee R$. *This reflects the fact that* $\neg P \vee Q$ *is no longer needed to reach* $D \vee Z$ *if* $Q \vee R$ *has been generated. If we consider* $Q \vee R$ *itself, we have that its p-distance is* 2 *and its f-distance is* 1 *in the left-hand side of Figure 3. After it is generated, its p-distance is* 3 *and its f-distance is* 0.

Example 4.4 *The p-distance and f-distance of* $f(g(g(x)), y) \simeq f(x, y)$ *on ancestor-graph* t_3 *in Figure 4 were originally both equal to* 2. *After the simplification step is executed, the p-distance becomes* 3 *and the f-distance becomes* 1. *Note that the f-distance does not become infinite, because* $f(g(x), f(x, y)) \simeq f(e, y)$ *is not relevant after the simplification step. This reflects the fact that the deletion of* $f(g(x), f(x, y)) \simeq f(e, y)$ *does not prevent the generation of* $f(g(g(x)), y) \simeq f(x, y)$ *on this path. On the contrary, it is part of a step towards the generation of* $f(g(g(x)), y) \simeq f(x, y)$. *On the other hand, the f-distance of* $f(e, f(e, y)) \simeq f(e, y)$ *on ancestor-graph* t_2 *was originally* 1, *but it becomes infinite after* $f(g(x), f(x, y)) \simeq f(e, y)$ *is deleted by simplification.*

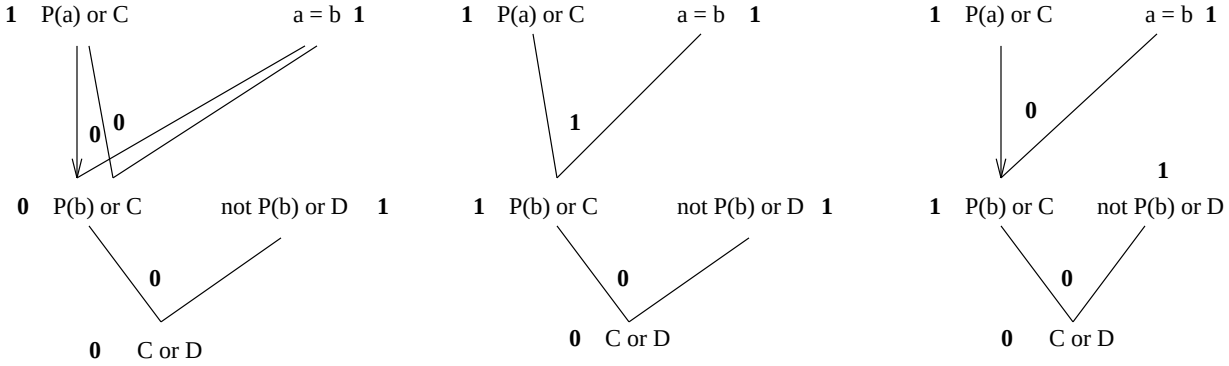

Figure 5: From left to right, ancestor-graphs t_4 and t_5 of $C \vee D$ prior to any step, t_4 if the paramodulation step is selected, t_5 if the paramodulation step is selected.

Example 4.5 *The leftmost picture of Figure 5 contains two ancestor graphs,* t_4 *and* t_5, *of* $C \vee D$. *In* t_4, $a \simeq b$ *paramodulates into* $P(a) \vee C$ *to generate* $P(b) \vee C$, *which then resolves with* $\neg P(b) \vee D$ *to generate* $C \vee D$. *In* t_5, $a \simeq b$ *simplifies* $P(a) \vee C$ *to* $P(b) \vee C$ *(assuming* $a \succ b$*), which then resolves with* $\neg P(b) \vee D$ *to generate* $C \vee D$. *Such a graph may occur in the search space of an inference system that features both paramodulation and simplification. The f-distance of* $C \vee D$ *is* 2 *on both* t_4 *and* t_5. *If the paramodulation step in* t_4 *is performed, the situation evolves as shown in Figure 5. The f-distance of* $C \vee D$ *reduces to* 1 *on both ancestor-graphs. On the other hand,*

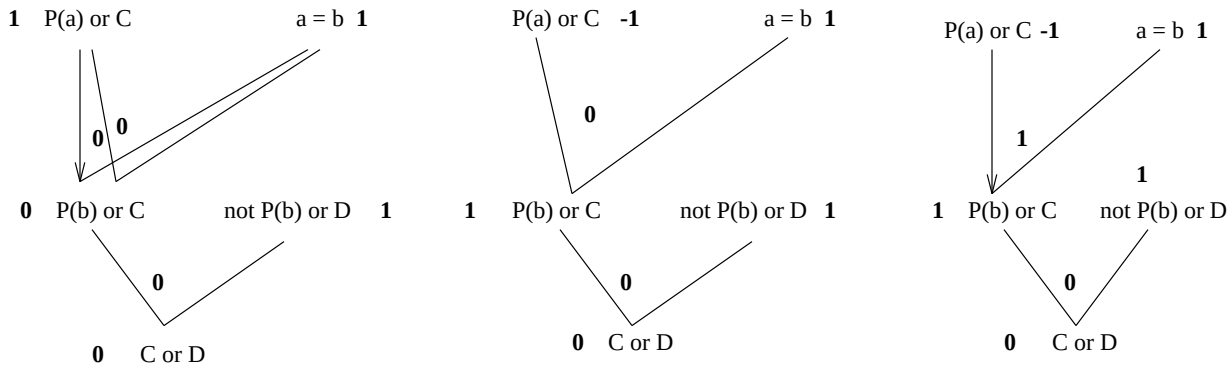

Figure 6: From left to right, ancestor-graphs t_4 and t_5 of $C \vee D$ prior to any step, t_4 if the simplification step is selected, t_5 if the simplification step is selected.

if we execute first the simplification step in t_5, the situation evolves as shown in Figure 6. The f-distance of $C \vee D$ on t_5 reduces to 1, but the f-distance of $C \vee D$ on t_4 becomes infinite, because the marking of $P(a) \vee C$ is -1 and $P(a) \vee C$ is relevant. This expresses the fact that the deletion of $P(a) \vee C$ makes the paramodulation step impossible. The f-distance to $C \vee D$ in the graph is 1 (the minimum between 1 and ∞) as $C \vee D$ is only one step away.

Example 4.6 *Similar considerations apply to clause $P(b) \vee C$ itself in Figures 5 and 6. Initially, $P(b) \vee C$ has f-distance 1 in both t_4 and t_5. If the paramodulation step is executed, the f-distance of $P(b) \vee C$ reduces to 0 in both t_4 and t_5. If the simplification step is executed later, the marking of $P(b) \vee C$ becomes 2 (two variants present) and its f-distance remains 0. On the other hand, if the simplification step is executed first, the f-distance of $P(b) \vee C$ reduces to 0 in t_5, but it is infinite in t_4. This expresses the fact that a second variant of $P(b) \vee C$ cannot be generated by the paramodulation step in t_4.*

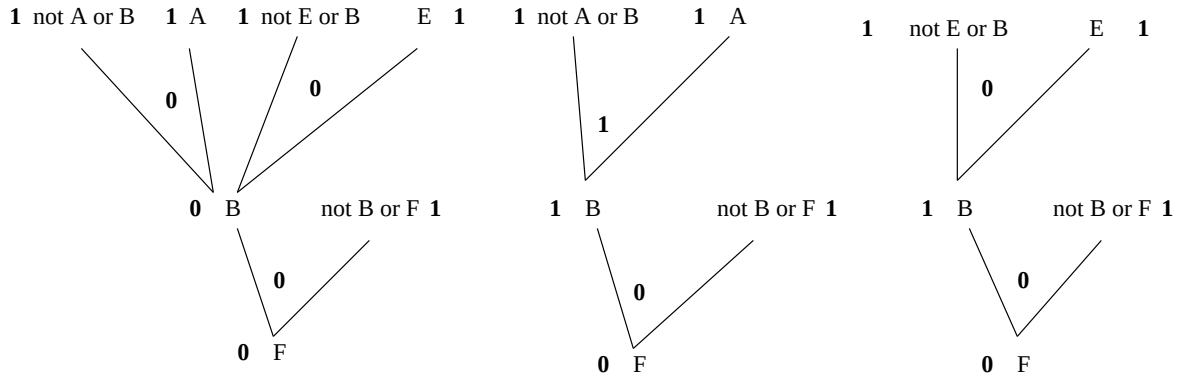

Figure 7: From left to right, ancestor-graphs t_6 and t_7 of F prior to any step, t_6 if resolution of $\neg A \vee B$ and A is selected, t_7 if resolution of $\neg A \vee B$ and A is selected.

Example 4.7 *Figure 7 shows on the right ancestor-graphs t_6 and t_7 of F. The two ancestor-graphs differ in that t_6 includes the resolution step generating B from $\neg A \vee B$ and A, whereas*

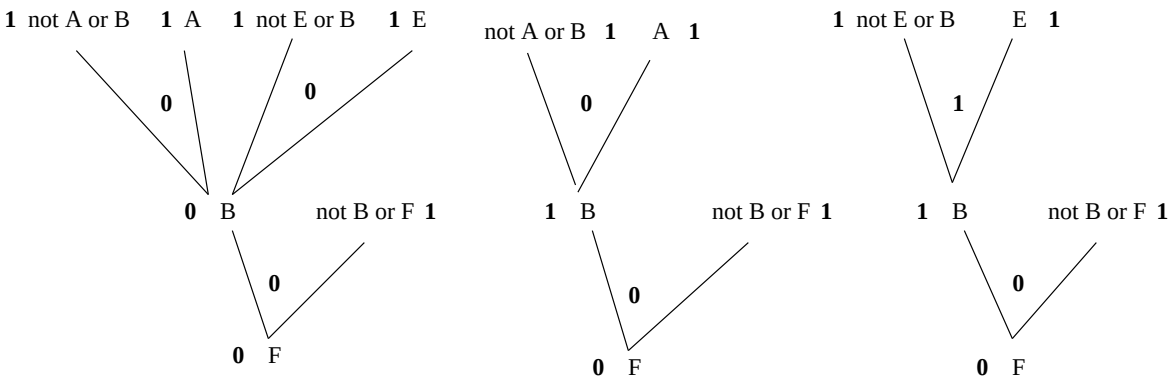

Figure 8: From left to right, ancestor-graphs t_6 and t_7 of F prior to any step, t_6 if resolution of $\neg E \vee B$ and E is selected, t_7 if resolution of $\neg E \vee B$ and E is selected.

t_7 includes the resolution step generating B from $\neg E \vee B$ and E. Starting from the indicated marking, if the resolution step generating B from $\neg A \vee B$ and A is executed, we have the situation shown in Figure 7. The f-distance of F reduces from 2 to 1 on both ancestor-graphs. If A is later deleted, the f-distance of F on t_6 is unaffected. On the other hand, if E is later deleted, the f-distance of F on t_7 becomes infinite. Of course, the f-distance of F in the graph remains 1 (the minimum between 1 and ∞). The situation is symmetric if the other resolution step is executed first as shown in Figure 8.

4.2 The bounded search spaces

We now consider the portion of search graph that contains all the clauses whose distance is within a certain bound. Note that although the entire search graph is infinite, *the search graph within bounded distance is finite*. In other words, the notion of distance allows us to "slice" the infinite search graph into finite layers:

Definition 4.4 *Given a marked search-graph $G = (V, E, l, h, s, c)$, for all $j > 0$, the* bounded search space within distance j *is the multiset of clauses*

$$space(G, j) = \sum_{v \in V, v \neq \mathsf{T}} mul_G(v, j) \cdot l(v)$$

where $mul_G(v, j) = |\{t \mid t \in at_G(v), 0 < gdist_G(t) \leq j\}|$.

For the ease of expression, we use the representation of multisets as polynomials, with the multiplicities as coefficients. The multiplicity $mul_G(v, j)$ of $\varphi = l(v)$ is the number of ancestor-graphs of φ with g-distance not greater than j in the current marked graph. It follows that φ appears in $space(G, j)$, i.e., $mul_G(\varphi, j) > 0$, if and only if $gdist_G(\varphi) \leq j$. If $gdist_G(\varphi) > j$, then $mul_G(\varphi, j) = 0$ and φ does not appear in $space(G, j)$. Therefore, $space(G, j)$ is the finite portion of search space that contains all the clauses reachable in at most j steps.

The notion of bounded search space allows us to analyze the complexity of search by transforming the problem of search in an infinite search space into the problem of search in an infinite

succession of finite search spaces $\{space(G_i, j)\}_{i\geq 0, j>0}$. The problem of comparing the infinite search spaces of different theorem-proving strategies becomes the problem of comparing their bounded search spaces.

To summarize, complexity measures for theorem proving include the multisets of existing clauses, ordered by the multiset extension $\succ_{mul}$, and the bounded search spaces, also ordered by $\succ_{mul}$. In the next section we study the effects of inferences on these measures.

5 Analysis of the effects of inferences on the complexity measures

In this section we analyze the effects of inferences on the present and the future at any stage of a derivation. We assume throughout the section that $\mathcal{C} =< I, \Sigma >$ is a strategy, S is a theorem-proving problem, $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} S_i \vdash_{\mathcal{C}} \ldots$ is the derivation by $\mathcal{C}$ on input $S_0 = S$, G is the underlying search graph $G(S_I^*)$ and G_i is the marked search-graph associated to stage i of the derivation. Since the results of this section will be applied in Section 6 to the comparison of contraction-based strategies, we assume that Σ is an eager-contraction search plan.

5.1 The effects of inferences on the present

We start by showing that expansion steps make the multiset of present clauses greater, whereas contraction steps make it smaller:

Theorem 5.1 *Given a well-founded ordering on clauses $\succ$, for all inference steps $S_i \vdash_f S_{i+1}$,*

- *$S_{i+1} \succ_{mul} S_i$, if f is an expansion rule,*
- *$S_{i+1} \prec_{mul} S_i$, if f is a contraction rule with respect to $\succ$.*

Proof: if f is an expansion rule, $S_{i+1} = S_i \cup \{\varphi_1, \ldots, \varphi_n\}$ where $\varphi_1, \ldots, \varphi_n$ are the clauses generated by the step: hence $S_{i+1} \succ_{mul} S_i$. If f is a contraction rule, $S_{i+1} = S_i - \{\psi_1, \ldots, \psi_m\} \cup \{\varphi_1, \ldots, \varphi_n\}$ where $\varphi_1, \ldots, \varphi_n$ are generated and $\psi_1, \ldots, \psi_m$ deleted by the step. Because f is a contraction rule with respect to $\succ$, for all i, $1 \leq i \leq n$, there exists a j, $1 \leq j \leq m$, such that $\psi_j \succ \varphi_i$. It follows that $S_{i+1} \prec_{mul} S_i$. □

This proof applies to all expansion steps, regardless of whether they generate "new" clauses or variants of already existing clauses, because in either case the multiplicity of the generated clause in the multiset is incremented. Similarly, it applies to all contraction steps, including both replacement and deletion, since deletion is replacement by *true* (see Section 3.1).

In a derivation made only of expansion steps the multiset of present clauses is monotonically increasing. The active part of the search space and the generated search space coincide (i.e., $G_i^+ = G_i^*$ for all $i \geq 0$), because no clause is ever deleted. This expresses the fact that solving a problem by pure expansion consists in expanding the generated portion of the search space until it includes the empty clause. In a derivation with contraction instead, the multiset of present clauses oscillates non-monotonically.

5.2 The effects of inferences on the future

The most important effect of inferences on the future is that the deletion of a clause ψ affects not only ψ, but also every clause φ that has ψ as ancestor in an ancestor-graph t, because the distance of φ on t becomes infinite. As a consequence, some clauses may become unreachable. We first give a lemma which expresses an important property of eager contraction: once a clause is deemed redundant, it will not be used for expansion. We recall that in a strategy $\mathcal{C}$, we use I_e for its set of expansion inference rules and I_R for its set of contraction inference rules.

Lemma 5.1 *Let $\mathcal{C} =< I_e \cup I_R, \Sigma >$ be a contraction-based strategy and $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots S_i \vdash_{\mathcal{C}} \ldots$ be a derivation by $\mathcal{C}$. For all clauses φ, if $s_i(\varphi) < 0$ and $s_j(\varphi) > 0$ for some $0 < i < j$, then there exists a k, $k > j$, such that $s_k(\varphi) < 0$, and φ is not selected as premise of an expansion step at any stage h, $j \leq h \leq k$.*

Proof: if $s_i(\varphi) < 0$, it means that φ has been deleted by contraction. Thus $\varphi \in R(S_i)$, and by monotonicity of redundancy, $\varphi \in R(S_n)$ for all $n > i$. If $s_j(\varphi) > 0$ for some $j > i$, it means that φ is generated again through some other path. Because φ is still redundant and the search plan Σ is an eager-contraction plan, a step that deletes φ will be selected, so that $s_k(\varphi) < 0$ for some $k > j$. Furthermore, since Σ is an eager-contraction plan, it will not select any expansion step as long as contraction rules are applicable. Therefore, φ will not be selected as premise of an expansion step at any stage between j and k. □

The consequence of this lemma is that, in a strategy with eager contraction, we may ignore inferences that regenerate a clause which had been already deleted. Such a clause will be deleted before an expansion step is performed, therefore the impact of its regeneration on the reachability of other clauses is null. More formally, this lemma justifies the following approximation that we make in the rest of the paper: if a distance $fdist_{G_i}(t)$ is infinite, then $fdist_{G_j}(t)$ can be regarded as infinite for all $j > i$.

We start with the case of an (expansion) inference step that generates a clause ψ. All clauses that were reachable before generating ψ, are still reachable afterwards. Thus, the bounded search spaces are unchanged:

Theorem 5.2 *If $S_i \vdash S_{i+1}$ generates a clause ψ, then $\forall j > 0$, $space(G_i, j) = space(G_{i+1}, j)$.*

Proof: we distinguish two cases based on the marking of ψ.

1. If $s_i(\psi) > 0$, $S_i \vdash S_{i+1}$ generates a new variant of a clause that was already present. The marking of ψ remains positive ($s_{i+1}(\psi) > 0$), so that the distance of no clause is affected, and $\forall j > 0$, $space(G_i, j) = space(G_{i+1}, j)$.

2. If $s_i(\psi) = 0$, the marking of ψ goes from 0 to 1. For all ancestor-graphs t in G, if $\psi \notin t$, then $gdist_{G_{i+1}}(t) = gdist_{G_i}(t)$. If $\psi \in t$, then $pdist_{G_{i+1}}(t) = pdist_{G_i}(t) + 1$, $fdist_{G_{i+1}}(t) = fdist_{G_i}(t) - 1$ and $gdist_{G_{i+1}}(t) = gdist_{G_i}(t)$. It follows that for all clauses the multiplicities in the bounded search spaces remain the same: $\forall j > 0$, $space(G_i, j) = space(G_{i+1}, j)$. □

In particular, for all those clauses that have ψ as ancestor in an ancestor-graph of minimum f-distance, the f-distance in the graph decreases, i.e. $fdist_{G_{i+1}}(\varphi) = fdist_{G_i}(\varphi) - 1$. The activity of an expansion step consists only in generating clauses. By the above theorem, expansion inferences do not change the bounded search spaces. This reflects the fact that expansion inferences consist in *visiting* the search space, without modifying it.

A contraction step deletes a clause ψ or replaces it by a clause ψ'. Since pure deletion is equivalent to replacing the deleted clause by *true*, we study these two cases together. The deletion of ψ affects those clauses that have ψ as relevant ancestor. We denote this set by $D_i(\psi)$: $D_i(\psi) = \{\varphi \mid \exists t \in at_G(\varphi) \textit{ such that } \psi \in Rev_{G_i}(t)\}$. If ψ is deleted at stage i, it has negative marking at stage $i+1$. Therefore we are interested in the clauses in $D_{i+1}(\psi)$. The distances of these clauses on the ancestor-graphs that include ψ become infinite. For all bounded search spaces whose bound is sufficiently deep to include these ancestor-graphs, the multiplicity of the descendants of ψ decreases. When the multiplicity becomes 0, clauses that are in $space(G_i, j)$ are no longer in $space(G_{i+1}, j)$. Thus, the bounded search spaces are *contracted*:

Theorem 5.3 *If $S_i \vdash S_{i+1}$ replaces a clause ψ by ψ', then $\forall j > 0$, $space(G_{i+1}, j) \preceq_{mul} space(G_i, j)$. If $s_i(\psi) = 1$ and $D_{i+1}(\psi) \neq \emptyset$ then $\exists k > 0$, $\forall j \geq k$, $space(G_{i+1}, j) \prec_{mul} space(G_i, j)$.*

Proof: by Theorem 5.2, the generation of ψ' has no effect on the bounded search spaces. We distinguish two cases based on the marking of ψ.

1. If $s_i(\psi) > 1$, $S_i \vdash S_{i+1}$ deletes (e.g., by subsumption) a clause ψ such that at least another variant of ψ is left in S_{i+1}. The marking of ψ remains positive ($s_{i+1}(\psi) > 0$), so that the distance of no clause is affected, and $\forall j > 0$, $space(G_i, j) = space(G_{i+1}, j)$.

2. If $s_i(\psi) = 1$, then $s_{i+1}(\psi) = -1$. The f-distance of ψ becomes infinite. Thus, $\psi \in space(G_i, j)$ for all $j \geq gdist_{G_i}(\psi)$, but $\psi \notin space(G_{i+1}, j)$ for any j. For all other clauses (including ψ' and its descendants), if $\varphi \notin D_{i+1}(\psi)$, its distance, and therefore its multiplicities in the bounded search spaces, remains the same. If $\varphi \in D_{i+1}(\psi)$, there exist some $t \in at_G(\varphi)$ such that $fdist_{G_i}(t)$ is finite, whereas $fdist_{G_{i+1}}(t) = \infty$, because ψ has been deleted.[5] It follows that $gdist_{G_i}(t)$ is finite and $gdist_{G_{i+1}}(t) = \infty$. Graph t contributes one unit to the multiplicity of φ in $space(G_i, j)$ for all $j \geq gdist_{G_i}(t)$, but it does not contribute to the multiplicity of φ in $space(G_{i+1}, j)$ for any j. Therefore, for all $j \geq gdist_{G_i}(t)$, $mul_{G_{i+1}}(\varphi, j) < mul_{G_i}(\varphi, j)$. Since there may be more than one ancestor-graph of φ where ψ is relevant, let $p(\varphi) = min\ \{j \mid \exists t \in at_G(\varphi),\ j = gdist_{G_i}(t),\ gdist_{G_{i+1}}(t) = \infty\}$. In other words, $p(\varphi)$ is the minimum bound that is deep enough to register a change in the multiplicity of φ. It follows that for all $j \geq p(\varphi)$, $mul_{G_{i+1}}(\varphi, j) < mul_{G_i}(\varphi, j)$, and for all $j < p(\varphi)$, $mul_{G_{i+1}}(\varphi, j) = mul_{G_i}(\varphi, j)$. In summary, since for all clauses the multiplicity either reduces or remains the same, we have $\forall j > 0$, $space(G_{i+1}, j) \preceq_{mul} space(G_i, j)$. Furthermore, let $h = min\{p(\varphi) \mid \varphi \in D_{i+1}(\psi)\}$ and $k = min(h, gdist_{G_i}(\psi))$. Then $\forall j \geq k$, $space(G_{i+1}, j) \prec_{mul} space(G_i, j)$. □

[5] In particular, ψ' may be in $D_{i+1}(\psi)$ if it is descendant of ψ through some arc other than the contraction step that has been executed. The same holds for any descendant of ψ'.

For those clauses in $D_{i+1}(\psi)$ that have ψ as relevant ancestor in *all the ancestor-graphs of minimum f-distance*, the f-distance in the graph grows when ψ is contracted (i.e., $fdist_{G_{i+1}}(\varphi) > fdist_{G_i}(\varphi)$). The g-distance in the graph may grow accordingly: $gdist_{G_{i+1}}(\varphi) \geq gdist_{G_i}(\varphi)$. If a clause in $D_{i+1}(\psi)$ has ψ as a relevant ancestor in *all* of its ancestor-graphs, then its f-distance (and g-distance) becomes infinite when ψ is contracted. In other words, such a clause becomes unreachable. Thus, contraction *prunes* the future search space.

In practice, the pruning effect of contraction is usually most visible on non-trivial problems. For reasons of space, we give an example with a simple set of clauses.

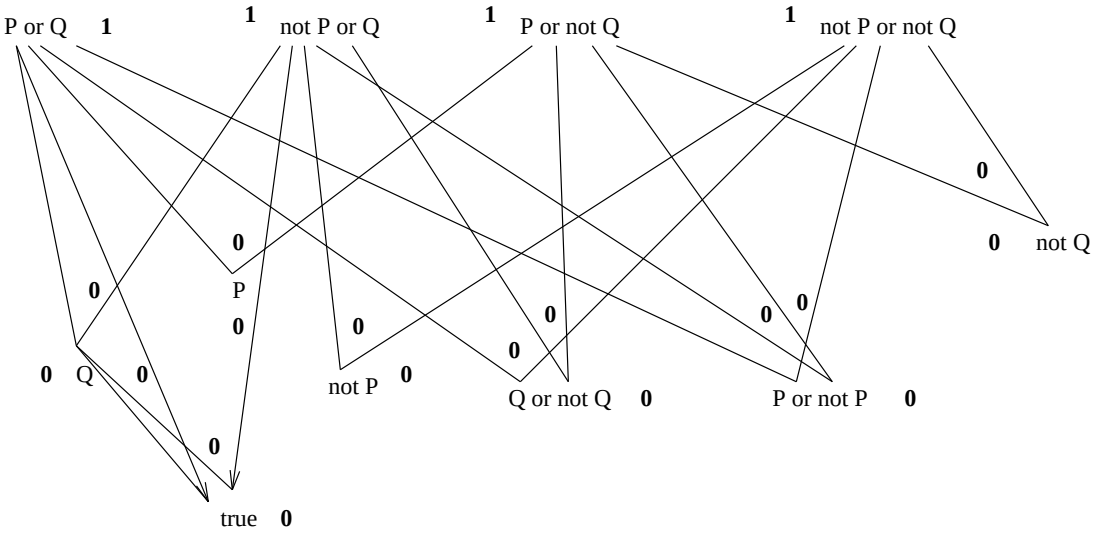

Figure 9: A fragment of the search graph for $\{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$ prior to any step.

Example 5.1 *We consider the search graph generated from* $S_0 = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$ *by resolution and subsumption. Figure 9 shows the initial state of the fragment of the graph corresponding to* $space(G_0, 3)$*, with, in addition, the clause* $true$*. We have* $space(G_0, 3) = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q, Q, P, \neg P, Q \vee \neg Q, Q \vee \neg Q, P \vee \neg P, P \vee \neg P, \neg Q\}$*. If resolution of* $P \vee Q$ *and* $\neg P \vee Q$ *is selected,* Q *is generated:* $S_1 = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q, Q\}$ *and* $space(G_1, 3) = space(G_0, 3)$*. Then, if* Q *subsumes* $P \vee Q$*, both the multiset of present clauses and the bounded search spaces are contracted:* $S_2 = \{\neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q, Q\}$ *and* $space(G_2, 3) = \{\neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q, Q, \neg P, Q \vee \neg Q, P \vee \neg P, \neg Q\}$*. In the latter,* $P \vee Q$ *is removed because it is deleted, and* P *is removed because its multiplicity has reduced from* 1 *to* 0*: the distance of* P *on the ancestor-graph that generates* P *from* $P \vee Q$ *has become infinite. For the same reason, the multiplicity of* $Q \vee \neg Q$ *and* $P \vee \neg P$ *has reduced from* 2 *to* 1*. Next, if* Q *subsumes also* $\neg P \vee Q$*, the multiplicity of* $\neg P$*,* $Q \vee \neg Q$ *and* $P \vee \neg P$ *reduces from* 1 *to* 0*:* $S_3 = \{P \vee \neg Q, \neg P \vee \neg Q, Q\}$ *and* $space(G_3, 3) = \{P \vee \neg Q, \neg P \vee \neg Q, Q, \neg Q\}$*. Clauses* $Q \vee \neg Q$ *and* $P \vee \neg P$ *have become unreachable. On the other hand,* P *and* $\neg P$ *are still reachable: if we take a larger bound and consider* $space(G_3, 5)$*, we find* P *and* $\neg P$ *in* $space(G_3, 5)$*, because of the ancestor-graphs that generate them from* Q *and* $P \vee \neg Q$*, and from* Q *and* $\neg P \vee \neg Q$*, respectively. Figure 10 shows how the state of the search graph in Figure 9 has been modified by these inferences.*

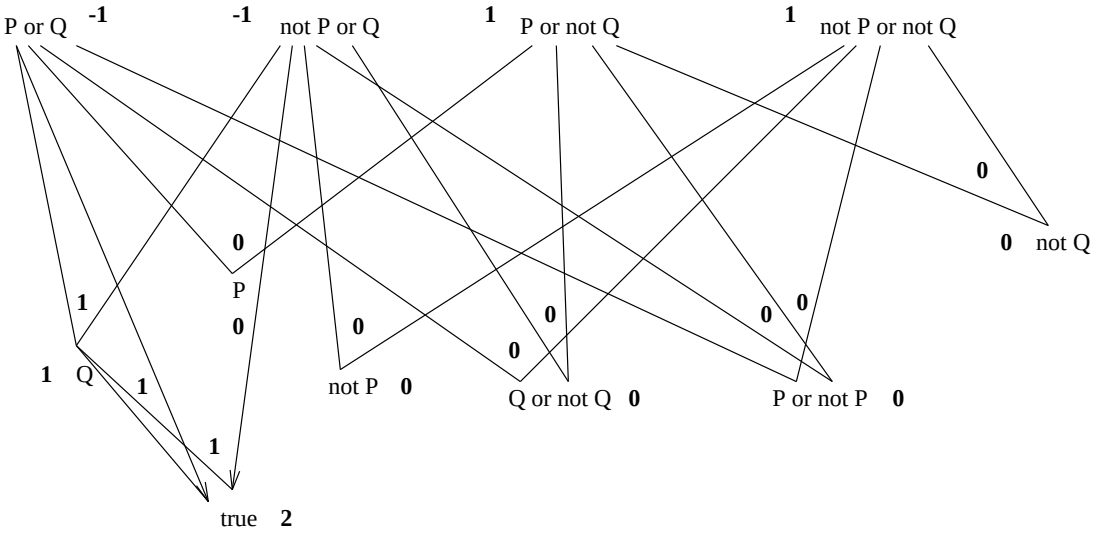

Figure 10: The fragment of search graph of Figure 9 after one resolution step and two subsumption steps.

6 Comparison of strategies

In this section we apply our approach for the measurement of search complexity to the comparison of strategies. Theorem-proving strategies may differ in many ways. As a first cut, one may distinguish between comparing strategies that have the same inference system and different search plans, and comparing strategies that have the same search plan and different inference systems. In this paper, we consider an instance of the second type of problem: we compare strategies that have the same search plan and differ solely in the contraction component of the inference system. Namely, we shall assume that one strategy has more contraction power than the other.

Comparing two strategies $\mathcal{C}_1 =< I_1, \Sigma >$ and $\mathcal{C}_2 =< I_2, \Sigma >$ with different inference systems poses the problem that given an input set of clauses S, the search spaces $G^1 = G(S^*_{I_1})$ and $G^2 = G(S^*_{I_2})$ are different in general. This is reflected by the complexity measure in that $space(G^1_0, j)$ and $space(G^2_0, j)$ are different. Therefore, we cannot compare absolute values of the complexity measures for the two strategies. We need to compare them *relative* to the different search spaces $G(S^*_{I_1})$ and $G(S^*_{I_2})$. In other words, we need to compare *variations* of the complexity measures rather than absolute values. For this purpose, we introduce the following Δ notation to represent the variation in the bounded search spaces:

$$\Delta space(G_i, j) = \sum_{v \in V, v \neq \mathsf{T}} \Delta mul_{G_i}(v, j) \cdot l(v),$$

where $\Delta mul_{G_i}(v, j) = mul_{G_0}(v, j) - mul_{G_i}(v, j)$, and $G_i = (V, E, l, h, s_i, c_i)$ is the marked search-graph associated to stage i during a derivation. Since we proved in Section 5.2 that all inferences either leave unaffected or decrease the multiplicities of clauses in the bounded search spaces, it follows that $\Delta mul_{G_i}(v, j) \geq 0$.

Then, we restrict our attention to the case where the search spaces $G(S^*_{I_1})$ and $G(S^*_{I_2})$ contain the same clauses. This is expressed by requiring that the inference systems I_1 and I_2 are *equipollent*:

Definition 6.1 *Two inference systems I_1 and I_2 are* equipollent *if $S^*_{I_1} = S^*_{I_2}$ for all theorem-proving problem S.*

Equipollence means that the two systems have equivalent generation power. If I_1 and I_2 are equipollent, then $G(S^*_{I_1})$ and $G(S^*_{I_2})$ contain the same clauses, although they have different structure in general.

We can now focus on a specific instance of the comparison problem. Let $\mathcal{C}_1 =< I_1, \Sigma >$ and $\mathcal{C}_2 =< I_2, \Sigma >$ with $I_1 = I_e \cup I_{R_1}$ and $I_2 = I_e \cup I_{R_2}$, be two complete, contraction-based strategies with the same eager-contraction search plan Σ. We assume that the two strategies have the same set of expansion rules I_e, that I_1, I_2 and I_e are equipollent, and that for all sets of clauses S, $R_1(S) \subseteq R_2(S)$. In other words, the redundancy criterion of $\mathcal{C}_2$ is more powerful than the redundancy criterion of $\mathcal{C}_1$. In particular, we assume that this condition is satisfied because $I_{R_1} \subseteq I_{R_2}$. It follows that $I_1 \subseteq I_2$. We assume further that I_e is refutationally complete with redundancy criterion R_2 (and therefore also with R_1), so that it is sufficient that Σ is uniformly fair with respect to I_e and R_2 (and with respect to I_e and R_1; see Section 2.2).

The following results compare the derivations generated by the two strategies $\mathcal{C}_1$ and $\mathcal{C}_2$ applied to the same problem S. We denote by $S^1_0 \vdash_{\mathcal{C}_1} \ldots \vdash_{\mathcal{C}_1} S^1_i \vdash_{\mathcal{C}_1} \ldots$ and $S^2_0 \vdash_{\mathcal{C}_2} \ldots \vdash_{\mathcal{C}_2} S^2_i \vdash_{\mathcal{C}_2} \ldots$, where $S^1_0 = S^2_0 = S$, the derivations generated by $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. We use G^1 for $G(S^*_{I_1})$, G^2 for $G(S^*_{I_2})$, G^1_i for the marked search-graph associated to S^1_i and G^2_k for the marked search-graph associated to S^2_k. The search graphs G^1 and G^2 are different because G^2 may contain more contraction hyperarcs.

The first lemma says that whatever is generated by one strategy is either generated or redundant for the other. This property is a consequence of uniform fairness and the equipollence of I_1, I_2 and I_e:

Lemma 6.1 *For all $j \geq 0$ and for all clauses $\varphi \in S^1_j$, there exists a $k \geq 0$ such that $\varphi \in S^2_k$ or $\varphi \in R_2(S^2_k)$. Symmetrically, for all $k \geq 0$ and for all clauses $\varphi \in S^2_k$, there exists a $j \geq 0$ such that $\varphi \in S^1_j$ or $\varphi \in R_1(S^1_j)$.*

Proof: if φ is generated by $\mathcal{C}_1$, then, since $S^*_{I_1} = S^*_{I_2} = S^*_{I_e}$, it can be generated by expansion by $\mathcal{C}_2$. If $\varphi \in I_e(S^2_\infty - R_2(S^2_\infty))$, then it will be generated in the derivation by $\mathcal{C}_2$ ($\varphi \in S^2_k$) by uniform fairness of Σ with respect to I_e and R_2. Otherwise, φ can be generated only by using clauses that are R_2-redundant or non-persistent, hence R_2-redundant, in the derivation by $\mathcal{C}_2$. Thus, φ itself is R_2-redundant ($\varphi \in R_2(S^2_k)$). The proof is the same for the symmetrical statement. □

The difference between the two strategies is made by the redundancy criteria. The second lemma shows that whatever is redundant for $\mathcal{C}_1$ is redundant for $\mathcal{C}_2$. The proof uses the hypothesis that $R_1(S) \subseteq R_2(S)$ for all S, so that it does not hold in the opposite direction:

Lemma 6.2 *For all $j \geq 0$ and for all clauses $\varphi \in R_1(S^1_j)$, there exists a $k \geq 0$ such that $\varphi \in R_2(S^2_k)$.*

Proof: let $\{\psi_1, \ldots, \psi_m\} \subseteq S^1_j$ be the smallest subset of clauses such that $\varphi \in R_1(\{\psi_1, \ldots, \psi_m\})$. By Lemma 6.1, for all i, $1 \leq i \leq m$, there exists a k_i such that $\psi_i \in S^2_{k_i} \cup R_2(S^2_{k_i})$. Because

all clauses deleted by $\mathcal{C}_2$ are redundant with respect to R_2, and R_2 is monotonic ($R_2(S_0) \subseteq R_2(S_1) \subseteq R_2(S_2) \subseteq \ldots$), if there exists a k such that $\psi \in S_k^2 \cup R_2(S_k^2)$, then for all $h > k$, $\psi \in S_h^2 \cup R_2(S_h^2)$. Thus, if we take $k = max\{k_i \mid 1 \leq i \leq m\}$, we have $\psi_i \in S_k^2 \cup R_2(S_k^2)$ for all i, $1 \leq i \leq m$. The hypothesis that for all S, $R_1(S) \subseteq R_2(S)$, and $\varphi \in R_1(\{\psi_1, \ldots, \psi_m\})$, imply $\varphi \in R_2(\{\psi_1, \ldots, \psi_m\})$, and hence $\varphi \in R_2(S_k^2 \cup R_2(S_k^2))$. Since redundant clauses are irrelevant to establish other redundancies (Property 2 in Definition 2.3 of redundancy criterion), it follows that $\varphi \in R_2(S_k^2)$. □

On the other hand, clauses that are redundant for $\mathcal{C}_2$ (and therefore are not generated or deleted after generation), may not be redundant for $\mathcal{C}_1$ (and therefore are persistent if generated), because $\mathcal{C}_2$ has a more powerful redundancy criterion.

The consequence of the difference in the redundancy criteria is that $\mathcal{C}_2$ prunes the search graph at least as much as $\mathcal{C}_1$, as shown by the following two lemmas:

Lemma 6.3 *For all $i \geq 0$, for all clauses φ in G^1, if $s_i^1(\varphi) = -1$, there exists a $k \geq 0$ such that either $s_k^2(\varphi) = -1$, or $s_k^2(\varphi) = 0$ and $fdist_{G_k^2}(\varphi) = \infty$.*

Proof: since the two strategies have equipollent inference systems, φ is also in G^2. If $s_i^1(\varphi) = -1$, it means that φ has been deleted by $\mathcal{C}_1$. Since deleted clauses are redundant, $\varphi \in R_1(S_i^1)$. By Lemma 6.2, there exists an $h \geq 0$ such that $\varphi \in R_2(S_h^2)$. Let h be the smallest such index. By monotonicity of R_2, $\varphi \in R_2(S_j^2)$ for all $j \geq h$. If $\mathcal{C}_2$ generates φ at some stage (smaller or greater than h is irrelevant), there exists a $j \geq h$ such that $\varphi \in S_j^2 \cap R_2(S_j^2)$. Then φ can be deleted by a contraction rule in I_{R_2}. Since Σ is an eager-contraction search plan, there exists a $k > j$ such that $s_k^2(\varphi) = -1$. If $\mathcal{C}_2$ does not generate φ, since $\mathcal{C}_2$ is uniformly fair, it means that φ is made unreachable by deleting other clauses, that is, relevant ancestors on all its ancestor-graphs: there exists a $k \geq 0$ such that $s_k^2(\varphi) = 0$ and $fdist_{G_k^2}(\varphi) = \infty$. □

Lemma 6.4 *For all $i \geq 0$, for all clauses φ in G^1 and for all ancestor-graphs $t \in at_{G^1}(\varphi)$, if $fdist_{G_i^1}(t) = \infty$, there exists a $k \geq 0$ such that $fdist_{G_k^2}(t) = \infty$.*

Proof: since the two strategies have equipollent inference systems, φ is also in G^2. Because $I_1 \subseteq I_2$, t is an ancestor-graph of φ also in G^2: $t \in at_{G^2}(\varphi)$. If $fdist_{G_i^1}(t) = \infty$, either $s_i^1(\varphi) = -1$ or there is a relevant ancestor ψ of φ in t such that $s_i^1(\psi) = -1$. In the first case, we apply Lemma 6.3 to φ itself: there exists a $k \geq 0$ such that either $s_k^2(\varphi) = -1$, or $s_k^2(\varphi) = 0$ and $fdist_{G_k^2}(\varphi) = \infty$. It follows that $fdist_{G_k^2}(\varphi) = \infty$ and, in particular, $fdist_{G_k^2}(t) = \infty$. In the second case, we apply Lemma 6.3 to the ancestor ψ: there exists a $k \geq 0$ such that either $s_k^2(\psi) = -1$, or $s_k^2(\psi) = 0$ and $fdist_{G_k^2}(\psi) = \infty$.

- If $s_k^2(\psi) = -1$, since we know that $\mathcal{C}_1$ deletes ψ when it is relevant to φ in t, and the two strategies have the same eager-contraction search plan, also $\mathcal{C}_2$ deletes ψ when it is relevant to φ in t. It follows that $fdist_{G_k^2}(t) = \infty$.
- If $s_k^2(\psi) = 0$ and $fdist_{G_k^2}(\psi) = \infty$, for all $t' \in at_{G^2}(\psi)$ there exists a relevant ancestor $\psi' \in Rev_{G_k^2}(t')$ such that $s_k^2(\psi') = -1$. By transitivity of the ancestorship relation, at least

one of these ancestors ψ' of ψ is an ancestor of φ in t. (Equivalently, at least one of the ancestor-graphs $t' \in at_{G^2}(\psi)$ is a subgraph of t). Since $s^2_k(\psi) = 0$, ψ is relevant to φ. Since ψ' is relevant to ψ and ψ is relevant to φ, it follows that ψ' is relevant to φ. Thus, $\psi' \in Rev_{G^2_k}(t)$ and $fdist_{G^2_k}(t) = \infty$. □

By using these lemmas, the next theorem shows that $\mathcal{C}_2$ may contract the bounded search spaces more than $\mathcal{C}_1$:

Theorem 6.1 *For all $i \geq 0$, $\exists k \geq 0$, such that $\forall j > 0,\ \Delta space(G^2_k, j) \succeq_{mul} \Delta space(G^1_i, j)$.*

Proof: by Lemmas 6.3 and 6.4, $\mathcal{C}_2$ may prune by contraction the ancestor-graphs of any clause φ at least as much as $\mathcal{C}_1$. By Theorem 5.3, pruning the ancestor-graphs of a clause by contraction reduces its multiplicity in all bounded search spaces (strictly, beyond a certain threshold of the bound). It follows that $\mathcal{C}_2$ may reduce the multiplicities of clauses at least as much as $\mathcal{C}_1$. Thus, for all $i \geq 0$, there exists a $k \geq 0$, such that for all $\varphi \in S^*_{I_1} = S^*_{I_2}$, $\forall j > 0,\ \Delta mul_{G^2_k}(\varphi, j) \geq \Delta mul_{G^1_i}(\varphi, j)$. It follows that $\forall j > 0,\ \Delta space(G^2_k, j) \succeq_{mul} \Delta space(G^1_i, j)$. □

We conclude with two corollaries that show how, under additional conditions, the higher reduction of the bounded search spaces of $\mathcal{C}_2$ translates into smaller bounded search spaces. In the first corollary, we measure the impact of contraction on the portion of the search space induced by the set of expansion rules I_e. This is relevant, because Σ is uniformly fair with respect to I_e. For this purpose, let $at^e_G(v)$ denote the set of ancestor-graphs of v that are made of expansion steps only. Then we can define a measure that counts only ancestor-graphs in $at^e_G(v)$:

$$espace(G, j) = \sum_{v \in V, v \neq \mathsf{T}} emul_G(v, j) \cdot l(v),$$

where $emul_G(v, j) = |\{t \mid t \in at^e_G(v), 0 < gdist_G(t) \leq j\}|$. Accordingly, we have

$$\Delta espace(G_i, j) = \sum_{v \in V, v \neq \mathsf{T}} \Delta emul_{G_i}(v, j) \cdot l(v),$$

where $\Delta emul_{G_i}(v, j) = emul_{G_0}(v, j) - emul_{G_i}(v, j)$. Since the two strategies have the same set of expansion inference rules, we have $at^e_{G^1}(\varphi) = at^e_{G^2}(\varphi)$, and therefore $emul_{G^1_0}(\varphi, j) = emul_{G^2_0}(\varphi, j)$, for all $\varphi \in S^*_{I_1} = S^*_{I_2}$. Thus, $espace(G^1_0, j) = espace(G^2_0, j)$ for all $j > 0$, and we have the following:

Corollary 6.1 *For all $i \geq 0$, $\exists k \geq 0$, such that $\forall j > 0,\ espace(G^2_k, j) \preceq_{mul} espace(G^1_i, j)$.*

Proof: by the same proof of Theorem 6.1, we have that for all $i \geq 0$, $\exists k \geq 0$, such that $\forall j > 0,\ \Delta espace(G^2_k, j) \succeq_{mul} \Delta espace(G^1_i, j)$. By definition, for all $j > 0$, $espace(G^1_i, j) = espace(G^1_0, j) - \Delta espace(G^1_i, j)$ and $espace(G^2_k, j) = espace(G^2_0, j) - \Delta espace(G^2_k, j)$. Since $espace(G^1_0, j) = espace(G^2_0, j)$ for all $j > 0$, the thesis follows. □

The second corollary applies to the special case where all rules in $I_2 - I_1$ are deletion rules, such as subsumption and tautology deletion. In this case, the only difference between G^1 and G^2 is

given by additional contraction arcs ending on vertex T in G^2. Since these arcs do not contribute to the ancestor-graphs of any clause different from *true*, it follows that for all $\varphi \in S^*_{I_1} = S^*_{I_2}$, and for all $j > 0$, $mul_{G^1_0}(\varphi, j) = mul_{G^2_0}(\varphi, j)$. Thus, $space(G^1_0, j) = space(G^2_0, j)$ for all $j > 0$. Therefore, under this additional assumption, we have the following:

Corollary 6.2 *For all $i \geq 0$, $\exists k \geq 0$, such that $\forall j > 0$, $space(G^2_k, j) \preceq_{mul} space(G^1_i, j)$.*

Proof: by definition, for all $j > 0$, $space(G^1_i, j) = space(G^1_0, j) - \Delta space(G^1_i, j)$ and $space(G^2_k, j) = space(G^2_0, j) - \Delta space(G^2_k, j)$. Since $space(G^1_0, j) = space(G^2_0, j)$ for all $j > 0$, the thesis follows from Theorem 6.1. □

To summarize, a strategy with a more powerful redundancy criterion induces a higher reduction of the bounded search spaces, that is, a higher reduction of search complexity. The property that contraction rules preserve completeness means that this contraction of the search space is done in such a way that the capability of the strategy to reach the empty clause is not impaired. Furthermore, since the search space is contracted, a solution may be found sooner.

7 Discussion

In this section first we discuss the relation with other works, and then we summarize our contributions. We conclude with some directions for future work.

7.1 Comparison with related work

Contraction inference rules based on well-founded orderings were developed over several years by many authors. We cite at least [22, 6, 35], and we refer to [14] for a systematic treatment and more references. The idea that contraction helps by reducing the search space in some intuitive sense appeared as early as [22]. It was intuitively clear that contraction helps not only by deleting clauses, but also because deletions prevent other clauses from being generated. The notion of redundancy was developed to capture this effect [35, 7, 11, 8, 12, 37, 9, 10, 14]. The purpose of these works as far as redundancy is concerned was to define it properly at the level of the inference system. They were not concerned with search, the measurement of search spaces, and the formal comparison of strategies. Thus, while intuitive explanations of the advantage of contraction have been known in theorem proving for a long time, they were never turned into formal results.

A main reason for this is that it was not known how to approach the problem of the infinite search space. Contraction and all other forms of redundancy control do not make the search space finite. If we delete finitely many branches in an infinite search space we still have an infinite search space: how do we compare two infinite spaces to say that one is "smaller"? This is precisely the question that our result on the bounded search spaces begins to answer. To our knowledge, ours is the first approach to formulate and address questions of this nature.

Most of the results obtained by applying classical complexity theory to theorem proving are results on the complexity of propositional proofs. This line of research originated in [41]. Since

the result by Cook and Reckhow in [17] that $NP = co\text{–}NP$ if and only if there is a polynomially bounded proof system for the classical (propositional) tautologies, the goal of the "Cook's program" has been to prove that there is no polynomially bounded proof system in order to prove that $NP \neq co\text{–}NP$. A survey of the results in this subfield of complexity theory can be found in [43].

The research that we have reported in this paper is different in many ways from this type of research in complexity theory. First, we are interested in general theorem proving, which entails working with infinite search spaces that are not considered by classical complexity theory. Second, we study theorem-proving strategies, not proof systems. A proof system as defined in [17], according to [43], is a function f such that $f(x) = y$ if x is a string representing a proof and y is a string representing the tautology proved by x. The proof system f is polynomially bounded if the length of x is polynomially bounded by the length of y. The definition of proof system captures a notion of *proof checking*, i.e., whether x is a legitimate proof and what x is a proof of, rather than the notion of *theorem proving*, or *searching for a proof*, that we work with.

The measure *proof length* is appropriate for the purpose of Cook's program, and therefore it is generally adopted as the complexity measure. On the other hand, an important part of our work has been to define complexity measures that are appropriate for the problem of searching for proofs in infinite spaces. One may remark that the size of the generated proof is a lower bound of the size of the visited search space. While this is generally true, there are two fundamental reasons why proof size is not a suitable measure for search in theorem proving. First, a proof cannot be measured until it is available. But when the theorem-proving strategy has generated a proof the theorem-proving problem has been solved and the complexity of searching for a solution has disappeared. Second, experiments in theorem proving show that it may be necessary to search a larger portion of the search space in order to find a shorter proof. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two strategies, that generate proofs P_1 and P_2 for a given problem, visiting search spaces G_1^* and G_2^*, respectively. Even if P_1 is smaller than G_1^* and P_2 is smaller than G_2^*, it may be that P_1 is greater than P_2, while G_1^* is smaller than G_2^*. In other words, comparison of search spaces cannot be reduced to comparison of proofs, because going from search spaces to proofs does not preserve in general the ordering relation.

Another existing line of research works with complexity measures based on Herbrand theorem, such as the size of the smallest unsatisfiable set of ground instances of the input clauses (e.g., [20, 31, 38]). The remarks made above for the measure proof length also apply to these measures. Until the strategy has succeeded, we do not have an unsatisfiable set of ground instances. In fact, we do not even know whether one exists. Therefore, we cannot use it to evaluate the behaviour of the strategy. The measures based on the Herbrand theorem are suitable to study lower bounds for sets of clauses, but they are not applicable to strategy analysis.

Finally, both Cook's program and the complexity studies based on Herbrand theorem seek lower bounds, e.g., exponential lower bounds for propositional proof systems. Since we are motivated by the practice of theorem proving, we are interested in upper bounds, that is, in results about the strategies. It is premature to obtain upper bounds such as those of algorithm analysis, but we can begin to compare strategies and analyze how their features affect their behaviour.

7.2 Summary

In this paper we have proposed an approach towards the analysis of theorem-proving strategies. Our motivation has been a long-standing gap between the theory and practice of theorem proving. On one hand, there are indeed theorem-proving strategies whose implementations are capable of solving significant problems. On the other hand, there is no theory of "strategy analysis," comparable to algorithm analysis, for studying the complexity of such strategies. As a consequence, theorem-proving strategies are usually evaluated in a purely empirical manner, by comparing the run-times of their implementations. This is clearly unsatisfactory, since theorem provers are complex objects made of many components, and it is difficult to establish how different components contribute to the observed results. Also, at least in principle, it is impossible to draw general conclusions through experimentation, because testing cannot be exhaustive.

Since the analysis of strategies is a new area of investigation, we had to formulate almost everything from scratch. Our model comprises the representation of search, a notion of complexity of search in infinite search spaces, and measures of complexity of search. We summarize here the most important features of this approach. The structure of the search space of a theorem-proving problem, as induced by the inference mechanism of the strategy, is represented by a search graph. For instance, if a strategy features resolution and another strategy features ordered resolution, the search graph for the former contains arcs for all resolution steps, whereas the search graph for the latter only contains arcs for the ordered resolution steps. Other refinements of expansion rules, such as the critical pair criteria of [5] can be captured in the same way. The selection of inferences by the search plan of the strategy is represented by the marking of the vertices in the hypergraph. We have pointed out that both the graph and its marking are needed to represent contraction: since contraction inferences may generate clauses, the graph needs to have arcs for the contraction steps; since only clauses that were generated may be deleted by contraction, the marking needs to be used to represent deletions. The search graph and its marking are equally important components of the representation, similar to inference mechanism and search plan in a theorem-proving strategy.

The second contribution of our approach is a notion of complexity of search that is suitable for search problems whose space is infinite. The complexity of an algorithm expresses the complexity of its computation from initial step to final step, generally as a function of a measure of its input data. A search strategy also works on a finite amount of data, but this data represents only a portion of the infinite search space that the strategy is exploring. Since the search space is infinite, derivations may not halt. Therefore, we have proposed a notion of complexity that involves two domains: the "present" and the "future." Intuitively, the former captures the complexity of the computation performed so far, and the latter captures the potential complexity of what remains to be done. At each stage k of a derivation, the complexity on the domain "present" is measured by the multisets S_k of existing clauses, and the complexity on the domain "future" is measured by the bounded search spaces $\{space(G_k, j)\}_{j\geq 0}$. These complexity measures have a few properties that we regard as important:

- The infinity that is inherent to the theorem-proving problem (infinite search space and potentially infinite computation) appears in that the succession $\{space(G_k, j)\}_{j\geq 0}$ is infinite.

However, each of the $space(G_k, j)$ is finite and therefore bounded search spaces may be compared.

- The definition of $space(G_k, j)$ is based on a dynamic notion of distance, in such a way that $space(G_k, j)$ reflects the consequences of the inferences performed up to stage k on the unexplored search space.

- All objects being compared are multisets of clauses. Multisets, rather than sets, are appropriate, because theorem-proving derivations may generate duplicates, or variants, of clauses.

- All multisets are compared by the multiset extension of a well-founded ordering on clause that is in turn the multiset extension of a well-founded ordering on atoms. This ordering plays for theorem-proving strategies the role that the natural ordering on $\mathbb{N}$ usually plays for algorithms. This reflects the fact that in theorem proving, and more generally in computation based on deduction, the underlying universe is the Herbrand base, rather than the natural numbers.

Once measures of complexity are available, we can apply them to compare different strategies independently of their implementation. In this paper, we have applied them to compare strategies with the same search plan, the same expansion rules, but different contraction capabilities, proving that strategies with higher contraction power have smaller search complexity according to the above measures.

7.3 Directions for future work

Since very little work has been done on the analysis of infinite search problems, there are many possibilities for further research. We shall mention a few.

One area for future work is the continuation of the effort to provide foundations for strategy analysis. We give an example that helps to point out the issues. Assume to have the set of clauses $\{P(a), \neg P(a), a \simeq b\}$, where $a \succ b$, and a strategy that features simplification and an eager-contraction search plan. The strategy will first reduce $P(a)$ to $P(b)$, then reduce $\neg P(a)$ to $\neg P(b)$, and finally deduce the empty clause from $P(b)$ and $\neg P(b)$. At each simplification step the multiset of present clauses and the bounded search spaces become smaller. However, the eager application of simplification does not reduce the number of executed steps. On the contrary, for this set, a strategy without simplification generates $\square$ in one step from $P(a)$ and $\neg P(a)$, and therefore performs fewer steps. On one hand, this observation is trivial, because if the empty clause can be generated in one step, it is obvious that the number of steps cannot be reduced. On the other hand, it means that a reduction of search complexity according to our measures does not necessarily imply a reduction of the number of steps for all theorem-proving problems.

Pruning the search space by contraction may not reduce the number of steps, because the contraction steps need to be included in the step count. A strategy that visits and prunes the search space is more sophisticated than a strategy that merely visits the search space. This higher sophistication may induce more work and incur cost. Similar to algorithms, the more sophisticated strategy typically will not outperform the simpler strategy on all input problems.

Since sophistication has a cost, the more sophisticated strategy will perform better on problems that are sufficiently hard that the advantage of pruning the search space offsets the disadvantage of consuming resources to prune it.

In analysis of algorithms, the above point is taken into account by working asymptotically. That is, one says that algorithm A with time complexity $f(n)$, where n is the length of the input, is better than algorithm B with time complexity $g(n)$, if there exist k and c such that for all $n \geq k$, $f(n) \leq c \cdot g(n)$. A "sufficiently hard problem" is captured by a "sufficiently long input." For first-order theorem proving, this is not possible in these terms, because no known measure of the input captures how hard the problem is. However, our observations suggest that an asymptotic approach to search complexity is needed for strategy analysis.

Search complexity as measured by our measures and time complexity are different in nature. The latter alone is not suitable for infinite search problems. In this paper we provided tools to analyze search complexity during a derivation. Future research may include investigating relationship between the search complexity and the time complexity of terminating derivations, possibly in the framework of an asymptotic analysis of search complexity.

Other directions for future investigations consist in applying a search complexity approach to other methods in theorem proving or in artificial intelligence. We feel that the main ideas in our work, such as the distinction between present and future, the notions of dynamic distance and bounded search spaces, may capture essential aspects of infinite search, and therefore may be relevant to other problems where infinite search is involved. Other aspects are not as essential, and may be modified as needed. For instance, the assumption that the state of a derivation is a single component depends on having in mind strategies that work primarily by forward reasoning, that is, generate clauses from the axioms and the negation of the theorem until an empty clause is generated.

A possible direction for future work is to apply our approach to analyze backward-reasoning strategies, that work by reducing goals to subgoals. In such a case the state of a derivation, and therefore the complexity measure on the present, may be modified to express that an essential component of the current state is the current goal. Subgoal-reduction strategies do not feature contraction. Rather, experimental studies [4] and theoretical analysis [33] showed that techniques for *lemmaizing* or *caching* can reduce the search effort for these strategies. Thus, one may analyze how such techniques may affect the complexity of search in the search space of the goals.

Another direction for future research is to extend our approach to compare strategies that have different search plans, since in this paper we have focused on strategies that differ in the inference system but have the same search plan. Finally, we plan to work towards the analysis of strategies for parallel or distributed theorem proving.

References

[1] S. Anantharaman and M. P. Bonacina. An application of automated equational reasoning to many-valued logic. In M. Okada and S. Kaplan, editors, *Proceedings of the Second International Workshop on Conditional and Typed Term Rewriting Systems (CTRS90)*, number

516 in Lecture Notes in Computer Science, pages 156–161. Springer Verlag, 1991.

[2] S. Anantharaman and J. Hsiang. Automated proofs of the Moufang identities in alternative rings. *Journal of Automated Reasoning*, 6(1):76–109, 1990.

[3] O. L. Astrachan and D. W. Loveland. METEORs: High performance theorem provers using model elimination. In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 31–60. Kluwer Academic Publisher, Dordrecht, 1991.

[4] O. L. Astrachan and M. E. Stickel. Caching and lemmaizing in model elimination theorem provers. In D. Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 224–238. Springer Verlag, 1992. Full version available as Technical Report 513, SRI International, December 1991.

[5] L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.

[6] L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume II: Rewriting Techniques, pages 1–30. Academic Press, New York, 1989.

[7] L. Bachmair and H. Ganzinger. On restrictions of ordered paramodulation with simplification. In M. E. Stickel, editor, *Proceedings of the Tenth International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 427–441. Springer Verlag, 1990.

[8] L. Bachmair and H. Ganzinger. Completion of first-order clauses with equality by strict superposition. In M. Okada and S. Kaplan, editors, *Proceedings of the Second International Workshop on Conditional and Typed Term Rewriting Systems (CTRS90)*, number 516 in Lecture Notes in Computer Science, pages 162–180. Springer Verlag, 1991.

[9] L. Bachmair and H. Ganzinger. Non-clausal resolution and superposition with selection and redundancy criteria. In A. Voronkov, editor, *Proceedings of the Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 273–284. Springer Verlag, 1992.

[10] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[11] M. P. Bonacina and J. Hsiang. Completion procedures as semidecision procedures. In M. Okada and S. Kaplan, editors, *Proceedings of the Second International Workshop on Conditional and Typed Term Rewriting Systems (CTRS90)*, number 516 in Lecture Notes in Computer Science, pages 206–232. Springer Verlag, 1991.

[12] M. P. Bonacina and J. Hsiang. On fairness of completion-based theorem proving strategies. In R. V. Book, editor, *Proceedings of the Fourth International Conference on Rewriting*

Techniques and Applications, number 488 in Lecture Notes in Computer Science, pages 348–360. Springer Verlag, 1991.

[13] M. P. Bonacina and J. Hsiang. On search in theorem proving – towards a theory of strategy analysis. Technical Report 95-05, Department of Computer Science, The University of Iowa, December 1995.

[14] M. P. Bonacina and J. Hsiang. Towards a foundation of completion procedures as semidecision procedures. *Theoretical Computer Science*, 146:199–242, 1995.

[15] M. P. Bonacina and J. Hsiang. A category-theoretic treatment of automated theorem proving. *Journal of Information Science and Engineering*, 12:101–125, 1996.

[16] C. L. Chang and R. C. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973.

[17] S. A. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

[18] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, Amsterdam, 1990.

[19] E. Eder. *Relative Complexities of First-order Calculi*. Vieweg, Braunschweig, 1992.

[20] J. Goubault. The complexity of resource-bounded first-order classical logic. In *Proceedings of the Eleventh Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lecture Notes in Computer Science*, pages 59–70. Springer Verlag, 1994.

[21] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[22] J. Hsiang and M. Rusinowitch. On word problems in equational theories. In T. Ottman, editor, *Proceedings of the Fourteenth International Conference on Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71. Springer Verlag, 1987.

[23] D. Kapur and H. Zhang. An overview of RRL: rewrite rule laboratory. In N. Dershowitz, editor, *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 513–529. Springer Verlag, 1989.

[24] R. Kowalski. Search strategies for theorem proving. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 181–201. Edinburgh University Press, 1969.

[25] R. Letz. On the polynomial transparency of resolution. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 123–129, 1993.

[26] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: a high performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[27] W. McCune. Otter 3.0 reference manual and guide. Technical Report 94/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

[28] W. McCune. 33 Basic test problems: a practical evaluation of some paramodulation strategies. In R. Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, pages 71–114. MIT Press, 1997.

[29] W. McCune. Solution of the Robbins problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997.

[30] R. Niewenhuis, Rivero J. M., and M. A. Vallejo. The Barcelona prover. *Journal of Automated Reasoning*, 18(2), 1997.

[31] V.-P. Orevkov. Lower bounds for increasing complexity of derivations after cut elimination. *Journal of Soviet Mathematics*, pages 2337–2350, 1982.

[32] J. Pearl. *Heuristics – Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley, Reading, Massachusetts, 1984.

[33] D. A. Plaisted. The search efficiency of theorem proving strategies. In A. Bundy, editor, *Proceedings of the Twelfth Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 57–71. Springer Verlag, 1994. Full version available as Technical Report of the Max Planck Institut für Informatik, MPI-I-94-233.

[34] D. A. Plaisted and A. Sattler-Klein. Proof lengths for equational completion. *Information and Computation*, 125(2):154–170, 1996.

[35] M. Rusinowitch. Theorem-proving with resolution and superposition. *Journal of Symbolic Computation*, 11(1 & 2):21–50, 1991.

[36] E. Sandewall. Concepts and methods for heuristic search. In D. E. Walker and L. N. Norton, editors, *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 199–218, 1969.

[37] R. Socher-Ambrosius. How to avoid the derivation of redundant clauses in reasoning systems. *Journal of Automated Reasoning*, 9(1):77–98, 1992.

[38] R. Statman. Lower bounds on Herbrand theorems. In *Proceedings of the American Mathematical Society*, volume 75, pages 104–107, 1979.

[39] M. E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[40] M. E. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood. Deductive composition of astronomical software from subroutine libraries. In A. Bundy, editor, *Proceedings of the Twelfth International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 341–355. Springer Verlag, 1994.

[41] G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in constructive mathematics and mathematical logic*, volume 2, pages 115–125. Consultants Bureau, New York, 1970. Reprinted in J. Siekmann and G. Wrightson (eds.), *Automation of reasoning*, Vol. 2, 466–483, Springer Verlag, New York, 1983.

[42] A. Urquhart. Hard examples for resolution. *Journal of the Association for Computing Machinery*, 34(1):209–219, 1987.

[43] A. Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1:425–467, 1995.

[44] C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER, version 0.42. In M. McRobbie and J. Slaney, editors, *Proceedings of the Thirteenth International Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 141–145. Springer Verlag, 1996.