On the Representation of Dynamic Search Spaces in Theorem Proving

Maria Paola Bonacina *
Department of Computer Science
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 Overview

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 problems 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 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 usually infinite. Given that the search space is infinite, it is no longer meaningful to discuss about average case analysis, much less worst case. However, contemporary theorem provers from different approaches are still capable of solving problems of practical interest (e.g., [1, 15, 20]). 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. One may remark that the size of the generated proof is a lower bound of the size of the visited search space.

*Supported in part by the National Science Foundation with grant CCR-94-08667.

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

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. (This remark also applies to complexity measures based on Herbrand theorem, such as the size of the smallest unsatisfiable set of ground instances of the input clauses. Until the strategy has succeeded, we do not have such a set, in fact we do not even know whether it exists. Therefore, we cannot use it to evaluate the strategy.) 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. Then P_1 may be smaller than P_2 in size even if $\mathcal{C}_1$ may have traversed a greater portion of search space in order to produce P_1. 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. 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 representation of search and the analysis of theorem-proving strategies. To demonstrate how it is used, we apply our framework to compare strategies with contraction and those without contraction. 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 [13] should be sufficient. In [13], 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*.

To solve this problem, we distinguish between a *static* part and a *dynamic* part of a search space. 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.

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 theorem-proving derivation may not halt (the search space is infinite), and therefore 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. In order to capture the complexity of the search problem in the *global* search space, we need to measure the effects of the inference steps performed by the strategy on *both domains*, present and future.

Complexity measures for theorem proving A complexity measure is essentially a *well-founded ordering* over the domain of mathematical objects being considered. Conventional complexity measures usually assume the ordering to be the natural ordering over $\mathbb{N}$. This is no longer sufficient for theorem

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

proving strategy analysis. To be more precise, the mathematical objects that are representatives of the behaviour of a strategy in the domains that we just described are as follows. 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 present state by at most j steps of inference. 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. [19], 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. Remark 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.

Comparison with other work The classical representation of the search space of clauses for a theorem-proving problem was given in [13]. Our work is essentially compatible with [13] for the representation of expansion inferences, and adds the representation of contraction. Most works on search emphasized heuristics (e.g., [16]), while studies of the complexity of theorem proving analyzed provability or the length of computed proofs (e.g., [7, 9, 10, 11, 14, 18, 21, 22] and [23] for a survey). Our problem is the complexity of search. This is also the interest of [17], 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 [17] may apply worst-case analysis and classical counting techniques.

Organization of the paper Section 2 contains the basic definitions, Section 3 the representation of the search space, Section 4 the complexity measures for theorem proving, and Sections 5 and 6 the analysis of inferences and strategies. The complete version of this paper, including all proofs and many examples, is available as [4].

2 Inference rules, search plan and derivation

A *theorem-proving problem* 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 inconsistency is shown by deriving the empty clause $\Box$ from S.

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 [6], 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 [6, 8], delete clauses and possibly replace them by smaller ones. Given a well-founded ordering $\succ$ on clauses, an inference rule is a *contraction inference rule with respect to* $\succ$ if $\psi \succ \varphi$ whenever the rule replaces a clause ψ by a clause φ. The ordering on clauses is often based on a complete simplification ordering [12] on the atoms, and then extended to clauses via the multiset ordering. For example, the inference rule of (equational) *simplification* may use the rewrite rule $f(x) \to x$ to simplify clause $P(f(f(0)))$ to $P(0)$ 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

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

proving the theorem (e.g., [3, 5]). Accordingly, it is possible to justify a contraction rule by a redundancy criterion [3]. 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. 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*[3]. Keeping with the above example, since clause $P(f(f(0)))$ is redundant in any set that contains $P(0)$ and $f(x) \to x$, all inferences that use $P(f(f(0)))$ are also redundant. 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. In addition to contraction rules, redundancy criteria may be used to refine expansion rules (e.g., [2]) to further prevent the generation of redundant clauses.

A set of inference rules forms an *inference system* or *inference mechanism*. Given an inference system I, we use I_e for its expansion rules and I_R for its contraction rules justified by redundancy criterion R. Given a set of clauses S, $I(S)$ is the set obtained by adding to S the clauses that can be generated by I in one step (by either expansion or contraction).

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, in a strategy it is necessary to couple I with a *search plan* Σ that chooses the step to be executed among all possible candidates. By applying repeatedly this selection, a strategy $\mathcal{C} =< I, \Sigma >$ generates a *derivation* $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} S_i \vdash_{\mathcal{C}} \ldots$, where for all $i \geq 0$ S_i is the *multiset* of existing clauses and represents the *state* of the derivation at stage i.

A derivation is *successful* if it generates the empty clause, and a strategy $\mathcal{C} =< I, \Sigma >$ is *complete* if it is guaranteed to succeed whenever the input set is inconsistent. 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 [5]. A sufficient condition for fairness is *uniform fairness*, that requires that all clauses that can be generated by expansion rules from premises that are persistent (i.e., not deleted during the derivation) and non-redundant, are generated eventually [3]. A search plan that always gives priority to contraction rules [12] 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]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.

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 $k \geq 1$.*

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. We denote by $S_I^*/\stackrel{\bullet}{=}$ the quotient set of S_I^* with respect to the relation $\stackrel{\bullet}{=}$ of equivalence up to variable renaming:

Definition 3.2 *Given a theorem proving problem S, 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 a bijective function $l\colon V \to S_I^*/\stackrel{\bullet}{=}$.*

Thus, $l(v)$ is a representative of an equivalence class of variants. A vertex-labelling function is required to be bijective, so that only one vertex corresponds to each clause and all the clauses in the search space have a vertex. The search graph will have a hyperarc for every applicable inference:

Definition 3.3 *Given a theorem-proving problem S 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 $l\colon V \to S_I^*/\stackrel{\bullet}{=}$, h is an arc-labelling function $h\colon E \to I$ and if $f \in I$, applied to premises $\varphi_1, \ldots, \varphi_n$, generates clauses $\psi_1, \ldots, \psi_m$ and deletes clauses $\alpha_1, \ldots, \alpha_p$, then E contains a hyperarc $e = (v_1, \ldots, v_k; w_1, \ldots, w_p; u_1, \ldots, u_m)$, where $h(e) = f^n$ and*

- *$v_1, \ldots, v_k$ are labelled by the 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 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 labelled by the generated clauses, i.e. $l(u_j) = \psi_j$, for all j, $1 \leq j \leq m$.*

Without loss of generality, we consider hyperarcs where at most one clause is added or deleted:

$$(v_1, \ldots, v_n; v_{n+1})$$

for expansion, and

$$(v_1, \ldots, v_n; v_{n+1}; v_{n+2})$$

for contraction. In the latter, $\varphi = l(v_{n+1})$ is replaced by $\varphi' = l(v_{n+2})$, $\varphi \succ \varphi'$ in the ordering on clauses, and the clauses of $v_1, \ldots, v_n$ justify the step. For contraction rules that merely delete clauses, such as subsumption, we assume that there are a dummy clause *true*, such that $true \prec \varphi$ for all φ, and a vertex T in the search graph labelled by *true*. Then, deletion of a clause is represented as replacement by *true*. In the following, we use interchangeably vertices and their labels, e.g. v and φ if $\varphi = l(v)$.

3.1 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) and

- *A* vertex-marking function $s\colon V \to Z$ *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 $c\colon E \to Z^+$ *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 possibile 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.

3.2 The evolution of the search space during the derivation

The marking allows us to represent a derivation on the search graph. In the following two definitions we assume that a marked search-graph (V, E, l, h, s, c) is given. We define first the *pre-conditions* for the execution of an inference step:

Definition 3.5 *A 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 *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:

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

and 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:*

- $\forall e \in E$, $c_0(e) = 0$; $\forall v \in V$, $s_0(v) = 1$ *if clause $\varphi = l(v)$ is in S_0, $s_0(v) = 0$ otherwise, and*
- $\forall i \geq 0$, *if e is the 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 an associated search graph $G_i = (V, E, l, h, s_i)$, and S_i is exactly the multiset of clauses with positive marking in G_i. If we consider all the clauses with non-zero marking in G_i, we obtain the multiset G_i^* of all the clauses generated up to stage i.

4 Measures of complexity of search

In this section, we define complexity measures for search in an infinite search space. 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, the analysis needs to involve both the *present*, the state represented by G_i itself, and the *future*, the portion of the search graph which is still undiscovered. The crucial, and difficult, point is to define complexity measures that take the future into account, since the unexplored search graph is infinite. To do it properly we need notions of *ancestor-graph* and *dynamic distance*, that replace conventional notions of path and path-length:

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, with $e = (v_1, \ldots, v_n; v_{n+1}; v)$. 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. Remark that all ancestor w of v such that $s(w) = 0$ is relevant, because $s(w) = 0$ implies that the arc connecting w to v cannot have been executed. An irrelevant ancestor ($c(e) \neq 0$) is one that has been already used towards the generation of φ. If such an ancestor is deleted, the f-distance of φ is unaffected. On the other hand, if a relevant ancestor of φ in t is deleted, φ is no longer reachable via t and therefore the f-distance of φ on t becomes 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* $pdist_G(t) = |\{w \mid w \in t, s(w) \neq 0\}|$.
- *The* f-distance of φ on t *is*

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

  where $n = |\{w \mid w \in t, s(w) = 0\}|$.
- *The* g-distance of φ on t *is*

$$gdist_G(t) = pdist_G(t) + fdist_G(t).$$

Then, the f-distance of φ in G *is*

$$fdist_G(\varphi) = min\{fdist_G(t) \mid t \in at_G(\varphi)\}$$

and 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. 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 that was 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 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 $gdist_G(\varphi) \leq j$. 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 $\{space(G_i, j)\}_{i \geq 0, j > 0}$. As complexity measure we take the bounded search spaces ordered by the multiset extension $\succ_{mul}$. The problem of comparing the infinite search spaces of different theorem-proving strategies becomes the problem of comparing their bounded search spaces.

5 Analysis of the effects of inferences on the complexity measures

In this section we analyze the effects of inferences on the bounded search spaces. 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.

Lemma 5.1 *Let* $\mathcal{C}$ *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*,* $j \leq h \leq k$*.*

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

This theorem 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'. The deletion of ψ affects those clauses that have ψ as relevant ancestor. We denote this set by $D_i(\psi)$: $D_i(\psi) = \{\varphi \mid \exists t \in at_G(\varphi)$ *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 5.2 *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 $\exists k > 0$, $\forall j \geq k$, $space(G_{i+1}, j) \prec_{mul} space(G_i, j)$.*

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. If a clause in $D_{i+1}(\psi)$ has ψ as a relevant ancestor in *all* of its ancestor-graphs, then its f-distance becomes infinite when ψ is contracted. In other words, such a clause becomes unreachable. Thus, contraction *prunes* the future search space.

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 $\Delta mul_{G_i}(v, j) = mul_{G_0}(v, j) - mul_{G_i}(v, j)$. Since we proved in Section 5 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, although they have different structure in general. 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.*

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. $\mathcal{C}_1$ and $\mathcal{C}_2$ have the same set of expansion rules, I_1, I_2 and I_e are equipollent, and 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 re-

spect to I_e and R_2 (and with respect to I_e and R_1) (e.g., [3]).

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

$$S_0^1 \underset{\mathcal{C}_1}{\vdash} \ldots \underset{\mathcal{C}_1}{\vdash} S_i^1 \underset{\mathcal{C}_1}{\vdash} \ldots$$

and

$$S_0^2 \underset{\mathcal{C}_2}{\vdash} \ldots \underset{\mathcal{C}_2}{\vdash} S_i^2 \underset{\mathcal{C}_2}{\vdash} \ldots,$$

where $S_0^1 = S_0^2 = S$, 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_i^1 for the marked search-graph associated to S_i^1 and G_k^2 for the marked search-graph associated to S_k^2. The search graphs G^1 and G^2 are different because G^2 may contain more contraction hyperarcs.

The first lemma 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 $\varphi \in S_j^1$, there exists a $k \geq 0$ such that $\varphi \in S_k^2$ or $\varphi \in R_2(S_k^2)$. Symmetrically, for all $k \geq 0$ and for all clauses $\varphi \in S_k^2$, there exists a $j \geq 0$ such that $\varphi \in S_j^1$ or $\varphi \in R_1(S_j^1)$.*

The difference between the two strategies is made by the redundancy criteria. The second lemma 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 $\varphi \in R_1(S_j^1)$, there exists a $k \geq 0$ such 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 $fdist_{G_k^2}(\varphi) = \infty$.*

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 $fdist_{G_k^2}(t) = \infty$.*

By using these lemmas, the theorem proves 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 $\forall j > 0$, $\Delta space(G_k^2, j) \succeq_{mul} \Delta space(G_i^1, j)$.*

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. Let $espace(G, j)$ be defined as in Definition 4.4 but counting only ancestor-graphs made solely of expansion steps. Since the two strategies have the same set of expansion inference rules, $espace(G_0^1, j) = espace(G_0^2, j)$ for all $j > 0$, and we have the following:

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

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. Since the arcs of pure deletion steps do not contribute to the ancestor-graphs, it follows that $space(G_0^1, j) = space(G_0^2, j)$ for all $j > 0$, and:

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

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.

References

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

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

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

[4] M. P. Bonacina and J. Hsiang. On the modelling of search in theorem proving – towards a theory of strategy analysis. Technical report, Department of Computer Science, The University of Iowa, 1995.

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

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

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

[8] 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, The Netherlands, 1990.

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

[10] 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.

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

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

[13] 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.

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

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

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

[17] 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, Nancy, France, 1994. Springer Verlag. Full version available as Technical Report of the Max Planck Institut für Informatik, MPI-I-94-233.

[18] D. A. Plaisted and A. Sattler-Klein. Proof lengths for equational completion. Technical Report SEKI Report SR-95-06, Fachbereich Informatik, Universität Kaiserslautern, 1995.

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

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

[21] 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.

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

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