Artificial Intelligence 57 (1992) 323–389
Elsevier

A theory of abstraction

Fausto Giunchiglia

Mechanized Reasoning Group, IRST, 38050 Povo, Trento, Italy; and DIST, University of Genoa, 16143 Genoa, Italy

Toby Walsh

Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, Scotland, UK

Received February 1990
Revised April 1992

Abstract

Giunchiglia, F. and T. Walsh, A theory of abstraction, Artificial Intelligence 57 (1992) 323–389.

Informally, abstraction can be described as the process of mapping a representation of a problem onto a new representation. The aim of this paper is to propose the beginnings of a *theory* of reasoning with abstraction which captures and generalizes most previous work in the area. The theory allows us to study the properties of abstraction mappings and provides the foundations for the mechanization of abstraction inside an abstract proof checker.

Notational conventions

$\emptyset$	the empty set,
$\sim$	an equivalence relation,
$\kappa(x)$	the equivalence class of x with respect to $\sim$,
φ, α, β	well-formed formulas (wffs),
$p(x), q(x), \ldots$	predicates,
$\top, \bot$	true and false,
$a, b, \ldots$	constants,
$x, y, \ldots$	variables,

Correspondence to: F. Giunchiglia, Mechanized Reasoning Group, IRST, 38050 Povo, Trento, Italy. E-mail: fausto@irst.it.

$\{a/x\}$	substitution of a for x,
$\Lambda, \Lambda_1, \ldots$	languages (set of wffs),
$\Omega, \Omega_1, \ldots$	axioms (set of wffs), $\Omega_1 \subseteq \Lambda_1$,
$\Delta, \Delta_1, \ldots$	deductive machineries (inference rules),
$\Sigma, \Sigma_1, \ldots$	formal systems, $\Sigma_1 = \langle \Lambda_1, \Omega_1, \Delta_1 \rangle$,
$\Pi, \Pi_1, \ldots$	proof trees,
$\mathrm{TH}(\Sigma)$	the theorems of Σ,
$\mathrm{NTH}(\Sigma)$	the nontheorems of Σ,
$f : \Sigma_1 \Rightarrow \Sigma_2$	an abstraction from Σ_1 to Σ_2,
f_Λ	the mapping function used to abstract wffs,
f^{-1}	inverse of the abstraction, f,
$\circ$	abstraction composition,
$\sqsubseteq, \preceq$	preorders on abstractions,
$\equiv$	equivalence of abstractions,
$\subseteq$	containment (of a language within another, of a formal system within another, ...).

1. Motivations and goals

Roughly speaking one can think of abstraction as the process which allows people to consider what is *relevant* and to forget a lot of *irrelevant* details which would get in the way of what they are trying to do.

The main motivation underlying the development of the work described in this paper is our belief that

> the process of abstraction is used pervasively in common sense reasoning.

Thus, for instance, while *solving the problem* of how to stack all your clothes in a suitcase, you will stack the shirts all together as if they were a single object, and not one by one. While *planning* to go from Trento to Edinburgh you will plan the flight Milan–London–Edinburgh and not consider, at least in the first instance, how to go to Milan airport. If someone is trying to give you an *explanation* of how to get to Milan airport, he will give you only the main directions and not tell you of all the little streets you will cross on the way. While driving your car on the way to the Milan airport you will try to survive the Italian traffic by behaving *analogously* to how Italians do. For instance, even if you are English, you will stay on the right-hand side of the road. Also you will drive slighly faster than you would do in Edinburgh. If, after arriving at the Milan airport, you are trying to remember, i.e. *to learn*, how you got there, you will remember a very rough outline of the things to do.

The pervasiveness of abstraction-based processes in human reasoning has been proposed, more or less explicitly, by many researchers in cognitive science. See for instance the introduction of "The nature of explanation" of [34, Section 1], and also [33,36,56].

The second motivation at the basis of this work is the fact that

> reasoning by abstraction has been used in many subfields of artificial intelligence,

often radically different both in the goals and in the methodology. Historically, the earliest and most common use of abstraction was in problem solving (see for instance Example 8.3 below and also [41,43,67,75]) and theorem proving (see for instance Examples 8.5, 8.8, 8.10, 8.14, 8.27 and 8.29 and also [81]). On the other hand, processes which can be formalized as abstraction (independently of whether the authors called them so) have been effectively used in many other areas of artificial intelligence and logic; for example in the definition of decision procedures (see for instance Example 8.12 and also [14,18,47]), in planning (see for instance Examples 8.1, 8.14, and 8.16), learning (see for instance [15,17,38,40,42,52,55]), explanation (see for instance [13]), common sense reasoning (see for instance Example 8.20), qualitative and model-based reasoning (see for instance [35,53,78,80]), approximate reasoning (see for instance Examples 8.23 and 8.24), analogy (see for instance [3,5,77]), software and hardware verification and synthesis (see for instance [1,12,49,50]).

The first main goal of the work described here is to provide the foundations of a theory of abstraction which can be used for *modeling and representing reasoning with abstraction as performed in common sense reasoning and by AI computer programs.* One of our main interests is therefore in representation theory. We claim that the theory of abstraction presented here is not only metaphysically adequate but, also and more important, epistemologically adequate [48]. In other words, this theory can be used *practically* to represent and perform reasoning by abstraction. Among other things, this allows us to use the proposed framework to explain, analyze, and compare previous work on abstraction from various subfields of artificial intelligence.

Our second main goal is to use the theory as *the basis for the development of a general environment for the use of abstraction in automated deduction* (both for proving theorems in various branches of mathematics and logic and for formalizing common sense reasoning). The most common use of abstraction in theorem proving (but also in problem solving and planning, see later) has been to abstract the goal, to prove its abstracted version, and then to use the structure of the resulting proof to help construct the proof of the original goal. This relies upon the assumption that the structure of the abstract proof is "similar" to the structure of the proof of the goal. From now

on, we will write "abstraction" to mean this intuitive idea of abstraction, and to distinguish this notion from the various formal and informal notions of abstraction used elsewhere in this paper. The abstractions described in Examples 8.5, 8.8, 8.10, 8.14, 8.27 and 8.29 below, are examples of uses of abstraction in theorem proving. However, Wos et al. [81] describe a mapping which can be formalized as an abstraction which is not an abstraction. The work done at the University of Texas at Austin by Woody Bledsoe and his group [3,5,77] on reasoning by analogy deserves special mention. In this work, the proof of an example is used as a proof plan to guide the proof of "analogous" theorems; the information extracted from the example is very similar to that extracted, in abstractions, from the proof of the abstraction of the goal.

Most of our work on the use of abstraction in theorem proving (but not all) concentrates on abstraction and on providing foundations to its use. Three important observations are in order.

The first is that the most comprehensive and theoretical work on abstraction done in the past focused on abstraction and on its use in resolution-based systems [60,61] (but see also [39,44]).

The second is about the effectiveness of abstraction in theorem proving. Abstraction was originally proposed as a very powerful and general purpose heuristic for constraining search in automated reasoning and, thus, for building more efficient theorem provers. We feel that gaining efficiency is only part of the story. There are other advantages, for instance the ability to provide high-level explanations, to learn abstract plans and to reason by analogy. Nevertheless, the issue of the efficiency of abstractions is very important and deserves a deeper analysis. Various papers in the area of problem solving and planning give experimental and theoretical results that show that the use of abstractions gives savings in efficiency [41,43,54,76]. None of them, on the other hand, shows convincingly or exhaustively enough that this is always the case. Which it is not. We have done some theoretical and experimental work in this area. This work is partially described in [25] and will be the topic of a forthcoming paper. Our experiments have shown savings in many but not all cases; the theoretical model shows that there are situations where abstraction saves time but also situations where it results in less efficiency. Describing his work on abstraction, Plaisted also notes that

> ... Although some reductions in search time [using abstraction] were obtained, usually the performance was disappointing .... [63, p. 309]

That abstraction is not always going to save us time is a consequence of many factors. For instance, in order to use any kind of abstraction, one has to invest time in the construction of the abstract space(s) (that is, in

forgetting the details). Moreover, a correct choice of the abstract spaces is vital [7,8].

All the work done so far is based on the idea of using abstraction (and in particular `abstraction`) automatically; on the other hand we believe that abstraction provides a much more useful tool if used in guided search. In this radically new use, `abstraction` guides a proof checking system. The idea is that, since the abstract space ignores irrelevant details, we can interactively build an abstract proof which is an outline of the original proof. The details are then integrated back into the outline, again interactively with the help of the proof checker, in a provably correct way. We call a proof checker which supports this kind of reasoning, an *abstract proof checker.* The description of the details of the implementation of the abstract proof checker is beyond the goals of this paper; in [26] a preliminary and high-level description of it are given. Here it is worthwhile noticing that in this way we partially avoid the problem of inefficiency (the user is in charge of control) and that this kind of reasoning is used all the time by human mathematicians: they first build an outline of the proof and then refine this outline by adding details which have been abstracted away. For instance, Polya [65] proposes a four-part strategy for solving mathematical problems: understanding the problem, devising a plan, carrying out this plan, and finally examining the solution.

As third and last observation, even if we use logic and formal deduction and our main interests lie in the automation of deduction in logical systems, the results and the ideas described here are entirely general and can be used in the study of reasoning with abstraction independently of the particular formalism used.

2. Structure of the paper

According to the motivations and goals defined in the previous section,

(1) we define a theory of abstraction;

and use it to:

(2) understand the meaning of abstraction;
(3) classify the different types of abstraction;
(4) analyze and classify past work;
(5) investigate the formal properties and the operations which can be defined on abstractions;
(6) define ways of building abstractions; and
(7) study at a preliminary level how the proof in the abstract space can be used to help find a proof of the goal.

The paper can be divided into three parts. In the first part, the basic ideas are given and the various forms of abstraction are introduced. Section 3 gives an informal characterization of abstraction. Section 4 gives the basic definitions of the different types of abstractions, Section 5 then explores how these abstractions can be used in automated reasoning. In Section 6, we consider the complications that arise from using abstraction with refutation systems. Finally, further classifications of abstractions are introduced in Section 7. This covers points (1), (2), (3) and (7) in the above list.

In the second part, consisting of just Section 8, some previous work in abstraction is presented and discussed. We view this section as one of the most important contributions of the paper. First, it should convince the reader that our proposed framework is very powerful and can capture most previous work in abstraction. Second, it allows us to give a unified view of work from many different areas carried out with many different aims. Finally, concentrating on the work in theorem proving, the topic of the final part of the paper, it allows us to point out the strengths and the weaknesses of various proposed abstractions. This covers point (4) of the above list.

To finish, the last part of the paper describes the main body of the theory of abstraction under development. In Section 9, we explore how abstractions affect consistency and discuss the problem of inconsistent abstract spaces when a consistent space maps onto an inconsistent abstract space. This problem cannot be avoided; once we commit ourselves to certain very common and useful types of abstraction, it is always possible to find a set of axioms such that the abstract space is inconsistent even though the original space is consistent. Section 10 defines the basic operations which can be performed on abstractions. In Section 11 we show how to order abstractions; in Section 12 we consider the use of hierarchies of abstractions and suggest a way that an ordered hierarchy of abstractions can be used to tackle the problem of inconsistent abstract spaces. Finally, in Section 13 we suggest a general methodology for building abstractions, and end with a section devoted to some concluding remarks and hints about possible extensions of the theory. This covers points (5) and (6) in the list above.

3. What is an abstraction?

Some of the synonyms of the word "abstract" are "brief", "synopsis" and "sketch", some of the synonyms of the verb "to abstract" are "to detach" and, also, "to separate". The intuition which comes out of this list of synonyms is that the process of abstraction is related to the process of *separating*, extracting from a representation an "abstract" representation which consists of a *brief sketch* of the original representation.

The work described in this paper concentrates on the use of abstraction in reasoning. Reasoning is about solving problems. We therefore *informally* define abstraction as:

(1) *The process of mapping a representation of a problem*, called (following historical convention [69]) the "ground" representation, *onto a new representation*, called the "abstract" representation, which:
(2) *helps deal with the problem in the original search space by preserving certain desirable properties* and
(3) *is simpler to handle* as it is constructed from the ground representation by "throwing away details".

The following sections of the first part of this paper will be devoted to the formalization of the informal definition given above. Notice that we do not formally study the requirement for simplicity (property (3)) or any more global requirement for increased efficiency (see the description of the structure of the paper in the previous section). This would require some complexity arguments which will be discussed in subsequent papers. Some work in this direction can be found in [25,45].

4. Abstractions ...

... *as mappings* ...

Informally, we have described abstraction as a mapping between representations of a problem. Thus, in giving a *theory* of abstraction, we begin with a very general method for describing representations of a problem; as is common in AI, we shall use formal systems for this purpose. Following Kleene [37], a formal system is a formal description of a theory. A formal system Σ can be (minimally) described as a set of formulas Θ (which represent the statements of the theory) written in a language Λ (which provides the basic tools for writing formulas); in other words $\Sigma = \langle \Lambda, \Theta \rangle$. Usually the language Λ is defined by the alphabet, the set of well-formed terms, and the set of well-formed formulas (wffs from now on). To simplify matters we will forget about the alphabet and well-formed terms and just say that the language is the set of wffs. The alphabet and well-formed terms are given implicitly by providing the set of wffs. Thus $\Theta \subseteq \Lambda$. An abstraction is then simply a mapping between formal systems.

Definition 4.1 (*Abstraction*). An abstraction, written $f : \Sigma_1 \Rightarrow \Sigma_2$, is a pair of formal systems $\langle \Sigma_1, \Sigma_2 \rangle$ with languages Λ_1 and Λ_2 respectively and an effective total function $f_\Lambda : \Lambda_1 \rightarrow \Lambda_2$.

We call Σ_1 the ground space and Σ_2 the abstract space. Analogously we use the words "ground" and "abstract" to label objects of the ground and abstract spaces (we talk of e.g. ground and abstract formulas, ground and abstract terms, ground and abstract goals). f_A is called the mapping function; where there is no ambiguity, we write f for f_A. Occasionally we will also extend the mapping function in the obvious way to a mapping on sets of wffs. We write $f : \Sigma_1 \Rightarrow \Sigma_2$ to mean that $\langle \Sigma_1, \Sigma_2 \rangle$ plus f_A is an abstraction. "$\Rightarrow$" is not to be read as a "function"; an abstraction is simply a pair of formal systems, each being a representation of a problem (and not the problem itself), and a mapping f_A between them. This captures property (1) of the intuitive definition of abstraction.

We require that the mapping function be total since we want to be able computationally to "translate" any wff in Σ_1 into Σ_2. We require it to be computable since, from an implementational point of view this is the only interesting case and, from a epistemological point of view, we are interested only in those aspects of reasoning which can be actually mechanized.

... preserving certain properties ...

The next step is to try to capture property (2) of our informal definition, that of *preserving certain desirable properties*. For this task, since our main interests are in automated reasoning, a more restrictive but useful notion of formal system is that of axiomatic formal system.

Definition 4.2 (*Axiomatic formal system*). A formal system Σ is a triple $\langle A, \Delta, \Omega \rangle$, where A is the *language*, Ω is the set of axioms and Δ is the *deductive machinery* of Σ.

Note that $\Omega \subseteq A$. Ω can sometime be the empty set (e.g. in natural deduction). We say that $\Omega = \Omega_{\text{Logic}} \cup \Omega_{\text{Theor}}$, where Ω_{Logic} is the set of logical axioms and Ω_{Theor} is the set of theoretic axioms. The deductive machinery is a set of inference rules; this can also be an empty set (e.g., in nondeductive databases). As with the axioms, we say that $\Delta = \Delta_{\text{Logic}} \cup \Delta_{\text{Theor}}$, where Δ_{Logic} is the set of logical inference rules and Δ_{Theor} is the set of theoretic inference rules. If $\Sigma_1 = \langle A_1, \Omega_1, \Delta_1 \rangle$ and $\Sigma_2 = \langle A_2, \Omega_2, \Delta_2 \rangle$ are two formal systems, we write $\Sigma_1 \subseteq \Sigma_2$ to indicate that $A_1 \subseteq A_2$, $\Omega_1 \subseteq \Omega_2$, and $\Delta_1 \subseteq \Delta_2$. Throughout this paper we will use standard natural deduction (ND) conventions and terminology [66]. Whenever the deductive machinery can be arbitrarily chosen, proofs will be given assuming that Δ is (a subset of) the ND rules described in [66] (not in the sequent form). From now on, when we speak of formal systems we will mean axiomatic formal systems. In some places (e.g. [51]) the deductive machinery is defined as the pair of axioms and inference rules. Our definition of formal system as a triple better fits our

interests in mechanizing logic. For the sake of simplicity, unless we explicitly state to the contrary,

> we restrict ourselves to logical languages and to first-order formal systems.

That is, we will restrict ourselves to systems where Ω_{Logic} and Δ_{Logic} are complete for first-order logic, and among them to those where $\Omega_{\text{Logic}} = \emptyset$ (which implies $\Omega = \Omega_{\text{Theor}}$) and $\Delta_{\text{Theor}} = \emptyset$ (which implies $\Delta = \Delta_{\text{Logic}}$). We sometimes lift the condition of completeness of the deductive machinery, for instance to consider propositional logic or incomplete deductive machineries. The restrictions on the axioms and on the deductive machinery is not very important and it is easy to lift; it would cause complications only in few places.

Briefly, some conventions and terminology. The set of theorems of Σ, written $\text{TH}(\Sigma)$, is the minimal set of wffs that contains the axioms and is closed under the inference rules. Σ is syntactically incomplete if there is a formula α such that $\alpha \notin \text{TH}(\Sigma)$ and $\neg\alpha \notin \text{TH}(\Sigma)$. We call it syntactically complete otherwise.

With axiomatic systems, especially in theorem proving, a central notion is that of provability. We are therefore interested in how an abstraction affects provability; that is, when Θ, the set of statements of the theory, is $\text{TH}(\Sigma)$. This notion will play an central role in the rest of the paper. Preserving provability is actually only a very weak property to demand of an abstraction. There are other desirable properties which must be captured; for instance `abstractions` are used on the basis that the structure of the abstract proof is "similar" to the structure of the ground proof. The important point is that some very interesting results can be proved, even with such weak assumptions as the preservation of provability. Our next step towards capturing property (2), that of mappings preserving desirable properties, is therefore a classification of abstractions by their effect on provability:

Definition 4.3 (*T* abstractions*). An abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is said to be a
(1) *TC abstraction* iff, for any wff α, $\alpha \in \text{TH}(\Sigma_1)$ iff $f_\Delta(\alpha) \in \text{TH}(\Sigma_2)$;
(2) *TD abstraction* iff, for any wff α, if $f_\Delta(\alpha) \in \text{TH}(\Sigma_2)$ then $\alpha \in \text{TH}(\Sigma_1)$;
(3) *TI abstraction* iff, for any wff α, if $\alpha \in \text{TH}(\Sigma_1)$ then $f_\Delta(\alpha) \in \text{TH}(\Sigma_2)$.

(Here "T" stands for "theorem", "C" for "constant", "D" for "decreasing", and "I" for "increasing".) Note that, because of the totality of f_Δ, $f_\Delta(\alpha)$ is defined for any α. Note also that a TC abstraction is both a TD abstraction and a TI abstraction. We write that an abstraction is a T* abstraction iff it is a TC abstraction or a TD abstraction or a TI abstraction.

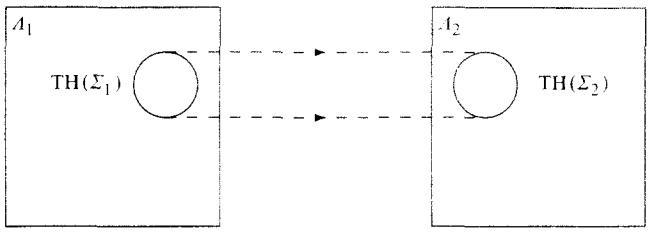

Fig. 1. TC abstractions. This is a graphical representation of a TC abstraction. In this (and the following figures) the two boxes represent the sets of wffs belonging to the two languages. The dashed lines show the behaviour of the abstraction mapping. If no dashed lines are shown there is no restriction on how a subset in the ground language, Λ_1, is mapped into the respective subset in the abstract language, Λ_2. In this figure, for example, all those wffs which are theorems of Σ_1 map onto wffs which are theorems of Σ_2.

We have characterized the statements of the theory, Θ, in terms of theoremhood. A more general concept is that of deducibility. By deducibility relation we mean here a set of ordered pairs; the first element of a pair is a subset of the language while the second element is a wff which can be deduced by assuming the members of the first element. Theoremhood is a particular case of deducibility; a wff is a theorem iff it is deducible from the empty set. Even if the paper deals with provability, most of the analysis could have been given for deducibility. We shall write $\Gamma \vdash_\Sigma \alpha$ to mean that α is deducible (derivable) in Σ from the set of wffs Γ. Thus $\vdash_\Sigma \alpha$ is an alternative notation for $\alpha \in \mathrm{TH}(\Sigma)$. The following theorem makes precise the conditions under which provability preserving abstractions also preserve deducibility.

Theorem 4.4. *If $f : \Sigma_1 \Rightarrow \Sigma_2$ is a T* abstraction, such that $f(\alpha \to \beta) = f(\alpha) \to f(\beta)$, and the deduction theorem holds in both Σ_1 and Σ_2, then f also preserves deducibility.*

Proof. We only consider TI abstractions. The other proofs are entirely analogous. If $\alpha_1, \ldots, \alpha_n \vdash_{\Sigma_1} \beta$, then, from the deduction theorem in Σ_1, $\vdash_{\Sigma_1} \alpha_1 \to \cdots \to \alpha_n \to \beta$. Since f is TI, $\vdash_{\Sigma_2} f(\alpha_1 \to \cdots \to \alpha_n \to \beta)$. But f is implication preserving. Thus $\vdash_{\Sigma_2} f(\alpha_1) \to \cdots \to f(\alpha_n) \to f(\beta)$. By modus ponens, $f(\alpha_1), \ldots, f(\alpha_n) \vdash_{\Sigma_2} f(\beta)$. □

In the remainder of the paper we shall restrict ourselves to abstractions which preserve provability; this emphasis has historical motivations, grounded in our interest in theorem proving. However, subject to the hypotheses of the above theorem, everything would also hold for deducibility.

We conclude this section with some more observations about provability preserving abstractions.

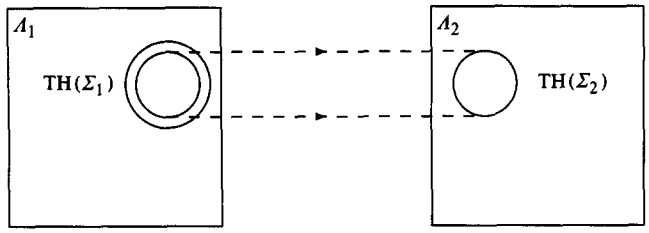

Fig. 2. TD abstractions.

TC abstractions map all the members of TH(Σ_1) onto members of TH(Σ_2) and these are the only members of TH(Σ_2). This is represented graphically in Fig. 1. For instance Herbrand's theorem suggests a TC abstraction which maps a first-order theory onto a propositional theory [30] (see Example 8.27). As another example, any mapping which maps into a new theory where logically equivalent formulas are collapsed onto a single formula can be described as a TC abstraction. TC abstractions are used in decision theory under the name of *reduction methods.* The aim is to prove the decidability/undecidability of the validity problem for certain subclasses of the first-order calculus. Having found a subclass of formula whose decidability is known, we show that there is a proof in the original formal system if and only if there is a proof in the new class. Many of these techniques are based on Herbrand's theorem (e.g. [19]).

In TD abstractions, only a subset of the members of TH(Σ_1) are mapped onto TH(Σ_2); this subset generates all the members of TH(Σ_2). This is represented graphically in Fig. 2. A trivial example of a TD abstraction is the deletion of axioms and/or inference rules. TD abstractions have been used to implement derived inference rules; this is discussed in Example 8.12. An analogous approach was also taken by Wos et al. [81] (as described in [6]).

In TI abstractions all the members of TH(Σ_1) are mapped onto a subset of TH(Σ_2). This is represented graphically in Fig. 3. In many ways, TI abstractions are dual to TD abstractions. Trivial examples of TI abstraction are adding some axioms or inference rules (such as induction), producing a nonconservative extension of a theory and so on.

... *concerning* `abstractions`

Most `abstractions` used in the past turn out to be TI (that is, TI abstractions). Indeed, because of the requirement that the abstract proof be "similar" to the ground proof, `abstractions` often satisfy much stronger requirements than just the preservation of provability. However, some mappings previously used in abstract theorem proving are not TI abstractions. In the attempt to capture the class of `abstractions`, why should we use

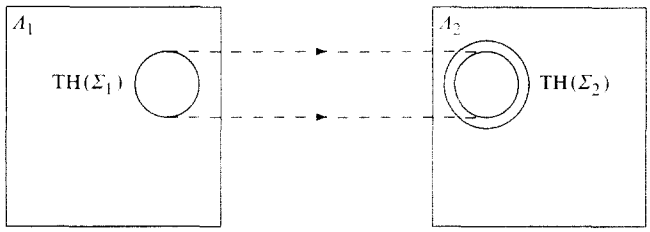

Fig. 3. TI abstractions.

TI abstractions and not TD or TC abstractions? Tenenberg, for instance, proposes TD abstractions (and others with similar properties) as a way to avoid the problem of inconsistent abstract spaces [73,74]. This is discussed in more detail in Example 8.16.

We don't often use TC abstractions as they are, in general, too strong; they do not give "simpler" proofs except in very special and limited cases. Of course this does not mean that they are useless. They are very useful, for instance, in changing the representation of a problem. However, they can only reduce the complexity in a limited way. For instance, if $f : \Sigma_1 \Rightarrow \Sigma_2$ is a TC abstraction with a recursive mapping function and Σ_1 is undecidable then, since the mapping function is computable, Σ_2 cannot be decidable. (If we had allowed nonrecursive abstraction mappings, an example of a nonrecursive TC abstraction with undecidable ground space and decidable abstract space would be one where any theorem of Σ_1 is mapped onto $\top$ and anything else onto $\bot$.)

We don't use TD abstractions which are not TC as completeness is lost—there is at least one theorem of the ground space whose abstraction is not a theorem of the abstract space. In applications where the abstract space is used to help find a proof in the ground space, we consider completeness to be a property you do not want to lose. We do not wish to take a great stand on the issue of completeness versus efficiency. We simply mean that there is no a priori reason for losing completeness, and even fewer reasons for losing it in an uncontrolled fashion. Of course TD abstractions can be used in other ways; for instance they can be used to implement derived inference rules but in these cases, to retain completeness, the overall strategy of the theorem prover should be different and not inside the `abstraction` tradition. Thus we claim that certain subclasses of

TI abstractions are the appropriate formalization for `abstraction`

and that these subclasses are captured and formalized inside this framework.

There are a few `abstractions` which do not preserve provability in any direction. Two interesting examples are the abstraction used in "gazing" [64], a heuristic which generalizes the peeking heuristic used in the UT theorem prover [4], and the abstraction used in its further refinement,

called "grazing" [70]. Gazing has proved to be reasonably successful in driving the unfolding of definitions in set theory. The idea underlying both gazing and grazing is that definitions should be unfolded in the ground space exactly in the same order as they are unfolded in the abstract space(s).

One criticism that can be made is that having to preserve provability may prevent us from capturing certain interesting applications. This may be true, but not of `abstractions`. For instance in [23] we have hinted how gazing could be modified to become complete, that is TI, without losing the original intuition. Outside the `abstractions` tradition, an interesting example of mapping in general not preserving provability in any direction is "analogy". When reasoning by analogy, people require very loose connections between the two theories, definitely nothing as strong as T* abstractions. It is the authors' opinion that this kind of reasoning can still be captured and formalized in our framework although no in-depth research has yet been carried out. A natural development of the current theory is to generalize all the concepts introduced in this section to range over parts of the formal systems using syntactic conditions (on the language, on the axioms, on the proof tree, etc.). For example, we might define an abstraction as being TI with respect to some subset of the language.

5. Using abstractions

Most previous work on `abstraction` has concentrated on just one way of using the abstract space to help find a proof in the ground space; that is, we build an abstract proof and then map it back onto a proof in the ground space. However, there are many more ways we could use abstractions. In this section, we shall not consider TC abstractions since, as we argued above, they are too strong and do not give simpler theories.

The different uses of abstractions can be divided along two main dimensions:

- In the first dimension we distinguish between *deductive uses* and *abductive uses*. With deductive uses the fact that a property of $f(\alpha)$ holds in the abstract space (e.g. $f(\alpha) \in \mathrm{TH}(\Sigma_2)$, $f(\alpha) \notin \mathrm{TH}(\Sigma_2)$) is a *guarantee* that the corresponding property of α holds in the ground space (e.g. $\alpha \in \mathrm{TH}(\Sigma_1)$, $\alpha \notin \mathrm{TH}(\Sigma_1)$). With abductive uses the fact that a property of $f(\alpha)$ holds in the abstract space is only a *suggestion*, not a guarantee, that the dual property of α holds in the ground space.
- In the second dimension, we have *positive uses* and *negative uses*. With positive uses we have a suggestion or a guarantee (depending on whether we are using abstraction deductively or abductively) that a wff α is a theorem of the ground space, that is $\alpha \in \mathrm{TH}(\Sigma_1)$. With

negative uses we have a suggestion or a guarantee that a wff is not a theorem and, under the hypotheses of syntactic completeness, that its negation is a theorem, that is $\neg\alpha \in \mathrm{TH}(\Sigma_1)$.

The deductive use of abstractions gives rise to "sound" theorem proving strategies since all the suggestions such strategies make are guaranteed to be true, while the abductive use of abstractions yields "complete" theorem proving strategies since such strategies will eventually allow us to find all theorems. It is only TC abstractions that will give us both sound and complete theorem proving strategies.

The following table summarizes the different uses of the classes of abstractions introduced so far.

	deductive	abductive
positive	TD	TI
negative	TI	TD

Let us start with the deductive use of TI abstractions to give negative information. For TI abstractions, if $f(\alpha) \notin \mathrm{TH}(\Sigma_2)$ then it follows that $\alpha \notin \mathrm{TH}(\Sigma_1)$. That is, if we cannot prove the abstraction of a wff then it definitely isn't a theorem in the ground space. This kind of information can be used to prune unprovable (sub)goals from the search space. Gelernter in his geometry theorem prover [16], and Reiter in an incomplete ND theorem prover [68] used (semantic) abstractions in this way. When Σ_1 is syntactically complete (where $\alpha \notin \mathrm{TH}(\Sigma_1)$ implies $\neg\alpha \in \mathrm{TH}(\Sigma_1)$), we can also deduce positive information as $f(\alpha) \notin \mathrm{TH}(\Sigma_2)$ implies $\neg\alpha \in \mathrm{TH}(\Sigma_1)$. Additionally, if Σ_1 contains the rule of double negation elimination we can even deduce un-negated wffs since if $f(\neg\alpha) \notin \mathrm{TH}(\Sigma_2)$ then $\alpha \in \mathrm{TH}(\Sigma_1)$. Thus, to prove a wff α, one interesting strategy is to attempt to prove that $\neg f(\neg\alpha)$is a theorem in Σ_2; theorem proving in Σ_2 should be easier than in Σ_1 so we can save effort. Unfortunately this strategy, though it will tell us if α is a theorem of Σ_1, will not directly give us a proof. Another disadvantage is that such a strategy is not complete; it will not allow us to prove all theorems of the ground space.

For TD abstractions, if $f(\alpha) \in \mathrm{TH}(\Sigma_2)$ then it follows that $\alpha \in \mathrm{TH}(\Sigma_1)$. That is, if we can prove the abstraction of a wff is a theorem in the abstract space then it is definitely a theorem in the ground space. This is the deductive use of TD abstractions to provide positive information. Often, the proof that $f : \Sigma_1 \Rightarrow \Sigma_2$ is a TD abstraction shows, given a proof in Σ_2 of an arbitrary wff $f(\alpha)$, how you can actually construct a proof of α in Σ_1. This strategy was used with success by Wos et al. [81] in an automatic theorem prover for group theory. Their TD abstraction replaced functions

by predicates; however, theorems that required the existence or uniqueness properties of functions could not be proved by their theorem prover; this is an example of the incompleteness of a theorem proving strategy based upon a deductive use of abstraction.

Let us now consider the abductive use of TI abstractions to suggest positive information. With TI abstractions which are not TC there will be some wffs α for which $f(\alpha) \in \mathrm{TH}(\Sigma_2)$ but for which $\alpha \notin \mathrm{TH}(\Sigma_1)$. However, provided the abstraction does not throw away too much information the implication that $\alpha \in \mathrm{TH}(\Sigma_1)$ implies $f(\alpha) \in \mathrm{TH}(\Sigma_2)$ will reverse in many cases. Thus, if we can prove that $f(\alpha) \in \mathrm{TH}(\Sigma_2)$ then it is likely that $\alpha \in \mathrm{TH}(\Sigma_1)$. TI abstractions can not only be used to suggest positive facts about Σ_1; they can also suggest proof steps. There is often a great similarity in shape between a proof of a wff in Σ_1 and a proof of the abstraction of the wff in Σ_2. This is especially true of abstractions which have the same deductive machinery (or a subset of it) in the ground and abstract spaces. The steps of a proof in Σ_2 can thus be used to guide theorem proving in Σ_1. The proof steps provide "islands" to get to; we move between these islands by filling in the details or applying the inference rules thrown away by the abstraction. This strategy saves us effort as it "divides and conquers" the problem solving. It is perhaps the most common use of abstraction and it is how `abstractions` have been used in the past.

Lastly, we conclude with the abductive use of TD abstractions to suggest negative information. With TD abstractions, if $f(\alpha) \notin \mathrm{TH}(\Sigma_2)$ then it is likely that $\alpha \notin \mathrm{TH}(\Sigma_1)$. This information could be used, for example, to suggest (sub)goals to prune from the search space. More weakly, it might just suggest which subgoals to delay attempting to prove. For abstractions in which Σ_1 is syntactically complete, TD abstractions can also provide positive information as $f(\neg\alpha) \notin \mathrm{TH}(\Sigma_2)$ suggests that $\alpha \in \mathrm{TH}(\Sigma_1)$. It is not clear how useful this way of using TD abstractions can be. We have found no references to applications which use TD abstractions in this way.

6. Refutation systems

In Definition 4.3, we classified abstractions by the relationship between provability in the ground space and provability in the abstract space. This is appropriate when the deductive machinery of both spaces is used to generate theorems. However, there are formal systems whose deductive machinery determines inconsistency, for instance resolution-based systems. We call the systems of the first type, provability systems, those of the second type, refutation systems. (We say that a formal system, Σ, is absolutely inconsistent iff, for any wff α, $\alpha \in \mathrm{TH}(\Sigma)$, and inconsistent iff there exists a wff α such that $\alpha \in \mathrm{TH}(\Sigma)$ and $\neg\alpha \in \mathrm{TH}(\Sigma)$. For an absolutely

inconsistent system, we do not require that negation be part of the language. However, when negation is part of the language and we are in a classical first-order logic, an absolutely inconsistent system is also inconsistent and vice versa.) In the cases when the deductive machinery determines inconsistency, abstractions can be better classified on how an (absolutely) inconsistent formal system is mapped onto an (absolutely) inconsistent formal system.

As pointed out at the very beginning, a formal system may be thought of as consisting of a language, Λ, and a subset of the language, Θ, which represents the statements of the theory. Thus, another very important example of Θ is when $\Theta = \mathrm{NTH}(\Sigma)$, where $\mathrm{NTH}(\Sigma)$ is the set of the wffs α which, if added as an assumption to Σ, make the resulting system absolutely inconsistent. Notice that assumptions are not axioms, the difference being in the case of formulas which are open (if they are closed formulas, assumptions behave like axioms). For instance, assumptions, differently from axioms, in ND may prevent the application of "forall" introduction and "exists" elimination [66] while in Hilbert calculi they are such that the deduction theorem does not hold [51]. In resolution the distinction between axioms and assumptions is irrelevant. As $\mathrm{NTH}(\Sigma)$ is used only when Σ is a refutation system, we will simply say that a wff is added to the axioms leaving implicit the distinction between axioms and assumptions. Given a formal system Σ, we call the elements of $\mathrm{NTH}(\Sigma)$ the nontheorems of Σ. $\mathrm{TH}(\Sigma)$ and $\mathrm{NTH}(\Sigma)$ are obviously related. For instance in classical first-order logics, $\alpha \in \mathrm{NTH}(\Sigma)$ iff $\neg\alpha \in \mathrm{TH}(\Sigma)$; Σ is inconsistent iff $\mathrm{TH}(\Sigma) = \mathrm{NTH}(\Sigma) = \Lambda_\Sigma$, or iff $\mathrm{TH}(\Sigma) \cap \mathrm{NTH}(\Sigma) \neq \emptyset$; outside $\mathrm{TH}(\Sigma)$ and $\mathrm{NTH}(\Sigma)$ but inside the language are all those formulas α such that neither α nor $\neg\alpha$ belongs to $\mathrm{TH}(\Sigma)$, if a theory is syntactically complete no formulas are outside the union of $\mathrm{TH}(\Sigma)$ and $\mathrm{NTH}(\Sigma)$.

We will now classify abstractions on the relationship between $\mathrm{NTH}(\Sigma_1)$ in the ground space and $\mathrm{NTH}(\Sigma_2)$ in the abstract space. Entirely dual to Definition 4.3, we give a definition of the following inconsistency preserving abstractions:

Definition 6.1 (*NT* abstractions*). An abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is said to be an

(1) *NTC abstraction* iff, for any wff α, $\alpha \in \mathrm{NTH}(\Sigma_1)$ iff $f_\Lambda(\alpha) \in \mathrm{NTH}(\Sigma_2)$;
(2) *NTD abstraction* iff, for any wff α, if $f_\Lambda(\alpha) \in \mathrm{NTH}(\Sigma_2)$ then $\alpha \in \mathrm{NTH}(\Sigma_1)$;
(3) *NTI abstraction* iff, for any wff α, if $\alpha \in \mathrm{NTH}(\Sigma_1)$ then $f_\Lambda(\alpha) \in \mathrm{NTH}(\Sigma_2)$.

(Here "N" stands for "non".) Dually to T* abstractions, any NTC abstrac-

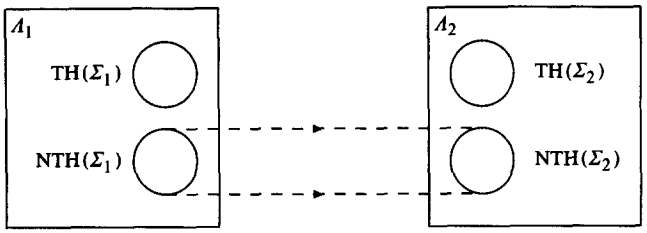

Fig. 4. NTC abstractions.

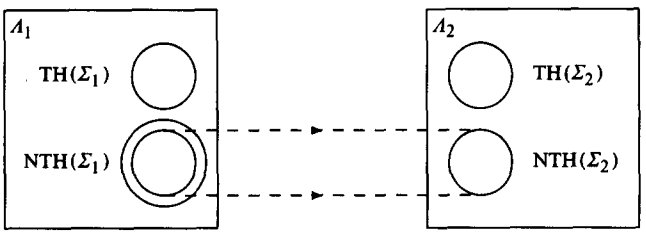

Fig. 5. NTD abstractions.

tion is both an NTD abstraction and an NTI abstraction.

Some more notational and naming conventions. An abstraction is said to be an NT* abstraction iff it is an NTC abstraction or an NTD abstraction or an NTI abstraction. It is said to be a TC* abstraction iff it is a TC abstraction or an NTC abstraction, TD* abstraction iff it is a TD abstraction or an NTD abstraction, TI* abstraction iff it is a TI abstraction or an NTI abstraction. When an abstraction is known to fall in more than one of the above classes we write all of them in the prefix. Thus for instance a TC/NTC abstraction is an abstraction which is both a TC and an NTC . We also write TH*(Σ) to mean NTH(Σ) or TH(Σ). Statements made involving names containing "*" (i.e. T* abstraction, TC* abstraction, etc.) can be read by substituting in all the valid ways the same letter (i.e. "C", "D", or "I" in first case; "N" or nothing in the second) uniformly inside the sentence.

We have introduced NT* abstractions with the goal of using them when the deductive machinery works by refutation. In such cases, we claim that they play the same role that T* abstractions play in provability systems. This claim deserves greater attention. After all, even when we use a refutation system we are still interested in provability; we should therefore expect to be still interested in T* abstractions.

This apparent contradiction can be explained as follows. Everything depends on how the user interacts with the theorem prover. The basic idea underlying refutation systems is that a wff is a theorem if adding its negation to the axioms gives an inconsistent formal system. In some (most?) cases, the goal is input and the system itself automatically negates it before adding it to the set of axioms. If this also happens in the abstract space, negation

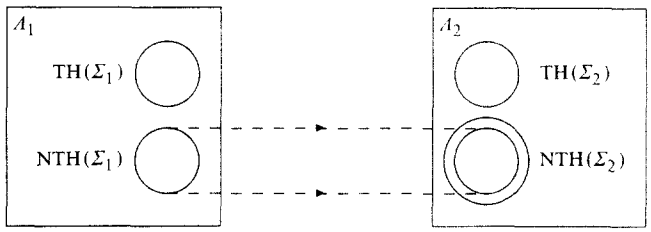

Fig. 6. NTI abstractions.

is not abstracted, the "inference engine" (resolution plus the negation of the goal) works on provability, and T* abstractions are the appropriate abstractions to use.

Before we can formalize this argument further, we need to define a very useful notion, that of a formal system with negation. We will use this notion to show under what assumptions a T* abstraction can be used in a refutation system.

Definition 6.2 (*System with negation*). Σ is a formal system with negation iff there is the unary connective "$\neg$", called negation, such that, for any expression α,

(1) α is a wff iff $\neg\alpha$ is a wff;
(2) $\alpha \in \mathrm{TH}(\Sigma)$ iff $\neg\alpha \in \mathrm{NTH}(\Sigma)$;
(3) $\neg\alpha \in \mathrm{TH}(\Sigma)$ iff $\alpha \in \mathrm{NTH}(\Sigma)$.

Three observations are worthwhile. First, condition (2) means (in Prawitz' ND terms) that the system must contain the "reasoning by contradiction" inference rule and

$$\frac{\alpha \quad \neg\alpha}{\bot}.$$

A system with negation is thus inconsistent iff it is absolutely inconsistent. As almost all the systems of interest are with negation and what is relevant in this context is "absolute consistency/inconsistency", in the following we will write "inconsistency/consistency" to mean "absolute inconsistency/absolute consistency". Second, any system which has an introduction rule corresponding to every elimination rule, as is usually the case, satisfies conditions (1) and (2) and has classical negation. In this case condition (2) yields condition (3) which holds intuitionistically. For many proofs in the paper intuitionistic negation is sufficient. Third, all the most common first-order systems, that is Hilbert systems, all the ND calculi, and resolution, are systems with negation. Notice that if we had defined $\mathrm{NTH}(\Sigma)$ as the set of those wffs which, if added as axioms (instead of as assumptions), make Σ inconsistent, we would have failed to capture at least Hilbert systems and

ND (the trivial counterexample being a system Σ where $p(a)$ is an axiom; we would have in fact that $\neg p(x) \in \mathrm{NTH}(\Sigma)$ with $p(x) \notin \mathrm{TH}(\Sigma)$).

The notion of system of negation allows us to define a set of cases where the different forms of abstractions coincide. The following theorem holds.

Theorem 6.3. *If f is an abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ and Σ_1 and Σ_2 are two syntactically complete systems with negation, then f is a NTI abstraction (TI abstraction) iff it is a TD abstraction (NTD abstraction).*

Proof. Let's consider NTI abstractions and TD abstractions and only one direction. The other proofs are entirely dual.

($\Rightarrow$) In an NTI abstraction (see Lemma 6.5 below), if $\neg f(p) \notin \mathrm{TH}(\Sigma_2)$ then $\neg p \notin \mathrm{TH}(\Sigma_1)$. But, from the completeness of Σ_1, we have $\neg p \notin \mathrm{TH}(\Sigma_1)$ iff $p \in \mathrm{TH}(\Sigma_1)$. Similarly, from the completeness of Σ_2, we have $\neg f(p) \notin \mathrm{TH}(\Sigma_2)$ iff $f(p) \in \mathrm{TH}(\Sigma_2)$. Thus, $f(p) \in \mathrm{TH}(\Sigma_2)$ implies $p \in \mathrm{TH}(\Sigma_1)$. But this is exactly the definition of TD abstraction. □

The intuition is that, as the abstract space is complete, for instance with TD abstractions, decreasing the theorems corresponds to increasing the nontheorems. An obvious corollary is that, if $f : \Sigma_1 \Rightarrow \Sigma_2$ is an abstraction and Σ_1 and Σ_2 are syntactically complete then f is a TC abstraction iff it is a NTC abstraction. Theorem 6.3 is not very relevant as almost all the theories we consider are incomplete, that is they are such that $\alpha \notin \mathrm{TH}(\Sigma)$ does not imply $\neg\alpha \in \mathrm{TH}(\Sigma)$. Generalizing to incomplete theories causes Theorem 6.3 to fail. Various situations may happen in this case. For example, an NTI abstraction with the abstract space, Σ_2, complete and the ground space, Σ_1, incomplete will map both the members of $\mathrm{TH}(\Sigma_1)$ and the formulas which neither belong to $\mathrm{TH}(\Sigma_1)$ nor to $\mathrm{NTH}(\Sigma_1)$ onto $\mathrm{TH}(\Sigma_2)$ while a TD abstraction will map into $\mathrm{NTH}(\Sigma_2)$ both the members of $\mathrm{NTH}(\Sigma_1)$ and the formulas which neither belong to $\mathrm{TH}(\Sigma_1)$ nor to $\mathrm{NTH}(\Sigma_1)$.

The question then remains under which conditions T* abstractions can be used in refutation systems. Let us consider the following lemma.

Lemma 6.4. *If Σ_1 and Σ_2 are two formal systems with negation and if $f : \Sigma_1 \Rightarrow \Sigma_2$ is a*

- *TC abstraction then, for any α, $\neg\alpha \in \mathrm{NTH}(\Sigma_1)$ iff $\neg f(\alpha) \in \mathrm{NTH}(\Sigma_2)$;*
- *TD abstraction then, for any α, if $\neg f(\alpha) \in \mathrm{NTH}(\Sigma_2)$ then $\neg\alpha \in \mathrm{NTH}(\Sigma_1)$;*
- *TI abstraction then, for any α, if $\neg\alpha \in \mathrm{NTH}(\Sigma_1)$, then $\neg f(\alpha) \in \mathrm{NTH}(\Sigma_2)$.*

Thus, provided both the ground and abstract spaces are systems with negation, we can use T* abstractions with refutation systems. For instance,

with a TI abstraction if α is a theorem in the ground space then the negation of its abstraction, $\neg f(\alpha)$, will make a refutation theorem prover stop with success (that is, by generating $\perp$).

> T* abstractions can be used in refutation theorem proving as long as the wff added to the axioms of the abstract space is the negation of the abstraction of the wff whose negation is added to the axioms of the ground space.

In formulas, instead of adding $f(\neg\alpha)$ to the axioms of the abstract space, we must add $\neg f(\alpha)$. However, when the system does not negate the goal, the "inference engine" uses refutation, and $f(\neg\alpha)$ is added to the axioms of the abstract space, only NT* abstractions can be used.

So far we have discussed the use of T* abstractions in refutation-based theorem proving. The dual question also arises, namely can NT* abstractions be used with a provability system? Lemma 6.4 holds dually, that is:

Lemma 6.5. *If Σ_1 and Σ_2 are two formal systems with negation, then if $f : \Sigma_1 \Rightarrow \Sigma_2$ is a*

- *NTC abstraction then, for any α, $\neg\alpha \in \mathrm{TH}(\Sigma_1)$ iff $\neg f(\alpha) \in \mathrm{TH}(\Sigma_2)$;*
- *NTD abstraction then, for any α, if $\neg f(\alpha) \in \mathrm{TH}(\Sigma_2)$ then $\neg\alpha \in \mathrm{TH}(\Sigma_1)$;*
- *NTI abstraction then, for any α, if $\neg\alpha \in \mathrm{TH}(\Sigma_1)$, then $\neg f(\alpha) \in \mathrm{TH}(\Sigma_2)$.*

The interpretation of this lemma is entirely dual to that of Lemma 6.4. NT* abstractions can be used in provability systems to prove $\neg\alpha$ provided we try to prove $\neg f(\alpha)$ in the abstract space.

Finally, are there abstractions which can be used both with refutation systems and with provability systems, irrespective of how the goal is negated? To answer this question, we introduce the notion of negation preserving abstractions. This notion is crucial for abstractions to be used in both refutation and provability systems.

Definition 6.6 (*Negation preserving abstractions*). Let Σ_1 and Σ_2 be two systems such that α is a wff iff $\neg\alpha$ is. An abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is negation preserving iff, for any α, $f(\neg\alpha) = \neg f(\alpha)$.

Preserving negation is a powerful concept. If an abstraction preserves negation then the notions of preserving provability and of preserving inconsistency collapse. In fact the following theorem holds.

Theorem 6.7. *If Σ_1 and Σ_2 are two formal systems with negation, then a negation preserving abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is a T* abstraction iff it is a NT* abstraction.*

Proof. Let us consider TI abstractions and only one direction. The other proofs are analogous. Suppose that f is a TI abstraction. Since Σ_1 is a system with negation, if $\alpha \in \mathrm{NTH}(\Sigma_1)$ then $\neg\alpha \in \mathrm{TH}(\Sigma_1)$. But f is TI . Thus $f(\neg\alpha) \in \mathrm{TH}(\Sigma_2)$. As f is negation preserving, this implies $\neg f(\alpha) \in \mathrm{TH}(\Sigma_2)$. From which it follows that $f(\alpha) \in \mathrm{NTH}(\Sigma_2)$ and that f is NTI. □

A consequence of this theorem is that *a negation preserving abstraction mapping can thus be used in both refutation and provability systems.* Notice moreover that we have made no hypotheses about the deductive machinery used in the ground and abstract spaces. The choice of deductive machinery is, in this perspective, irrelevant. All that matters is the deductive closure it generates. This intuition can be exploited to show that if a negation preserving abstraction f is T*/NT* then all the abstractions which differ from f only in the choice of the deductive machineries are T*/NT* as long as their deductive machineries compute the same set of theorems. This is what the following theorem says.

Theorem 6.8. *Let $\Sigma_1 = \langle \Lambda_1, \Omega_1, \Delta_1 \rangle$ and $\Sigma_2 = \langle \Lambda_2, \Omega_2, \Delta_2 \rangle$ be two systems with negation, and $f : \Sigma_1 \Rightarrow \Sigma_2$, any negation preserving T* abstraction. Let $\Sigma_1' = \langle \Lambda_1, \Omega_1, \Delta_1' \rangle$ and $\Sigma_2' = \langle \Lambda_2, \Omega_2, \Delta_2' \rangle$ be any two systems such that $\mathrm{TH}(\Sigma_i') = \mathrm{TH}(\Sigma_i)$, $i = 1, 2$. Then the abstraction $f' : \Sigma_1' \Rightarrow \Sigma_2'$, which uses the same mapping function as $f : \Sigma_1 \Rightarrow \Sigma_2$ is a T*/NT* abstraction.*

One significant consequence of Theorem 6.8 is that *any abstraction on the language and the axioms defined for a refutation-based system can be safely applied with a provability system (and vice versa) as long as they generate the same set of theorems.* Thus for instance, an abstraction developed for refutation systems (e.g. all those suggested by Plaisted) can be safely used with any provability system complete for first-order logic, for instance ND (modulo extending it to be defined over the richer syntax of ND, e.g. "↔", but this is a minor point).

As far as we know, most of the abstractions proposed in the past are negation preserving. However, there are abstractions which are not (see Table 1 at the end of Section 8). An interesting class of abstractions which are not negation preserving and work between systems with negation is the class of ground abstractions (see Example 8.27). In some cases, negation may not be part of the language of the ground or abstract spaces, or negation may only be partially preserved by the mapping. One such abstraction, used

in GPS [57] and described in Example 8.3, is such that the abstract language allows only sets of atomic formulas. The mapping deletes negation as well as any other logical symbol. Not considering the logical symbols seems quite a drastic step as a lot of the information is lost; on the other hand it allows very fast formula manipulation. Intuitively, it seems that it might work well with short proofs.

7. Some further classifications

The definitions of T* and NT* abstractions are not constraining enough; they only partially capture "certain desirable" properties preserved by a mapping (property (2)). Abstractions (and `abstractions` as a particular case) usually satisfy much stronger requirements than simply preserving provability or inconsistency.

Most abstractions f (and all `abstractions`) are such that for any element el in the abstract space (e.g. a wff in its language, an axiom in its axiom set, an inference rule in its deductive machinery), it is possible to find at least one corresponding element el', in the ground space such that, loosely writing, $f(el') = el$. This is very important, for instance in `abstractions`, in order to use effectively the similarity between abstract and ground proof. (We can exploit the similarity by substituting each wff/axiom/inference rule in the abstract proof with one among the ground wffs/axioms/inference rules mapped onto it.) The definition of abstraction given above can be refined to capture this idea of correspondence simply by exploiting Definition 4.2 of a formal system. It is sufficient to describe how the the axioms and the inference rules of the ground and abstract spaces are related via the use of two extra mapping functions f_Ω and f_Δ. Analogously to f_Λ, the functions f_Ω and f_Δ are requested to be total and computable. This definition of abstraction with three mapping functions corresponds closely to the way abstractions are often used. Given a ground space, we often construct the abstract space using three functions: a function that maps the wffs in the ground language onto the wffs in the abstract language, a second function that maps the axioms of the ground space onto the axioms of the abstract space, and a third function that maps the deductive machinery of the ground space onto that of the abstract space. This three-part description of an abstraction in terms of the mappings on the language, on the axioms, and on the deductive machinery was introduced in [21] and it is especially appropriate when the axioms of the ground space are not fixed. We often define the function for mapping the axioms sufficiently generally so that it will work with whatever axioms we happen to have in the ground space; the abstract space is not fixed but depends on the particular axioms of the ground space. In this paper we need not introduce the extra notation.

However we will sometimes allude to this description of abstraction when we talk about the *mapping of the axioms*, the *mapping of the inference rules*, and the *mapping of the ground space onto the abstract space.*

A second important requirement, satisfied by most abstractions f (and by all `abstractions`) is that the abstract space Σ_2 is completely generated from the ground space Σ_1; in other words that the language, the axioms, and the deductive machinery are constructed only on the basis of the corresponding elements in the ground space. In this case we write that $\Sigma_2 = f(\Sigma_1)$, where f is the abstraction, and say that f is *surjective.* This condition can be formally captured by requiring that the mapping functions be surjective. As in the notion of abstraction used here we do not consider the extra two mappings, we need only request that f_A be surjective.

In this paper we restrict ourselves to surjective abstractions.

Abstractions can be further classified depending on how they map axioms and inference rules in the ground onto those of the abstract space. This will be the topic of the remainder of this section. These definitions capture concepts that are very relevant to the theory of mapping an abstract proof back onto a ground proof [24]. In this paper they will be used extensively in Section 8 to classify previous examples of abstraction.

The first definition captures the class of abstractions which map the axioms the same way as the language. (If A is a set and f a function defined over A, by $f(A)$ we mean the set $\{f(a)\colon a \in A\}$.)

Definition 7.1 (*A/Ω-invariant abstraction*). Let $\Sigma_1 = \langle A_1, \Omega_1, \Delta_1\rangle$ and $\Sigma_2 = \langle A_2, \Omega_2, \Delta_2\rangle$ be two formal systems. An abstraction, $f : \Sigma_1 \Rightarrow \Sigma_2$ is said to be A/Ω-invariant iff $\Omega_2 = f_A(\Omega_1)$, and A/Ω-variant otherwise.

A/Ω-abstractions are used whenever we do not distinguish between wffs and axioms. This is often true of abstractions which are not tuned to a particular theory or when no special constraints are imposed on the abstract space. Example 8.16 uses a A/Ω-variant abstraction.

Our second definition is of an abstraction which keeps the same inference rules in the abstract space as in the ground space.

Definition 7.2 (*Δ-invariant abstraction*). Let $\Sigma_1 = \langle A_1, \Omega_1, \Delta_1\rangle$ and $\Sigma_2 = \langle A_2, \Omega_2, \Delta_2\rangle$ be two formal systems. An abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is called Δ-invariant iff $\Delta_2 \subseteq \Delta_1$, Δ-variant otherwise.

In a Δ-invariant abstraction, a subset of the inference rules of Σ_1 is used in Σ_2. Most `abstractions` are Δ-invariant since the same inference engine can be used for both the ground and the abstract spaces. This gives a great economy of implementation. Moreover the problem of mapping back

is simplified since abstract and ground proof trees can be matched more easily—the abstract proof looks essentially like a ground proof with gaps. Notice that $\Delta_2 \subseteq \Delta_1$ may hold even if the alphabet of A_1 is different from that of A_2. The variables used in defining the inference rules are meta-variables and only commit us to a particular logical syntax for the object language. The abstraction used in Example 8.3 is Δ-variant.

Our third definition is of an abstraction which maps the inference rules the same as the language.

Definition 7.3 (*A/Δ-invariant abstraction*). Let $\Sigma_1 = \langle A_1, \Omega_1, \Delta_1 \rangle$ and $\Sigma_2 = \langle A_2, \Omega_2, \Delta_2 \rangle$ be two formal systems. An abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is called A/Δ-invariant iff

$$\Delta_2 = \left\{ \frac{f_A(\alpha_1), \ldots, f_A(\alpha_n)}{f_A(\alpha_{n+1})} : \frac{\alpha_1, \ldots, \alpha_n}{\alpha_{n+1}} \in \Delta_1 \right\}.$$

It is called A/Δ-variant otherwise.

Note that $f_A(\alpha_j)$ must be read as applying f_A to the wff substituted for α_j when applying the inference rule. A/Δ-invariant abstractions are also often used as `abstractions` since they make the problem of mapping back easier. For example, it is easier to establish a connection between the premises of the application of an abstract inference rule and the premises of the application of a ground inference rule. Notice also that the notions of Δ-invariance and A/Δ-invariance are independent; A/Δ-invariance does not imply Δ-invariance (or vice versa). A/Δ-invariant abstractions change the input and outputs to the inference rules according to f_A while Δ-invariant abstractions keep the same inference engine. For example, the abstraction in GPS (see Example 8.3) is A/Δ-invariant, but it is not Δ-invariant; the inference rules used in the abstract theory are completely different to those in the ground theory.

Our fourth definition is of an abstraction which maps both the axioms and the inference rules the same as the language.

Definition 7.4 (*Σ-invariant abstraction*). An abstraction f is Σ-invariant iff it is A/Ω-invariant and A/Δ-invariant, Σ-variant otherwise.

In [70,71], veridicality, which is a notion very similar to (but slightly weaker than) Σ-invariance, has been claimed to be fundamental for an `abstraction` to make any sense at all.

The four definitions so far have characterized various relationships between the mapping of the language and the mapping of the axioms and inference rules. The final definition in this section, that of a theory abstraction, captures a large class of abstractions which map the theory and

not the logic; that is, abstractions that leave the logical syntax of formulas unchanged.

Definition 7.5 (*Theory abstraction*). If $f : \Sigma_1 \Rightarrow \Sigma_2$ is an abstraction, then f is called a theory abstraction iff for any wffs α and β,

- $f_A(*\beta) = *f_A(\beta)$ for all the logical unary connectives, $*$;
- $f_A(\alpha \circ \beta) = f_A(\alpha) \circ f_A(\beta)$ for all the logical binary connectives, $\circ$;
- $f_A(\diamond x.\alpha) = \diamond x.f_A(\alpha)$ for all the quantifiers, $\diamond$.

Note that a theory abstraction can be used to drop variables (if $f_A(p)$ does not mention x, then $\diamond x.f_A(p)$ is equivalent to $f_A(p)$) and to drop conjuncts and disjuncts (for example, if $f_A(q)$ maps onto $\bot$, then $f_A(p \vee q)$ is equivalent to $f_A(p)$). The idea underlying theory abstractions is that most useful abstractions abstract the theory but preserve the logic. That is, they map the atomic wffs and not the logical structure of the wff. In general, the logic is well-behaved and it is the theory that needs to be simplified. Indeed, the authors would argue that you should only change the logical structure of a wff with great care as the consistency of a logic is very finely balanced. As they preserve the logical structure of formulas, theory abstractions are often A-invariant with the same inference rules used in both the ground and the abstract spaces.

8. Examples of abstractions

This section describes some case studies, some of which are very famous, taken from various subfields of artificial intelligence. In most cases the original description was quite informal. In order to fit these examples into our framework we have performed a (hopefully faithful) reconstruction of what was said in the original papers. For each example we first briefly describe the work (this description is not meant to be self-contained) and then we formalize the abstraction and analyze its properties.

This section has many goals. First, we hope it will convince the reader that our framework is very powerful and can capture not only most previous work in abstraction but also work which initially seems completely unrelated. Second, it provides a unified view of work from many different areas which was carried out with a great variety of goals. Third, it supports the authors' belief that most work in automated reasoning should be formally characterizable in terms of the provability relation. Fourth, it demonstrates how this framework generalizes some important results. Fifth, if (as we believe) this sample of abstractions is representative of `abstractions` as a whole, it highlights the simple properties of most `abstractions` and thus suggests what properties the `abstractions` we construct should possess. We

view this section as one of the most important contributions of this paper. Our understanding and definition of abstraction has been strongly influenced by these examples.

The structure of this section is as follows. We begin by describing three historically important examples of abstraction. We divide the remaining examples into four classes: propositional abstractions (in which the abstract space is propositional), predicate abstractions (in which predicate symbols are mapped together), domain abstractions (in which constants are mapped together) and metatheoretic mappings (which capture some major metatheoretic results like Herbrand's theorem). We conclude with a summary of the properties of each of the examples, and a brief discussion of the general properties of abstractions.

8.1. Historical examples

Example 8.1 (*Planning*). ABSTRIPS [69] was one of the very first and most famous uses of abstraction. ABSTRIPS built STRIPS plans in which operators applied to states of the world, generating new states. The preconditions to operators are atomic formulas abstracted according to their *criticality*, computed semi-automatically by the system. Each precondition p_i has its own criticality which is a natural number associated to the predicate symbol occurring in it. Criticalities are therefore totally ordered. The idea is to build a hierarchy of abstract spaces, each level in the hierarchy containing all the preconditions above a given criticality. We write $i \in crit(\kappa)$ to mean that the criticality of the precondition p_i is greater than κ. (Therefore, if $\kappa_1 < \kappa_2$ then $crit(\kappa_2) \subset crit(\kappa_1)$.) To put this into a theorem proving context and following Green [29], we adopt a situation calculus. ABSTRIPS can then be formalized as a Λ/Ω-invariant abstraction, $f_{AB} : \Sigma_1 \Rightarrow \Sigma_2$, which maps between situation calculi, Σ_1 and Σ_2, with first-order languages, frame, operator, and theoretic axioms, and natural deduction rules of inference. Operators are wffs of the form " $\forall s.(\bigwedge_{1 \leqslant i \leqslant n} p_i(s) \rightarrow q(f(s)))$ ", where p_i is a precondition, s is a state of the world, f is some action, and q describes the new state of the world. Goals are wffs of the form "$\exists s\ r(s)$". Axioms are mapped from Σ_1 to Σ_2 using the same mapping function as the language.

The mapping of wffs (and axioms) is as follows:

- $f_{AB}(\alpha) = \alpha$ if α is an atomic formula;
- $f_{AB}(\neg\alpha) = \neg f_{AB}(\alpha)$;
- $f_{AB}(\alpha \circ \beta) = f_{AB}(\alpha) \circ f_{AB}(\beta)$, where "$\circ$" is "$\wedge$" or "$\vee$";
- $f_{AB}(\sharp x.\alpha) = \sharp x.f_{AB}(\alpha)$, where "$\sharp$" is "$\exists$" or "$\forall$";
- $f_{AB}(\alpha \rightarrow \beta) = f_{AB}(\alpha) \rightarrow f_{AB}(\beta)$, provided "$\alpha \rightarrow \beta$" is not an operator;
- $f_{AB}(\bigwedge_{1 \leqslant i \leqslant n} p_i(s) \rightarrow r) = \bigwedge_{i \in crit(\kappa)} p_i(s) \rightarrow f_{AB}(r)$, for any operator (where $i \in crit(\kappa)$ if the criticality of p_i is greater than κ).

For example, the operator for climbing a (climbable) object,

$$at(z,x,s) \wedge climbable(y,z,s) \rightarrow at(z,x,climb(y,z,s)),$$

might abstract to one in which we do not bother to check that the object is climbable,

$$at(z,x,s) \rightarrow at(z,x,climb(y,z,s))$$

This abstraction is TI.

Theorem 8.2. *If* $\alpha \in \mathrm{TH}(\Sigma_1)$, *then* $f_{\mathrm{AB}}(\alpha) \in \mathrm{TH}(\Sigma_2)$.

Proof (*Outline*). By proving that given a deduction tree Π_1 of α, we can build a deduction tree Π_2 of $f_{\mathrm{AB}}(\alpha)$ discharging the abstraction of the same assumptions. The proof proceeds by induction on the depth N of Π_1. For proofs of length 1, f_{AB} is applied to the single wff in the tree; this generates a valid deduction in Π_2. Assume that we have shown it for trees up to size N. We shall use $f_{\mathrm{AB}}(\Pi)$ to represent the tree in Σ_2 constructed from a tree Π in Σ_1 of size N or less. We show that it is true for deduction trees of size $N+1$ irrespective of the rule application used to construct the tree of size $N+1$ from tree(s) of size N (or less). Any rule application that is not modus ponens involving an operator translates unmodified. For instance, an or-introduction on φ in Π_1 becomes an or-introduction on $f_{\mathrm{AB}}(\varphi)$ in Π_2. For an operator application, the following transformation is performed:

$$\frac{\dfrac{\Pi}{\bigwedge\limits_{1\leqslant i\leqslant n} p_i} \qquad \bigwedge\limits_{1\leqslant i\leqslant n} p_i \rightarrow q}{q} \Longrightarrow \frac{f_{\mathrm{AB}}\left(\dfrac{\Pi}{\bigwedge\limits_{1\leqslant i\leqslant n} p_i}\right)}{\bigwedge\limits_{i\in crit(\kappa)} p_i \qquad \bigwedge\limits_{i\in crit(\kappa)} p_i \rightarrow f_{\mathrm{AB}}(q)}}{f_{\mathrm{AB}}(q)}$$

By the induction hypothesis and the fact that $\bigwedge_{i\in crit(\kappa)} p_i$ follows from $\bigwedge_{1\leqslant i\leqslant n} p_i$ by a (possibly empty) sequence of applications of and-elimination, this is a valid abstract deduction tree which discharges the (abstraction of the) same assumptions as the deduction tree in Σ_1. □

Note that the proof in the abstract space is longer than the proof in the ground space. The purpose of abstraction is not to find these longer proofs; we hope that there are also going to be shorter proofs. These shorter proofs are those that don't try to satisfy p_i for $i \notin crit(\kappa)$. However, there is no guarantee that there will be a shorter proof than the one exhibited; we will always be able to devise an obtuse theory in which to prove the

p_i for $i \in crit(\kappa)$ we have to prove all the other p_i for $i \notin crit(\kappa)$. This problem would be eliminated if ABSTRIPS had abstracted both left- and right-hand sides of operators. Under such circumstances, p_i for $i \notin crit(\kappa)$ would not even appear in the abstract language. Note also that since this abstraction is TI there is no guarantee that given an abstract proof we will be able to construct a ground proof which uses the same operators. ABSTRIPS works because the operators are (largely) independent. Thus we can solve the preconditions separately. Actually, ABSTRIPS is a little more clever than this; the use of criticalities allows some operator dependence to occur as it restricts the order in which the operator preconditions have to be satisfied. For an abstraction like ABSTRIPS to be useful, the area of the region $\mathrm{TH}(\Sigma_2) - f(\mathrm{TH}(\Sigma_1))$, that is those abstract theorems which don't map back, needs to be small in comparison to $f(\mathrm{TH}(\Sigma_1))$, those abstract theorems which do.

Example 8.3 (*Problem solving*). Abstraction has been a useful problem solving heuristic ever since the early days of AI research. Indeed, GPS used abstraction in its *planning method* [57] for state space search. After the objects and operators for a problem have been abstracted, the entire deductive machinery of GPS is used to solve the abstracted problem; this solution is then used to construct a plan to guide the solution of the original problem. We will just consider one of the Σ-invariant abstractions suggested for propositional logic problems in [57]. Let us call it $f_{\mathrm{GPS}} : \Sigma_1 \Rightarrow \Sigma_2$. Σ_1 is the propositional calculus of *Principia Mathematica*. Σ_2 is a formal system in which the wffs are (nested lists of) propositional sentence letters. To construct the abstract space, the same mapping is applied to the wffs, the axioms, and the premises and consequences of the inference rules of the ground space. If α and β are two formulas in the ground language then:

$$f_{\mathrm{GPS}}(\alpha \vee \beta) = f_{\mathrm{GPS}}(\alpha \wedge \beta) = f_{\mathrm{GPS}}(\alpha \rightarrow \beta) = (f_{\mathrm{GPS}}(\alpha), f_{\mathrm{GPS}}(\beta));$$

$$f_{\mathrm{GPS}}(\neg\alpha) = f_{\mathrm{GPS}}(\alpha);$$

$$f_{\mathrm{GPS}}(\alpha) = \alpha \quad \text{if } \alpha \text{ is a propositional sentence letter.}$$

For example, $p \vee (\neg q \rightarrow p)$ maps to $(p, (q, p))$. This is a TI abstraction.

Theorem 8.4. *If* $\alpha \in \mathrm{TH}(\Sigma_1)$, *then* $f_{\mathrm{GPS}}(\alpha) \in \mathrm{TH}(\Sigma_2)$.

The proof is by induction on the length of the proof. We just take a proof tree Π_1 of α and apply f_{GPS} to every wff in the tree. Note that the reverse of Theorem 8.4 is not true. For example $(p, p) \in \mathrm{TH}(\Sigma_2)$ but $p \wedge \neg p \notin \mathrm{TH}(\Sigma_2)$. Not every abstract theorem maps back.

Example 8.5 (*Theorem proving*). Plaisted developed a theory of `abstraction` for resolution systems [60,61]. His work is the most complete and well-developed work on the theory of abstraction developed in the past. He defines three classes of `abstractions` for refutation systems that preserve inconsistency. His first two classes, *ordinary abstractions* and *weak abstractions* map a set of clauses onto a simpler set of clauses. These abstractions are Λ/Ω-invariant abstractions; that is, the same mapping function is used to abstract the wffs and the axioms of the ground space onto those of the abstract space.

For ordinary abstractions, the mapping function maps a clause in the ground language onto a set of clauses in the abstract language subject to the following conditions (notice that we interpret the empty clause as falsity):

(a) $f(\bot) = \{\bot\}$;
(b) if α_3 is a resolvent of α_1 and α_2 in Σ_1 and $\beta_3 \in f(\alpha_3)$, then there exist $\beta_2 \in f(\alpha_2)$ and $\beta_1 \in f(\alpha_1)$ such that a resolvent of β_1 and β_2 subsumes β_3 in Σ_2;
(c) if α_1 subsumes α_2 in Σ_1 and $\beta_2 \in f(\alpha_2)$, then there exists $\beta_1 \in f(\alpha_1)$ such that β_1 subsumes β_2 in Σ_2.

Weak abstractions are identically defined to ordinary abstractions except condition (b) is weakened to the property that if α_3 is a resolvent of α_1 and α_2 in Σ_1 and $\beta_3 \in f(\alpha_3)$, then there exist $\beta_2 \in f(\alpha_2)$ and $\beta_1 \in f(\alpha_1)$ such that β_1 subsumes β_3, or β_2 subsumes β_3, or a resolvent of β_1 and β_2 subsumes β_3 in Σ_2. All ordinary abstractions are (trivially) weak, but not all weak abstractions are ordinary.

Theorem 8.6. *Weak and ordinary abstractions are NTI.*

This is a corollary to [60, Theorem 2.4, p. 268]. However, not all Λ/Ω-invariant NTI abstractions are weak or ordinary.

Theorem 8.7. *There exist NTI abstractions between resolution systems that are not weak or ordinary abstractions.*

Proof. Indeed, we can find NTI abstractions that fail every one of the three conditions in the definition of weak and ordinary abstractions.

Condition (a) is failed by the NTI abstraction for which $f(\varphi) = \{\varphi \vee \bot\}$.

The problem with condition (b) is that we may also need to resolve with an axiom of the theory. Consider, for instance, the abstraction defined by $f(p \vee q) = \{p \vee r\}$ and $f(\varphi) = \{\varphi\}$ otherwise. If Σ_1 contains the axioms, $\neg q$ and $\neg r$, then f is NTI. In particular, $p \vee q$ resolves with $\neg p$ in Σ_1 to give q. However, no clause in the abstraction of $p \vee q$ or $\neg p$ (or their

resolvent) subsumes the clause q found in the abstraction of q. We therefore fail condition (b).

For condition (c), consider the abstraction defined by $f(p \vee q) = \{r, p \vee q\}$ and $f(\varphi) = \varphi$ otherwise. Now f is NTI. However, f fails condition (c) of the definition of weak and ordinary abstractions as p subsumes $p \vee q$ but no clause in the abstraction of p subsumes r which is in $f(p \vee q)$. □

Note that the definition of weak and ordinary abstractions can be extended to overcome the counter-examples given above in the proof of Theorem 8.7. For instance, as far as the first counter-example is concerned, we could replace condition (a) with the more general requirement that there must exist $\varphi \in f(\bot)$ such that $\neg\varphi \in \mathrm{TH}(\Sigma_2)$. In general there are many NTI abstractions which are not weak or ordinary. But the right question is whether there are any NTI abstractions which are not weak or ordinary and, at the same time, are "interesting" abstractions between resolution systems. (The examples in the above proof, for instance, are not very interesting.) Weak and ordinary abstractions seem to capture most of the interesting abstractions. One exception is given by Plaisted's generalization functions (see Example 8.8). NTI abstractions which are not weak or ordinary can also be built for particular theories by allowing us to perform extra resolutions, for instance, against a subset of the axioms or against some derived theorems. ABSTRIPS, for instance, uses this idea and distinguishes between operators and the other axioms (which for instance codify the state of the world). This makes ABSTRIPS fail condition (c) above. (Notice that the abstraction which deletes all literals containing a predicate symbol p is a weak abstraction [60, p. 267]. This abstraction looks very similar to ABSTRIPS. However ABSTRIPS abstractions do not delete all the occurrences of p but only those occurring in the preconditions to operators.)

The restriction of f_{AB} to clausal form and resolution is an abstraction $f^{\mathrm{r}}_{\mathrm{AB}}$ such that:

- $f^{\mathrm{r}}_{\mathrm{AB}}(\neg p_1 \vee \cdots \vee \neg p_n \vee q) = \{\neg p_{k_1} \vee \cdots \vee \neg p_{k_m} \vee q\}$ in the case of operators, where $\{\neg p_{k_1}, \ldots, \neg p_{k_m}\} \subseteq \{\neg p_1, \ldots, \neg p_n\}$ is computed, as for f_{AB}, according to the criticality associated to preconditions;
- $f^{\mathrm{r}}_{\mathrm{AB}}(r) = \{r\}$ otherwise.

Weak and ordinary abstractions map onto sets of clauses. This mapping is actually only onto sets with one member, so we can forget the set brackets from now on. Let us concentrate on condition (c). It requires that if α_1 subsumes α_2 in Σ_1 and $\beta_2 \in f^{\mathrm{r}}_{\mathrm{AB}}(\alpha_2)$, then there exists $\beta_1 \in f^{\mathrm{r}}_{\mathrm{AB}}(\alpha_1)$ such that β_1 subsumes β_2 in Σ_2. Which is not the case. It is sufficient to consider the case where $f^{\mathrm{r}}_{\mathrm{AB}}(\neg p_1 \vee \neg p_2 \vee q) = \neg p_2 \vee q$ (where $\neg p_1 \vee \neg p_2 \vee q$ is an operator) and $f^{\mathrm{r}}_{\mathrm{AB}}(\neg p_1 \vee q) = \neg p_1 \vee q$ (where $\neg p_1 \vee q$ is not an

operator). $\neg p_1 \vee q$ subsumes $\neg p_1 \vee \neg p_2 \vee q$ but $f^{\mathrm{r}}_{\mathrm{AB}}(\neg p_1 \vee q)$ does not subsume $f^{\mathrm{r}}_{\mathrm{AB}}(\neg p_1 \vee \neg p_2 \vee q)$.

Ordinary and weak abstractions have the advantage over NTI abstractions of capturing a notion of mapping into simpler theories. The proof of [60, Theorem 2.4] shows in fact how, given a proof of a theorem in the ground space, we can construct a proof of the abstract theorem; this proof of the abstract theorem, Plaisted notes, is no longer than the original ground proof. Ordinary and weak abstractions are therefore guaranteed to satisfy the simplicity property as there will be an abstract proof no longer than the shortest ground proof.

We feel that our definitions of abstraction are "more natural" in the sense that they better reflect and capture the properties for which they are defined (mapping search spaces, preserving provability, preserving inconsistency, and so on). Also they capture a more general phenomenon than weak or ordinary abstractions (which formalize `abstractions` in refutation systems). NTI abstractions on the other hand have the disadvantage of not guaranteeing to map onto simpler theories. We have identified a subclass of the NTI abstractions, called PI (and NPI) abstractions, which have this simplicity property [24] (this topic will be covered in a following paper).

Example 8.8 (*Theorem proving*). Plaisted's third class of `abstractions` use *generalization functions* [62]. Ordinary and weak abstractions abstract the language and axioms of a theory but keep the deductive machinery the same (what we call Δ-invariant abstractions and what Plaisted called *input abstractions*). Abstractions using generalization functions, on the other hand, keep the language and axioms the same but abstract the resolution rule of inference; after every resolution, a "generalization" operation is performed on the resulting clause. For example, we might replace all terms of depth n or greater by (new) variables. We thereby construct a proof which is "more general" than one we could find in the ground theory. This general proof is guaranteed to be no longer than one of the ground proofs and can be used to aid the search for a ground proof as it has a similar structure. Note that, though the abstract proof can be shorter than the ground proof, the cost of inference in the abstract theory is more expensive than that in the ground theory. In general we have to perform both full-blown resolution and generalization.

An abstraction using a generalization function maps a first-order calculus using resolution onto another first-order calculus with the same axioms but with a "generalized resolution" rule of inference. The identity function is used to map wffs between the two formal systems. Generalization functions are therefore Λ/Ω-abstractions with f_Λ the identity function. Generalized resolution is resolution followed by application of a generalization function,

g, to the resolvent; this generalization function maps a wff, φ, onto a set of (more general) wffs that have φ as instances. In other words:

$$g(\varphi) \subseteq \{\alpha\colon\ \varphi \mapsto \alpha\theta\}.$$

(When $g(\varphi) = \{\}$, we interpret $\{\}$ as $\bot$.) If $g(\varphi) = \{\varphi\}$, the identity generalization, then Σ_1 will be identical to Σ_2, and the abstraction will trivially be TC/NTC. In general abstractions with generalization functions are TI/NTI.

Theorem 8.9. *If $f : \Sigma_1 \Rightarrow \Sigma_2$ is a generalization abstraction for which, for any φ, $g(\varphi) \neq \{\}$, then f is TI/NTI.*

The proof that f is NTI is a simple corollary to [62, Theorem 1, p. 368] (using the notation of [62], when $S = T$ and when all the deductions considered are refutations). This assumes that $g(\bot) = \bot$; this is true of all generalization functions. Since f is negation preserving, it is also TI. It is also guaranteed by [62, Theorem 1] that the abstract proof is no longer than the ground proof.

Abstractions with generalization functions are not usually TD* as generalizing a theorem can produce a nontheorem. For example, if we have the generalization function that replaces all terms by free variables and the set of axioms, $\Omega_1 = \{p, \neg p \vee q(a), \neg p \vee \neg q(b)\}$, then $\bot \in \mathrm{TH}(\Sigma_2)$ but $\bot \notin \mathrm{TH}(\Sigma_1)$. We can come up with (extreme) examples that are TD*. Consider, for instance, the formal system which contains p as an axiom and for which

$$g(\varphi) = \begin{cases} \{\bot\}, & \text{if } \varphi = \neg p, \\ \{\varphi\}, & \text{otherwise.} \end{cases}$$

This is TC/NTC as the generalization function merely replaces a wff that is inconsistent with $\bot$ itself.

8.2. Propositional abstractions

We call an abstraction *propositional* if the abstract theory is propositional. Such abstractions are very important when the abstract space is decidable. (Section 11 suggests one way to exploit the decidability of the abstract space of propositional abstractions.)

Example 8.10 (*Theorem proving*). Connection methods have been proposed in various forms as an efficient way to perform theorem proving [10,11,46]. Common to these proposals is a *connection graph* which represents possible refutations between complementary literals; the approaches differ in

how they search this graph. Most of these approaches can be treated as propositional abstractions.

Let us consider, for example, Chang [10], who describes an approach in which the connection graph is searched for a *resolution plan*, a list of possible resolutions from which we can derive $\perp$; this plan is then executed, by finding a unifier that simultaneously makes all the appropriate literals in the plan complementary. The (expensive) cost of unification is thus delayed until we have a complete plan. This approach can eliminate many redundancies of conventional resolution (e.g. simple reorderings of the resolutions) and allows all the strategies developed for resolution (like linear, set of support, ...) to be used. Indeed, the fact that we can use conventional resolution strategies is a trivial observation, once we have described this example, as we note that the abstract theory in which we construct the resolution plan merely uses a "restricted" form of propositional resolution.

We will formalize this example as a propositional Λ/Ω-invariant theory abstraction, $f : \Sigma_1 \Rightarrow \Sigma_2$, which maps from a first-order calculus using resolution to a propositional calculus using "restricted" propositional resolution. The abstraction preserves the logical structure of the wffs, mapping the atomic formulas as follows:

$$f(p(x,\ldots)) = p.$$

All atomic formulas with the same predicate symbol map onto the same propositional constant. The "restriction" on the propositional resolution in the abstract theory is that $f(\alpha \vee \beta)$ and $f(\alpha' \vee \neg\beta')$ are only allowed to resolve together in Σ_2 if β resolves with $\neg\beta'$ in Σ_1 (allowing for renaming of any common variables); this prevents us from drawing up a plan in which there is no hope of ever finding a suitable unification to make the literals complementary (e.g. $f(p(a))$ and $f(\neg p(b))$ are not allowed to unify in the abstract theory even though they map to p and $\neg p$). The resolution plan may not be of any use as no consistent substitution for the variables need exist. The links in a connection graph are just a means of pre-compiling the allowed resolutions.

Consider, for example, the problem in [10] where we wish to show that the clauses

$$\{\neg p(x) \vee p(g(x)), p(a), \neg p(g(g(a)))\}$$

are unsatisfiable. On the left, we give a proof in the abstract theory and show how this resolution plan maps onto a proof (on the right) in the ground theory by finding suitable unifications for every step suggested by the abstract proof. Not all abstract proofs will be able to map back; however, we can always find one that will.

$$\dfrac{\dfrac{\dfrac{\neg p \vee p \quad p}{p} \quad \neg p \vee p}{p} \quad \neg p}{\bot} \quad \Rightarrow \quad \dfrac{\dfrac{\dfrac{\neg p(x) \vee p(g(x)) \quad p(a)}{p(g(a))} \quad \neg p(y) \vee p(g(y))}{p(g(g(a)))} \quad \neg p(g(g(a)))}{\bot}$$

The proof in the abstract theory is not the shortest we can find; having deduced p, there is a redundant step where we resolve with $\neg p \vee p$ to deduce p again; this extra resolution is needed in the ground theory in order to make the unifications work. Presumably if we were using this abstraction for general purpose theorem proving, we would find the abstract proof without the redundant step first; being unable to find a suitable unifier to map the proof back into the ground theory, we would then backtrack and generate the abstract proof with the necessary redundant step. As redundant steps can be added without limit, this theorem proving strategy will not terminate, which is not surprising as it is complete. We may also have to resolve with the input clauses more than once, and to use multiple versions of them with distinct variables. See [10] for a longer discussion of these issues.

The fact that the abstract proof is useful as a plan for the ground proof follows from the proof that this abstraction is TI. More precisely, this proof shows that there is an abstract proof which contains the same resolutions as the ground proof. Thus, given an abstract proof, we generate the ground proof by finding a unifier that makes all the appropriate literals complementary.

Theorem 8.11. f_{chang} *is TI/NTI.*

The proof can be easily given by mapping a proof Π_1 of φ in Σ_1 onto a proof Π_2 of $f(\varphi)$ in Σ_2. We simply need to apply f to every wff in Π_1. Note that Π_2 is simpler than Π_1 even though it contains the same resolutions as Π_1 since we do not perform any unification in Σ_2.

A significant problem with all TI abstractions, this one included, is that they can map a consistent system into an inconsistent system [22,27]. The restriction on resolution in Σ_2 overcomes many problems (e.g. $\Omega_1 = \{p(a), \neg p(b)\}$ does not map into an inconsistent theory as $f(p(a))$ and $\neg f(p(b))$ are not allowed to resolve together); however, there still exist less trivial sets of axioms which map a consistent theory into an inconsistent theory (e.g. $\Omega_1 = \{p(x) \vee q(x), \neg p(a), \neg q(b)\}$).

Example 8.12 (*Decision procedures*). A propositional abstraction was used in [19] to implement a decider for first-order logic; this decides whether a (first-order) wff is derivable from a set of (first-order) wffs applying only the propositional connective inference rules. We define it as a Λ/Ω-invariant abstraction, $f_{\text{GG}} : \Sigma_1 \Rightarrow \Sigma_2$ between a first-order calculus with a

complete deductive machinery and a propositional calculus with a complete propositional decider. The mapping function used to abstract the wffs (and the axioms) is defined by:

(1) $f_{GG}(\alpha) = P_k$, where α is an atomic formula; occurrences of identical atomic formulas are rewritten as occurrences of the same propositional constant, P_k; occurrences of different atomic formulas are rewritten differently.
(2) $f_{GG}(\exists x.\alpha) = P_i$; occurrences of identical existential formulas or of existential formulas which differ only in the name of the bounded variables are rewritten as occurrences of the same propositional constant, P_i;
(3) $f_{GG}(\forall y.\alpha) = P_j$; occurrences of identical universal formulas or of universal formulas which differ only in the name of the bound variables are rewritten as occurrences of the same propositional constant P_j;
(4) $f_{GG}(\alpha \wedge \beta) = f_{GG}(\alpha) \wedge f_{GG}(\beta)$;
(5) $f_{GG}(\alpha \vee \beta) = f_{GG}(\alpha) \vee f_{GG}(\beta)$;
(6) $f_{GG}(\neg\alpha) = \neg f_{GG}(\alpha)$;
(7) $f_{GG}(\alpha \rightarrow \beta) = f_{GG}(\alpha) \rightarrow f_{GG}(\beta)$;
(8) $f_{GG}(\alpha \leftrightarrow \beta) = f_{GG}(\alpha) \leftrightarrow f_{GG}(\beta)$.

This is a TD/NTD abstraction.

Theorem 8.13. *f_{GG} is TD/NTD.*

The theorem can be proved by noticing that, for any proof in Σ_2, a proof can be built in Σ_1 which performs the same sequence of applications of inference rules. There is no problem with the different names of bound variables as we can always prove the equivalence of formulas which differ only in the names of the bound variables. The reverse of Theorem 8.13 is clearly false. For example "$\forall x.(p(x) \vee \neg p(x))$" is provable in Σ_1 but its abstraction is not in Σ_2.

8.3. Predicate abstractions

Predicate abstractions are abstractions, $f : \Sigma_1 \Rightarrow \Sigma_2$, in which distinct predicate symbols on Σ_1 are mapped onto (possibly not distinct) predicate symbols in Σ_2.

Example 8.14 (*Theorem proving, planning*). Plaisted [61] and Tenenberg [73] both consider the class of predicate abstractions which are Λ/Ω-

invariant theory abstractions between first-order calculi using resolution. Atomic formulas are mapped by a function f_{pred} for which

$$f_{\text{pred}}(p(x,\ldots)) = q(x,\ldots)$$

for all p belonging to a given class. Such predicate abstractions are TI and, since they are negation preserving, NTI. Of course, any such mapping that is 1–1 is trivially TC/NTC.

Theorem 8.15. f_{pred} *is TI/NTI.*

The proof can be given by showing that given a resolution proof Π_1 of α, we can build a resolution proof of $f_{\text{pred}}(\alpha)$. The proof proceeds by induction on the depth N of Π_1. We just take the proof Π_1 and apply f_{pred} to every wff in it. Note that the reverse (that f_{pred} is TD) is not in general true. For example, if $f_{\text{pred}}(p_1) = f_{\text{pred}}(p_2) = p$ then $p \vee \neg p \in \text{TH}(\Sigma_2)$ but $p_1 \vee \neg p_2 \notin \text{TH}(\Sigma_1)$. As with propositional abstractions, predicate abstractions can map consistent theories into inconsistent theories. For example, if $f_{\text{pred}}(p_1) = f_{\text{pred}}(p_2) = p$ and $\Omega_1 = \{p_1, \neg p_2\}$ then Σ_1 is consistent but Σ_2 is inconsistent.

Example 8.16 (*Theorem proving, planning*). To overcome the problem of inconsistent abstract spaces encountered in the last example, Tenenberg [73] has suggested a *restricted* predicate mapping. This abstraction is guaranteed to map a consistent theory, Σ_1, onto a consistent theory, Σ_2. Unfortunately, imposing the restriction involves an arbitrary amount of theorem proving in generating Σ_2 and loses the property of being TI; as a consequence it does not seem a very satisfactory solution to the problem of having inconsistent abstract spaces. As before any restricted predicate mapping that is 1–1 is (trivially) TC/NTC.

A restricted predicate mapping is a theory abstraction between first-order calculi using resolution. The same function, f_{pred}, as the (unrestricted) predicate mapping is used to map wffs in Σ_1 onto wffs in Σ_2. The "restriction" is that not every abstraction of the axioms of Σ_1 is kept in Σ_2; this restriction ensures that we don't keep the axioms that introduce inconsistency. To this end, the axioms of Σ_2 are given by $g(\Omega_1)$ where:

$$g(\Omega_1) = \{f_{\text{pred}}(\varphi):\ \varphi \in \Omega_1 \text{ and } (\varphi \text{ is a positive clause or for every } \zeta \text{ such that } f_{\text{pred}}(\zeta) = f_{\text{pred}}(\varphi) \text{ we have } \Omega_1 \vdash_{\Sigma_1} \zeta)\}.$$

In other words g preserves all the axioms φ which satisfy at least one of two conditions. The first is that they are positive clauses; the second is that $g(\varphi)$, to be kept as an axiom, must be such that all the formulas ζ which abstract into $g(\varphi)$ (that is, that have the same abstraction as φ) are

theorems of the ground space. Note that to determine which axioms from Σ_1 we can include in Σ_2 requires an arbitrary amount of theorem proving in Σ_1 (deciding whether $\Omega_1 \vdash_{\Sigma_1} \zeta$). The purpose of this inference is to guarantee that consistency is preserved.

Theorem 8.17. *If $f : \Sigma_1 \to \Sigma_2$ is a restricted predicate mapping and Σ_1 is consistent, then Σ_2 is consistent.*

In [73], Tenenberg demonstrates how, given a model of the axioms of Σ_1, you can construct a model of the axioms of Σ_2. Since a set of clauses is consistent iff it is satisfiable this proves that if Σ_1 is consistent then Σ_2 is as well. As a simple corollary to this theorem, for every clause derivable in the abstract theory, some clause in the ground theory that abstracts to it is also derivable.

Corollary 8.18. *If f is a restricted predicate mapping and $\varphi_2 \in \mathrm{TH}(\Sigma_2)$, then there exists φ_1 such that $f(\varphi_1) = \varphi_2$ and $\varphi_1 \in \mathrm{TH}(\Sigma_1)$.*

Note that the property described in Corollary 8.18 is not the same as being TD; all TD abstractions satisfy this property, but not all restricted predicate mappings are TD. Consider, for instance, the restricted predicate mapping $f_{\mathrm{pred}}(p_1) = f_{\mathrm{pred}}(p_2) = p$ and $\Omega_1 = \{p_1\}$ then $f_{\mathrm{pred}}(p_2) \in \mathrm{TH}(\Sigma_2)$ but $p_2 \notin \mathrm{TH}(\Sigma_1)$. If, however, the definition of restricted predicate mappings is strengthened slightly, then they can be made TD. Indeed, we merely have to remove the condition in the definition of the g that allows positive clauses into the abstract axiom set regardless. That is, the axioms of Σ_2 are given by the set $g'(\Omega_1)$ where:

$$g'(\Omega_1) = \{f_{\mathrm{pred}}(\varphi): \ \varphi \in \Omega_1 \text{ and for every } \zeta \text{ such that } f_{\mathrm{pred}}(\zeta) = f_{\mathrm{pred}}(\varphi) \text{ we have } \Omega_1 \vdash_{\Sigma_1} \zeta\}.$$

We shall call such abstractions *very restricted predicate mappings*; unlike restricted predicate mappings, they are TD.

Theorem 8.19. *Very restricted predicate mappings are TD.*

Proof. Given a proof tree Π_2 in Σ_2 that ends in $f_{\mathrm{pred}}(\varphi)$ we show how you can construct a proof tree Π_1 in Σ_1 that ends in φ. Note that φ is not necessarily $\perp$. The proof proceeds by induction on the depth (that is, the length of the longest branch), N, of Π_2.

If $N = 1$, then $f_{\mathrm{pred}}(\varphi)$ is an axiom of Σ_2. From the definition of the axioms of Σ_2, φ is either an axiom of Σ_1 or $\varphi \in \mathrm{TH}(\Sigma_1)$.

Assume we have shown it for all proof trees up to depth M. Consider a proof tree Π_2 of depth $M + 1$ that ends in $f_{\mathrm{pred}}(\varphi)$ and in which $f_{\mathrm{pred}}(\varphi)$ is

the resolvent of $c \vee q_1 \vee \cdots \vee q_n$ and $c' \vee \neg q_1' \vee \cdots \vee \neg q_n'$. That is, $f_{\text{pred}}(\varphi) = (c \vee c')\theta$ where θ is a most general unifier and $q_i\theta \equiv q_i'\theta$ for $1 \leqslant i \leqslant n$. Let q_i have a predicate symbol r. Then, as they are complementary literals, q_i' must have the same predicate symbol. Construct the wffs p_i and p_i' which map to q_i and q_i' and which both have the same predicate symbol $s \in R_r$. Now, p_i and $\neg p_i'$ will resolve together with unifier θ (f_{pred} is transparent to substitutions). Pick the wffs d and d' which map to c and c' such that $\varphi \equiv (d \vee d')\theta$. By the induction hypothesis, we can prove $d \vee p_1 \vee \cdots \vee p_n$ and $d' \vee \neg p_1' \vee \cdots \vee \neg p_n'$. Finally, note that $d \vee p_1 \vee \cdots \vee p_n$ and $d' \vee \neg p_1' \vee \cdots \vee \neg p_n'$ will successfull resolve together with unifier θ to give φ. □

A major problem with (very) restricted predicate mappings, unlike their unrestricted counterparts, is that (unless they are 1–1) they are not TI. We therefore lose "completeness" as there is bound to be a wff, α, which is provable in Σ_1 but whose abstraction is not provable in Σ_2.

The other major problem with restricted predicate mappings is that determining which axioms to include in the abstract theory is, in general, undecidable. This makes them not very interesting even for uses outside the `abstraction` tradition. One solution suggested in [73] is to weaken the derivability condition in the definition of g (namely, $\Omega_1 \vdash_{\Sigma_1} \zeta$) to a condition of derivability within certain resources. This conservative approach would construct an abstract theory weaker than it theoretically needs to be to preserve consistency (that is, an abstract theory with fewer theorems). (See [74] for other similar solutions to this problem.)

8.4. Domain abstractions

Domain abstractions are abstractions which map the domain (that is, the constants) of the ground theory onto a (smaller) domain in the abstract theory.

Example 8.20 (*Common sense reasoning*). Hobbs has suggested [31] a theory of *granularity* in which a complex theory is abstracted onto a simpler, more "coarse-grained" theory with a smaller domain. For example, the real world of continuous time and positions could be mapped onto a (micro)world of discrete time and positions. Granularity can be formalized as a Λ/Ω-invariant theory abstraction (let us call it "$f_{\text{gran}} : \Sigma_1 \Rightarrow \Sigma_2$") between first-order calculi. Different constants in Σ_1 are mapped onto (not necessarily different) constants in Σ_1 according to an *indistinguishability relation*, "$\sim$" defined by the (second-order) axiom,

$$\forall x, y . x \sim y \leftrightarrow \forall p \in R . p(x) \leftrightarrow p(y),$$

where R is the subset of the predicates of the theory determined to be *relevant* to the situation at hand. As in [31], we define indistinguishability for unary predicates; it can, however, be easily generalized to n-ary predicates. The mapping function keeps the same logical structure of wffs (it is a theory abstraction) but translates any constant into its equivalence class, namely

$$f_{\text{gran}}(p(a)) = p(\kappa(a)),$$

where a is any constant symbol and $\kappa(a)$ is the constant in the abstract language representing the equivalence class of the constant a with respect to the indistinguishability relation; that is,

$$\kappa(x) = \{y\colon\ x \sim y\}.$$

Such an abstraction is TI.

Theorem 8.21. f_{gran} *is TI/NTI.*

In fact we can map a proof tree Π_1 of φ onto a proof tree Π_2 of $f_{\text{gran}}(\varphi)$ merely by applying f_{gran} to every wff in the proof tree. Like any TI abstraction that is not TC, f_{gran} can map a consistent theory onto an inconsistent theory. For example, if the constants a and b are "indistinguishable" and $\{a, b\}$ represents the equivalence class of a and b, then a consistent ground theory with equality and the axiom $\neg(a = b)$ maps into an inconsistent abstract theory with the axiom $\neg(\{a, b\} = \{a, b\})$. The consistency of the abstract theory can be guaranteed if we define a suitable indistinguishability relation.

Theorem 8.22. f_{gran} *preserves consistency if indistinguishability is defined over all predicates.*

Proof. By contradiction. Assume that a consistent theory, Σ_1, maps onto an inconsistent theory, Σ_2. That is, we can find a proof tree, Π_2, of $\perp$. We can then construct a proof tree, Π_1, of $\perp$ in Σ_1, contradicting the assumption that Σ_1 is consistent. For every equivalence class, $\kappa(a)$, we pick one member of that class, b; to every wff, φ, in Π_2 we apply the substitutions $\{\kappa(a)/b\},\ldots$. This will generate a proof tree, Π_1, whose assumptions will either be axioms of Σ_1 or will be derivable from them using the indistinguishability relation and substitution of equivalences. If indistinguishability is not defined over *all* predicates, this last fact will not necessarily be true. □

This theorem might seem to contradict our claim that, for every TI abstraction which is not TC, there exist a choice of axioms that map a consistent theory into an inconsistent theory. The obvious solution is that, if indistinguishability is defined over all predicates, then f_{gran} is TC (a TC abstraction can never map a consistent theory into an inconsistent one).

Finally, we note that f_{gran} is just a special case of the example of Plaisted's weak and ordinary abstractions described in [60,61] where function symbols (constants are 0-ary function symbols) are renamed in a systematic (but not necessarily 1–1) way.

Example 8.23 (*Approximate reasoning*). Abstraction has been proposed by Imielinski [32] as a basis for "approximate" methods of reasoning with low complexity. Though such methods can return answers that are not always correct, their errors should be characterizable. As an example, Imielinski considers two *domain abstractions*; the first is TI while the second is TD. However, there is a subset of the language over which both abstractions agree with the ground theory; that is, a subset of the language with respect to which both abstractions are TC.

Like granularity, Imielinski's TI abstraction maps objects in the domain onto their equivalence classes. In granularity, the equivalence classes consist of objects that can be shown to be *indistinguishable*. Imielinski's TI abstraction allows arbitrary equivalence relations. However, the language of the ground theory is restricted to a subset of first-order predicate logic sufficient to express queries to a database. We formalize Imielinski's TI abstraction, $f_1 : \Sigma_1 \Rightarrow \Sigma_2$, as a Λ/Ω-invariant theory abstraction between first-order database-like theories. The languages of these theories consist of closed formulas containing only the $\exists$ quantifier, and the $\wedge$ and $\vee$ connectives. The mapping function, f_1, used to abstract the language and the individual axioms of Σ_1 is defined by

$$f_1(p(a)) = p(\kappa(a)),$$

where a is any constant symbol, $\kappa(a)$ is the equivalence class of the constant a with respect to an indistinguishability relation, $\sim$, and

$$f_1(p(x)) = p(x),$$

where x is any variable. The mapping is extended to n-ary predicates in the obvious way. See [31,32] for some examples of indistinguishability relations. Analogously to the granularity abstraction f_{gran}, this abstraction is TI. A TI abstraction *overestimates* answers; that is, the abstract theory will suggest certain wffs are true which are not, in fact, true in the ground space. The problem of inconsistent abstract spaces is not relevant for this type of abstraction as the ground and abstract languages lack negation. However, we will often use a closed world assumption to deduce negative information from such a database, in which case we would have to worry about the consistency of the abstract space.

Example 8.24 (*Approximate reasoning*). Imielinski's second domain abstraction [32], which we will call f_2, is a TD abstraction. Such an abstraction underestimates answers; that is, the abstract space will suggest certain wffs are false which are, in fact, true in the ground space. Though f_2 is not complete, it is sound; all the abstract theorems it returns are guaranteed to be theorems of the ground space. For this abstraction, properties true of an object in the ground space hold in the abstract space for some unidentified member of the (possibly larger) equivalence class of the object.

We define f_2 as a Λ/Ω-invariant abstraction from a first-order database-like theory (whose language consists of closed formulas containing only the $\exists$ quantifier and the $\wedge$ and $\vee$ connectives) to another first-order theory (whose language consists of closed disjunctive normal formulas containing only the $\exists$ quantifier and the $\wedge$ and $\vee$ connectives). The language of Σ_2 differs in two ways from the abstracted language of the previous example: its domain is the same as that of Λ_1 (and not the equivalence classes) and it contains a new unary predicate for each equivalence class, $[\kappa(a)](x)$, true for every member x of the equivalence class $\kappa(a)$; we will use "$[\kappa(a)]$" to represent the name of this predicate.

The mapping function, f_2, is used to abstract the language (and the individual axioms) of Σ_1. All wffs of Λ_1 and Ω_1 are first transformed into disjunctive normal form. Then for each disjunct φ we have:

(a) If φ contains the constant a, then it is replaced by an existentially quantified variable restricted to members of a's equivalence class (that is, φ is rewritten to $\exists x.[\kappa(a)](x) \wedge \varphi\{x/a\}$). The idea is to replace each named constant by an arbitrary member of its equivalence class. The domain of Σ_2 is thus no smaller than that of Σ_1; however, the properties true in Σ_1 of an object a are merely true in Σ_2 of *some* object in the equivalence class of a.

(b) If φ contains multiple occurrences of an existentially quantified variable, each occurrence is replaced by a new existentially quantified variable. The occurrences of an existentially quantified variable can thereby represent different members of an equivalence class.

For example, $\exists x.p(a,x) \wedge p(b,x)$ is mapped to

$$\exists x,u,w.[\kappa(a)](u) \wedge p(u,x) \wedge [\kappa(b)](w) \wedge p(w,x)$$

by step (a), and then to

$$\exists u,v,w,y.[\kappa(a)](u) \wedge p(u,v) \wedge [\kappa(b)](w) \wedge p(w,y)$$

by step (b). This abstraction is TD.

Theorem 8.25. *f_2 is TD.*

Proof. By showing that the axioms of Σ_2 follow from a set of axioms logically equivalent to the axioms of Σ_1. $\mathrm{TH}(\Sigma_2)$ is therefore a subset of $\mathrm{TH}(\Sigma_1)$. First transform each axiom in Ω_1 into (the equivalent) disjunctive normal form. Then for each disjunct φ, if φ contains the constant a, replace φ by the equivalent formula, $\exists x.[\kappa(a)](x) \wedge \varphi\{x/a\} \wedge x = a$. If φ contains multiple occurrences of an existentially quantified variable, replace each occurrence by a new existentially quantified variable and add the equality condition that these new existentially quantified variables are equal. The resulting axioms are logically equivalent to those of Σ_1. The axioms of Σ_2 are just the result of dropping from these clauses all the equality conditions added for the constants and existentially quantified variables. □

Note that the language of the abstract space is more complex than the language of the ground space, and that the axioms of the abstract space are more complicated than the axioms of the ground space. It therefore seems doubtful that theorems will be easier to solve in the abstract space than in the ground space. The only saving is that instead of having to show a property true for a, we merely have to find it true for any of the members of $\kappa(a)$ (and one of these proofs might be easier to find).

Imielinski's two domain abstractions, one TI and the other TD, provide upper and lower bounds on the theorems of the database. That is, the answers returned by f_2 are a subset of the theorems of the ground space while those returned by f_1 are a superset. Imielinski is also able to characterize a subset of the language of Σ over which $f_1(\Sigma)$ and $f_2(\Sigma)$ agree exactly with Σ. Consider the "propositional" subset, Γ, of the language of Σ_1; that is, those wffs in Λ_1 without existential quantifiers. The following result holds.

Theorem 8.26. *f_1 and f_2 are TC with respect to Γ.*

(Here an abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is said to be TC with respect to Γ iff for any $\varphi \in \Gamma$, $\varphi \in \mathrm{TH}(\Sigma_1)$ iff $f(\varphi) \in \mathrm{TH}(\Sigma_2)$.) This theorem is stated as [32, Lemma 4].

8.5. Metatheoretic results

This section describes some important metatheoretic results which can be captured as abstractions.

Example 8.27 (*Theorem proving*). An important class of abstractions introduced by Plaisted [60,61] is the class of the so-called *ground abstractions.* (Note that here the word "ground" is used with a meaning different from before. This is due to an historical accident. We could have eliminated the confusion by changing the terminology, e.g. by talking of the "concrete

space" instead of the "ground space". As the two uses of the word "ground" overlap in a very limited number of places and in such a way that the context always makes clear the correct meaning, we have preferred not to do so.)

Ground abstractions are Λ/Ω-invariant abstractions from a first-order calculus to a formal system with a variable and quantifier-free first-order language and a complete propositional decider that treats ground atomic formulas as propositions. The mapping function, f_{ground} is used to abstract the wffs (and the axioms) of Σ_1 onto those of Σ_2. f_{ground} can be defined to work for refutation and provability systems. We consider the second case, the first case is dual. $f_{\text{ground}}(\varphi)$ first skolemizes φ; for provability systems this means replacing universally quantified variables and free variables by skolem functions and constants in the usual way. This mapping produces a formula which is provable if and only if φ is. The second step performed by f_{ground} is to build a disjunction of some ground instances of the skolemized wff, each ground instance obtained by substituting the existential variables with elements of the Herbrand Universe. That is, if we write $\bigvee_i p_i$ to mean a disjunction of formulas,

$$f_{\text{ground}}(\varphi) = \bigvee \varphi_{\text{g}}^*$$

where the φ_{g}^* are ground skolemized instances of φ, possibly infinite in number.

Theorem 8.28. f_{ground} *is TD.*

In fact f_{ground} is TC if we consider all the ground instances of φ. This is a corollary to Herbrand's theorem [30]. For Herbrand's theorem an existential formula is provable iff a finite disjunction of its ground instances is, which in turn is provable iff the (possibly infinite) disjunction of all its ground instances is. f_{ground} can be made TD but not TC simply by dropping some ground instances (if a wff is a theorem, then any disjunction containing it is). Dropping instances implements the idea of approximating a theory. A smaller subset gives more savings in the cost of theorem proving in the abstract space but, at the same time, loses more theorems. The TC version of f_{ground} has been used in [19] to build a complete decider for a class containing the class of UE-formulas (that is, those formulas whose prefix contains a sequence of universal quantifiers followed by a sequence of existential quantifiers).

Notice that f_{ground} is not negation preserving. Therefore the fact that it is TD does not imply that it is NTD. Indeed in general it is not. On the other hand the version of f_{ground} defined for refutation systems, let us call it $f_{\text{ground}}^{\text{r}}$, is NTD. The NTC version of $f_{\text{ground}}^{\text{r}}$ (that is, that considering all

the ground instances) has been proposed as an NTI abstraction by Plaisted [60].

Example 8.29 (*Theorem proving*). *Semantic abstractions* can be constructed because of the soundness theorem for a logic. They can be defined for any logic which has a model-theoretic semantics. The soundness theorem for a logic states:

Theorem 8.30 (Soundness). *If* $\vdash \varphi$, *then for all interpretations, I, which are models of the axioms,* $\models_I \varphi$.

The truth of a wff can be computed in an interpretation after providing a universe of objects for variables to range over, calculation procedures (computer programs in effect) for functions and predicates, plus the standard interpretations for connectives and quantifiers. The computation in the model can be formalized as deduction in a formal system consisting of some variation of the lambda calculus (this has been actually done to formalize the use of the simulation structure inside the FOL system [28,59,72,79]). The soundness theorem thus expresses a TI abstraction between a formal system in which we prove theorems and a formal system in which we compute truth in an interpretation (which constitutes the intended model).

The abductive use of semantic abstractions is not very interesting as it is not obvious how we could use a computation in the model to drive the search for a proof in the theory. Also this computation needs usually an infinite amount of time (the domains of interpretation are in general infinite). Semantic abstractions are instead useful when used deductively to prune the search space. It follows from the soundness theorem that:

Lemma 8.31. *If there exists a model I in which* $\not\models_I \varphi$, *then* $\not\vdash \varphi$.

Hence, if a (sub)goal turns out to be false in a given model then we know that this (sub)goal will not be provable and can be deleted from the search space. For example, Gelernter [16] in his famous geometry theorem prover and Reiter [68] in an incomplete natural deduction theorem prover used such counter-examples to remove unachievable subgoals. Semantics have been used extensively in the literature, sometimes in more sophisticated ways. For example, in [2,68] models are used to suggest the instantiation of variables. Plaisted [60,61] proposes a class of "semantic abstractions" for resolution theorem proving which falls half way between the ground abstractions and the semantic abstractions described here. In this class of ordinary abstractions, clauses are grounded and each term (but not predicate) is replaced by its interpretation; normal (propositional) resolution is then applied to these abstracted clauses.

8.6. Summary

The properties of the abstractions described in the examples in this section are summarized in Table 1. We will make some general remarks about this table. We believe that this is a representative sample of abstractions in general and of abstractions in particular.

First, all the examples preserve provability in one way or another; almost all are, in fact, TI* abstractions. This supports our emphasis on provability preserving abstractions, and more especially on TI* abstractions. Second, the majority of these examples are negation preserving and theory abstractions. Indeed, we would argue that you should only change the logical structure of wffs with great care. It is the theory, not the logic, that needs taming. Third, almost all the abstractions use the same deductive machinery in the ground and the abstract spaces. This makes implementation very economical. It also allows us to use hierarchies of abstractions. Fourth, nearly all abstractions are Λ/Ω-invariant. This, of course, guarantees that the abstraction of any wff that is an axiom of the ground space is itself an axiom of the abstract space. Fifth, all the TI* abstractions but the semantic abstraction are used abductively (to give a complete theorem proving strategy), while TD* abstractions are used both deductively (to give a sound theorem proving strategy) and abductively. Finally, most of the (theory) abstractions can be characterized by whether they map the terms or the predicate names. Since theory abstractions can only map the atomic formulas, this is perhaps not so surprising.

9. Inconsistent abstract spaces

Implicit in most work in abstraction is the assumption that the ground space is consistent. If this is not so, any abstraction is trivially TD/NTD.

Lemma 9.1. *If Σ_1 is inconsistent, then any abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is a TD/NTD abstraction.*

Another common assumption is that an abstraction maps a (consistent) formal system onto a consistent formal system. If this is not the case, then the following fact holds:

Lemma 9.2. *If $f : \Sigma_1 \Rightarrow \Sigma_2$ is an abstraction and Σ_2 is inconsistent, then it is a TI/NTI abstraction.*

The results of these two lemmas can be trivially composed. Thus, if Σ_1 and Σ_2 are both inconsistent then $f : \Sigma_1 \Rightarrow \Sigma_2$ is a TC/TD/TI/NTC/NTD/NTI

Table 1
Summary of abstractions.[a]

Abstraction	Example	T*/NT*	NP	TA	invariant			used	maps
					Δ	A/Ω	Σ		P/T
ABSTRIPS	8.1	TI	√	√	√	√	×	abd	N
GPS	8.3	TI	×	×	×	√	√	abd	N
Weak	8.5	NTI	?	?	√	√	?	abd	?
Ordinary	8.5	NTI	?	?	√	√	?	abd	?
Generalization	8.8	TI	√	√	×	√	×	abd	T
Chang	8.10	TI	√	√	√	√	√	abd	T
Decider[b]	8.12	TD	√	×	√	√	×	ded	N
Predicate	8.14	TI	√	√	√	√	√	abd	P
Restricted predicate[c]	8.16	TD	√	√	√	×	×	abd	P
Granularity	8.20	TI	√	√	√	√	√	abd	T
Imielinski	8.23	TI	√	√	√	√	√	abd	T
Imielinski	8.24	TD	√	×	√	√	×	ded	T
Ground	8.27	TD/NTD	×	×	√	√	×	a-d	T
Semantic[d]	8.29	TI	×	×	♯	♯	♯	ded	N

[a] In the column headings, NP stands for "negation preserving", TA for "theory abstraction", "maps P/T" for "maps the predicate names or the terms".

In the table entries, "√" means that this abstraction has this property, "×" that this abstraction does not have this property, "?" that abstraction can have this property (but does not necessarily have to), "♯" that this property is not relevant for this abstraction, "abd" that this abstraction was used abductively, "ded" that this abstraction was used deductively, "a-d" that this abstraction was used both deductively and abductively, "P" that this abstraction maps just the predicate names, "T' that this abstraction maps just the terms, and "N" if neither is true.

[b] The propositional decider is not strictly speaking Δ-invariant since (in the actual implementation of the decider) theorem proving in the abstract theory is by way of truth tables and not the propositional natural deduction rules of the ground theory.

[c] Restricted predicate abstractions satisfy a property very close to that of TD abstractions. It is only *very restricted predicate abstractions* that are actually TD.

[d] Since the abstract theory with semantic abstractions is one in which we compute the truth of a wff with respect to an interpretation, it is not very appropriate to discuss the various Δ-, A/Ω- and Σ-invariant properties of this abstraction.

abstraction. All these observations are neither deep nor particularly significant. The ground space is usually consistent even if in most cases this is not explicitly verified. The interesting question is whether something can be said about the consistency of the abstract space under the assumption that the ground space is consistent. The following fact holds.

Theorem 9.3. *If Σ_1 is consistent and $f : \Sigma_1 \Rightarrow \Sigma_2$ is a TD* abstraction, then Σ_2 is consistent.*

Proof. Proof by contradiction. Assume $\mathrm{TH}(\Sigma_2) = \Lambda_2$. From the totality of f_A, for any wff φ, $f_A(\varphi)$ is defined. But for any $f_A(\varphi)$, $f_A(\varphi) \in \mathrm{TH}(\Sigma_2)$. This means that, for any wff φ, $f_A(\varphi) \in \mathrm{TH}(\Sigma_2)$. Since f is TD, for any wff φ, $\varphi \in \mathrm{TH}(\Sigma_1)$. This contradicts the assumption that Σ_1 is consistent. The proof for NTD abstractions is very similar. □

TD* (and therefore TC*) abstractions always preserve the consistency of the ground space. However, this is not true for TI* abstractions; the abstract space can be inconsistent even though the ground space is consistent. Several examples of inconsistent abstract spaces have been given in the previous section. For instance, granularity (Example 8.20) and predicate abstractions (Example 8.14). For the ABSTRIPS abstraction, f_{AB}, consider the abstraction which drops the second conjunct of the two operators "$\alpha_1 \wedge \alpha_2 \rightarrow \alpha_3$" and "$\alpha_1 \wedge \alpha_4 \rightarrow \neg\alpha_3$", where α_1 is a theorem, and α_2 and α_4 are not both theorems. Examples of consistent abstract spaces can be trivially found (see previous section).

We have argued that TI* abstractions are the appropriate `abstractions` to be used. However, we have just shown how they can map a consistent ground theory onto an inconsistent abstract theory. This is a major problem for the use of TI* abstractions.

The problem of inconsistent abstract spaces was first identified by Nilsson [58] for ABSTRIPS abstractions. Tenenberg [73] identified the same problem for predicate abstractions, calling it the "false proof problem". (In [61], Plaisted talks of the problem of "false proofs" meaning abstract proofs which do not map back. The two problems are unrelated.) In [27] we show that the problem is even more general than either Nilsson or Tenenberg claimed—it can happen under certain very weak restrictions with TI abstractions (an thus with `abstractions`). The intuitive interpretation of why the problem occurs is as follows. In order to build an abstract space "simpler" than the ground space, the trick is to "forget" some "irrelevant" details and to keep around what is judged important. The problem is that the irrelevant looking details may be exactly those that preserve the theory from being inconsistent (thus making them not so irrelevant).

This is a major blow to the use of TI* abstractions as no proof which makes use of the inconsistency of the abstract space will map back. From an implementational point of view, at a first sight, it might not be obvious that an inconsistent abstract space is going to be such a bad thing. After all with TI* abstractions, it is never guaranteed that the abstract proofs map back. Besides the theoretical objections, which we think are of substantial importance, we argue that the problem is also very relevant from an implementational point of view. If the theorem proving strategy in the abstract space is not suitably constrained, a lot of inefficiency could be introduced, and reasoning with abstraction could become less efficient than reasoning

without abstraction. Additionally, if the abstract space uses refutation, we will not know if inconsistency comes from the fact our goal is true or from the fact that our abstract axioms are inconsistent. To make the problem even more serious, it is in general not possible to decide in a finite amount of time whether a formal system is consistent.

When working with a fixed formal system, one solution is to build abstractions which are proved a priori to construct a consistent abstract space. Often, however, the axioms are not fixed in advance. In such a situation, the ideal solution would be to find sufficient conditions which guarantee that whatever the axioms, the abstraction maps a consistent ground space onto a consistent abstract space. In [27] (or in the shorter version [22]) we show that this request turns out to be unsatisfiable. In fact, under certain very weak restrictions (satisfied by almost all `abstractions`), for (the mappings of) all the TI* abstractions which are not TC* *there always exists a set of consistent axioms whose abstract space is inconsistent.* This problem should therefore be a major consideration for anyone proposing the use of a TI* abstraction. A discussion of a possible solution to this problem requires further machinery which has not yet been introduced and it is therefore postponed to the end of Section 12.

10. Operations on abstractions

We have defined an abstraction as a mathematical object; that is, as a pair of formal systems and a mapping function. This has the advantage of allowing us to define mathematical operations (like composition and inversion) and to use them to formalize certain steps which are performed when using abstraction inside a theorem prover. In the following we give these definitions only considering f_A. In order to tackle the problem of how to use `abstractions` (when we need to describe how proof trees are mapped from the ground space to the abstract space and vice versa) we need to give stronger notions which consider also the mappings of the axioms and of the deductive machinery. These extensions are a trivial extension of the ones presented here and will be described in a following document.

First we define what it means for two abstractions to be "equal".

Definition 10.1 (*Equality*). If $f : \Sigma_1 \Rightarrow \Sigma_2$ and $g : \Sigma_1 \Rightarrow \Sigma_2$ are abstractions, then f is equal to g iff their mapping functions are identical, that is, iff $f_A = g_A$.

The notion of equality between abstractions, which relies on the notion of functional (extensional) equality, allows us to identify abstractions whose f_A's are given different intensional definitions.

We also need a notion of identity abstraction.

Definition 10.2 (*Identity*). If $f : \Sigma \Rightarrow \Sigma$ is an abstraction, then f is called the identity abstraction of Σ iff the mapping function is the identity function.

The identity abstraction maps any formal system, Σ into itself and it is uniquely determined by it. An identity abstraction is trivially TC/NTC.

Composition is a very important operation.

Definition 10.3 (*Composition*). If $f : \Sigma_1 \Rightarrow \Sigma_2$ and $g : \Sigma_2 \Rightarrow \Sigma_3$ are abstractions, then $f \circ g : \Sigma_1 \Rightarrow \Sigma_3$ is the abstraction composition of f and g with the mapping function $f_A \circ g_A$.

The composition of two abstractions (when possible) is itself an abstraction. It is therefore a useful way of constructing new abstractions from old ones. The composition of two abstractions will give an abstraction $f \circ g$ that is at least as "strong" as either of the individual abstractions, f or g as it throws away the same information as f and the same information as g.

Sometimes the proof of the abstract goal is not enough as the proof of the ground goal is also needed. The typical example is `abstraction` where the abstract space is used abductively, that is to suggest theorems or nontheorems (see Section 5). Another example is when the ground proof is a plan to be executed [74]. In these cases, we need to unabstract certain objects of the abstract space, for instance the wffs occurring in a proof, to their corresponding objects in the ground space. To formalize (part of) this notion we introduce the notion of inverse of an abstraction. Not all abstractions have inverses; in fact, it is only possible to invert those abstractions which do not throw away any information. To be more precise, an abstraction must be injective if it is to be invertible. (Remember that we are only considering surjective abstractions.)

Definition 10.4 (*Injectivity*). An abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ is injective iff if $\alpha \neq \beta$ then $f(\alpha) \neq f(\beta)$.

Definition 10.5 (*Inverse*). If $f : \Sigma_1 \Rightarrow \Sigma_2$ is injective, then the abstraction g, $g : \Sigma_2 \Rightarrow \Sigma_1$ such that $f \circ g = f_I$, where f_I is the identity abstraction of Σ_1, is called the inverse of f, written f^{-1}.

Notice that the inverse of an abstraction is itself an abstraction and that it is uniquely determined by f (if $g : \Sigma_2 \Rightarrow \Sigma_1$ and $h : \Sigma_2 \Rightarrow \Sigma_1$ are both inverses of $f : \Sigma_1 \Rightarrow \Sigma_2$, then $g = h$). Since they cannot throw away any information from the language, injective abstractions do not in general give a simpler abstract theory and are not of much practical use as `abstractions`.

In general, in the mapping back (for instance of a proof) from the abstract to the ground space there are many choices, and it is part of the theorem proving strategy to decide which to try first. This topic is discussed in [24]. A lot of the TC* abstractions are injective (for instance the TC* ground abstraction is injective modulo the renaming of the free and bound variable in the ground space, see Example 8.27).

The inverse and the composition of two abstractions are abstractions. In general we require stronger properties. For instance, we need to know how composition and inverse affect the preservation of provability and inconsistency. For instance, if we know that the composition of two `abstractions` is an `abstraction` then composition provides us with a tool for constructing new `abstractions`.

The inverse of an abstraction f^{-1}, not too surprisingly, has the inverse provability or inconsistency preserving property to f. In other words a TI* abstraction inverts to a TD* abstraction (and vice versa).

Theorem 10.6. *If $f : \Sigma_1 \Rightarrow \Sigma_2$ is a TI* abstraction (TD* abstraction) which admits an inverse f^{-1}, then f^{-1} is a TD* abstraction (TI* abstraction).*

Proof. We consider the case where f is TI. The other cases are analogous. If $\varphi \in \mathrm{TH}(\Sigma_1)$, then $f(\varphi) \in \mathrm{TH}(\Sigma_2)$. Now $f^{-1}(f(\varphi)) = \varphi$. Thus, $f^{-1}(f(\varphi)) \in \mathrm{TH}(\Sigma_1)$ implies $f(\varphi) \in \mathrm{TH}(\Sigma_2)$. Since f is surjective, $f(\varphi)$ ranges over the whole language of Σ_2. Thus $f^{-1} : \Sigma_2 \Rightarrow \Sigma_1$ is a TD abstraction. □

The composition of two abstractions preserves provability or inconsistency in the same way as its components. For example, the composition of two TI abstractions is itself a TI abstraction.

Theorem 10.7. *If $f : \Sigma_1 \Rightarrow \Sigma_2$ and $g : \Sigma_2 \Rightarrow \Sigma_3$ are TI* abstractions (TD* abstractions), then $f \circ g : \Sigma_1 \Rightarrow \Sigma_3$ is also a TI* abstraction (TD* abstraction).*

Proof. We consider the case of TI abstractions. The other cases are analogous. If f is a TI abstraction and $\varphi \in \mathrm{TH}(\Sigma_1)$, then $f(\varphi) \in \mathrm{TH}(\Sigma_2)$. However, g is also TI so if $f(\varphi) \in \mathrm{TH}(\Sigma_2)$ then $g(f(\varphi)) \in \mathrm{TH}(\Sigma_3)$. Thus, if $\varphi \in \mathrm{TH}(\Sigma_1)$ then $f \circ g(\varphi) \in \mathrm{TH}(\Sigma_3)$. □

Composing abstractions which preserve provability or inconsistency in different ways is definitely "not safe". Consider, for example, when f is a TI abstraction and g is an identity abstraction. Now $f \circ g$ is equal to f, and is thus TI. However, being a TI abstraction gives no guarantee that the

Table 2
Composition of abstractions.

	TD	TC	TI	NTD	NTC	NTI
TD	TD	TD	?	?	?	?
TC	TD	TC	TI	?	?	?
TI	?	TI	TI	?	?	?
NTD	?	?	?	NTD	NTD	?
NTC	?	?	?	NTD	NTC	NTI
NTI	?	?	?	?	NTI	NTI

abstraction maps nontheorems in a helpful way (except in syntactically complete systems); similarly, being NTI gives no guarantee that the abstraction maps theorems in a helpful way. Thus composing TI abstractions with NTI abstractions can give unpredictable effects; the resulting abstraction may be TI, NTI, or neither.

We summarize all these results (and more) in Table 2. This table describes the properties of an abstraction formed by composing an abstraction possessing the property given by the row heading with an abstraction possessing the property given by the column heading. Thus, the entry in the second row and third column indicates that composing a TC abstraction with a TI abstraction gives another TI abstraction. The symbol "?" is used to indicate that the provability or inconsistency preserving properties of the abstraction composition is not predictable. Note that only eight question marks would appear in a table for mappings between syntactically complete formal systems (in which NTI is the same as TD, etc.); however, such a table would contain much redundancy and could be represented by just one quadrant of this full table.

Two observations are worthwhile. First, in the case of `abstractions`, preserving provability is not enough as the similarity of proofs must be preserved as well. To guarantee that the composition of two abstractions preserves this property (and therefore that it is itself an abstraction) Plaisted's abstraction mappings [61] additionally have to preserve subsumption (condition (c)). We do not need this additional requirement here, but we use a definition similar in spirit (called *tree subsumption*) when studying how to use TI* abstractions as `abstractions` [24]. Second, notice that abstraction composition can be iterated. Does such iteration always converge? That is, do we reach a fixed point? This question will be dealt with in Section 12 below.

11. Ordering of abstractions

We have called one abstraction "stronger" than another without precisely defining how we might order abstractions. Being able to order abstractions helps to describe the complexity of the abstract spaces. It also offers a solution to the problem of inconsistent abstract spaces identified in the last section. The definitions of T* and NT* abstractions suggest two obvious ways of ordering abstractions: T* abstractions can be ordered by the number of theorems in the abstract systems, and NT* abstractions by the number of nontheorems. For syntactically complete systems, these two orders are, of course, related.

To construct an order, the abstractions must map from the same ground space. However, they can map to completely different abstract spaces. For T* abstractions, we use the following order:

Definition 11.1 ($\sqsubseteq$). If $f_i : \Sigma_1 \Rightarrow \Sigma_2$ and $f_j : \Sigma_1 \Rightarrow \Sigma_3$ are two abstractions, then $f_i \sqsubseteq f_j$ iff for all wffs φ, if $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$ then $f_j(\varphi) \in \mathrm{TH}(\Sigma_3)$. We also say that f_j is stronger than f_i, or that f_i is weaker than f_j.

f_j is "stronger" than f_i in the sense that there are more wffs, α, in $\mathrm{TH}(\Sigma_1)$ such that $f_j(\alpha) \in \mathrm{TH}(\Sigma_3)$ than wffs, β, in $\mathrm{TH}(\Sigma_1)$ such that $f_i(\beta) \in \mathrm{TH}(\Sigma_2)$. $\sqsubseteq$ will be used to order T* abstractions. Note that $f_i \sqsubseteq f_j$ does not imply that the abstract language of f_i is a subset of the abstract language of f_j.

For NT* abstractions we will use a different order:

Definition 11.2 ($\preceq$). If $f_i : \Sigma_1 \Rightarrow \Sigma_2$ and $f_j : \Sigma_1 \Rightarrow \Sigma_3$ are two abstractions, then $f_i \preceq f_j$ iff for all wffs φ if $f_i(\varphi) \in \mathrm{NTH}(\Sigma_2)$ then $f_j(\varphi) \in \mathrm{NTH}(\Sigma_3)$. We also say that f_j is stronger with respect to NTH than f_i, or that f_i is weaker with respect to NTH than f_j.

$\preceq$ is the analogous order to $\sqsubseteq$ for NT* abstractions. Its properties are entirely dual to those of $\sqsubseteq$. Indeed, when the systems involved are syntactically complete we have $f_j \preceq f_i$ iff $f_i \sqsubseteq f_j$ (by Theorem 6.3). The remainder of this section therefore concentrates on $\sqsubseteq$ with the observation that everything holds dually for $\preceq$. We first introduce the symbol for equivalence.

Definition 11.3 ($\equiv$). $f_i \equiv f_j$ iff $f_i \sqsubseteq f_j$ and $f_j \sqsubseteq f_i$.

If $f_i \equiv f_j$, we say that f_i is equivalent to f_j. $\equiv$ is in fact an equivalence relation and satisfies the usual properties of equivalence relations (transitivity, symmetry, and reflexivity). $\sqsubseteq$ is a preorder, in fact:

Lemma 11.4 (Preorder). $\sqsubseteq$ *is a preorder. That is, it is*

- *transitive: if* $f_i \sqsubseteq f_j$ *and* $f_j \sqsubseteq f_k$, *then* $f_i \sqsubseteq f_k$;
- *reflexive:* $f_i \sqsubseteq f_i$.

$\sqsubseteq$ is not a partial order as two equivalent abstractions are in general not equal as they have different abstract spaces and f_A's (this makes antisymmetry fail). Starting from $\sqsubseteq$ we can define, in the obvious way, the partial order $\sqsubseteq_=$ over the equivalence classes of abstractions.

One very important property of this ordering on abstractions is that if we can order two abstractions, the provability preserving properties of one abstraction are also possessed by the other.

Theorem 11.5. *If* $f_i : \Sigma_1 \Rightarrow \Sigma_2$ *is a TI abstraction (TD) and* $f_j : \Sigma_1 \Rightarrow \Sigma_3$ *is an abstraction such that* $f_i \sqsubseteq f_j$ $(f_j \sqsubseteq f_i)$, *then* f_j *is TI (TD).*

Proof. We only give the proof for TI abstractions; the proof for TD abstractions is entirely dual. If f_i is TI, then $\varphi \in \mathrm{TH}(\Sigma_1)$ implies $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$. As $f_i \sqsubseteq f_j$, $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$ implies $f_j(\varphi) \in \mathrm{TH}(\Sigma_3)$. Thus $\varphi \in \mathrm{TH}(\Sigma_1)$ implies $f_j(\varphi) \in \mathrm{TH}(\Sigma_3)$. That is, f_j is TI. □

Another useful property is that any TI abstraction is stronger than any TD abstraction.

Theorem 11.6. *If* $f_i : \Sigma_1 \Rightarrow \Sigma_2$ *is a TI abstraction and* $f_j : \Sigma_1 \Rightarrow \Sigma_3$ *is a TD abstraction, then* $f_j \sqsubseteq f_i$.

Proof. Since f_j is TD, $f_j(\varphi) \in \mathrm{TH}(\Sigma_3)$ implies $\varphi \in \mathrm{TH}(\Sigma_1)$. But, as f_i is TI, $\varphi \in \mathrm{TH}(\Sigma_1)$ implies $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$. Thus, $f_j(\varphi) \in \mathrm{TH}(\Sigma_3)$ implies $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$. That is, $f_j \sqsubseteq f_i$. □

A simple corollary to this last theorem is that $f_j \sqsubseteq f_I \sqsubseteq f_i$ where f_I is an identity abstraction, f_i is any TI abstraction, and f_j any TD abstraction. Another consequence is that $f_i \equiv f_I$ for any TC abstraction. Thus $\sqsubseteq$ generates orders with chains of TI abstractions on the right, all the TC abstractions in the middle, and chains of TD abstractions on the left. Given a set of ordered abstractions, if one of the abstractions is TI then all the stronger abstractions are also TI, and if one of the abstractions is TD than all the weaker abstractions are also TD.

Composition is a very natural way to build such ordered sets of T* abstractions as the following result holds.

Theorem 11.7. *If* $f_i : \Sigma_1 \Rightarrow \Sigma_2$ *is an abstraction and* $f_j : \Sigma_2 \Rightarrow \Sigma_3$ *is a TI abstraction (TD abstraction), then* $f_i \sqsubseteq f_j \circ f_i$ $(f_j \circ f_i \sqsubseteq f_i)$.

Proof. As usual the proof for TD abstractions is dual to the proof for TI abstractions. If $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$ and f_j is TI, then $f_j(f_i(\varphi)) \in \mathrm{TH}(\Sigma_3)$. Thus $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$ implies $f_j \circ f_i(\varphi) \in \mathrm{TH}(\Sigma_3)$. That is, $f_i \sqsubseteq f_j \circ f_i$. □

Notice that in Theorem 11.7 we have made no hypotheses about f_i which could be TI, TD, TC, or none of these. On the other hand, we can use Theorem 11.7 to construct ordered sets of TI abstractions out of a basic set of TI abstractions. For example, we might reduce the computational complexity of a planning domain by composing an ABSTRIPS-like abstraction, f_{AB}, with a propositional abstraction, f_{prop}. It quickly follows that,

$$f_I \sqsubseteq f_{\mathrm{AB}} \sqsubseteq f_{\mathrm{prop}} \circ f_{\mathrm{AB}}.$$

In Section 9 we have shown that TI* abstractions can map a consistent theory onto an inconsistent theory. A very useful property of $\sqsubseteq$ is that a totally ordered set of abstractions has all those with consistent abstract theories on the left and those with inconsistent abstract theories on the right; more precisely, if an abstract theory is consistent then all weaker abstractions map onto consistent abstract theories, while if an abstract theory is inconsistent then all stronger abstractions map onto inconsistent abstract theories.

Theorem 11.8. *If $f_i : \Sigma_1 \Rightarrow \Sigma_2$, and $f_j : \Sigma_1 \Rightarrow \Sigma_3$ are two abstractions such that $f_j \sqsubseteq f_i$, then if Σ_2 is consistent, Σ_3 is also. Alternatively, if Σ_3 is inconsistent then Σ_2 is also.*

Proof. As $f_j \sqsubseteq f_i$, then $f_j(\varphi) \in \mathrm{TH}(\Sigma_3)$ implies $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$. Assume that Σ_2 is consistent but that Σ_3 is inconsistent. There will exist a wff, ψ, for which $f_i(\psi) \notin \mathrm{TH}(\Sigma_2)$. But $f_j(\psi) \in \mathrm{TH}(\Sigma_3)$ as all wffs are provable in an inconsistent theory. Thus, $f_i(\psi) \notin \mathrm{TH}(\Sigma_2)$ and $f_j(\psi) \in \mathrm{TH}(\Sigma_3)$. But this contradicts $f_j(\varphi) \in \mathrm{TH}(\Sigma_3)$ implying $f_i(\varphi) \in \mathrm{TH}(\Sigma_2)$ for all φ. Hence, if Σ_2 is consistent then Σ_3 cannot be inconsistent. And if Σ_3 is inconsistent then Σ_2 cannot be consistent. □

This result will be used, in Section 12, to propose a solution to the problem of inconsistent abstract spaces discussed in Section 9.

12. Hierarchies of abstraction spaces

So far we have concentrated on applying abstraction only once. However, the process of abstraction can be iterated to generate hierarchies of abstract spaces. Almost all the work done in the past with `abstractions` used this technique as well as that on approximations (which uses also TD abstractions

as a means to underestimating the solution). The goal of this section is to discuss the use of hierarchies on the basis of the results presented before in this paper. We concentrate on TI abstractions but everything is generalizable to NT* and TD* abstractions.

By *iterating* the process of abstracting we mean repeatedly:

- picking an abstraction;
- explicitly generating the abstract space;
- using (or abstracting further) the abstract space.

By *composing* abstractions we mean abstraction composition (in the mathematical sense of function composition) as defined in Section 10. Notice that, while the iteration of abstractions implies the generation of all the intermediate spaces, this is not the case with abstraction composition.

At each step, both in abstraction iteration and in abstraction composition, a different abstraction can be applied, as long as it is TI. Thus, for instance, we might first collapse predicates (as Tenenberg proposes), then collapse constants (as Hobbs' granularity theory suggests), then apply an ABSTRIPS abstraction or granularity again, and so on. Which, among all the possible sequences of abstractions, is the best to use is very much dependent on the problem.

If we choose to perform abstraction iteration by composing abstractions, the abstract spaces can be constructed starting directly from the most abstract and then going back towards the ground space; this avoids us going forwards through all the intermediate abstract spaces. Having decided which set of abstractions to compose, it is sufficient to build the composite mapping function and with this the most abstract space. Note that this trick can also be used to generate all the intermediate abstract spaces back to the ground space.

The question which arises is then as follows. If we do not put a bound on the number of abstractions applied in the chain, can we get to an abstract space Σ_i such that abstracting Σ_i generates Σ_i itself or, more generally, the same number of theorems as Σ_i? That is, can we get to a fixed point? A related question is "if we get to such a situation, are we actually able to recognize it?". In general there are many sequences of abstractions. Are the fixed points computed by each sequence the same or different?

There are two possible situations.

In the first case, Σ_i is inconsistent. In fact, as we increase the strength of the abstraction, we monotonically increase the number of theorems; we eventually reach an upper bound when the set of theorems coincides with the language. Notice that it may not be possible to recognize that we are at the fixed point due to undecidability. Thus, in trying to generate simpler and simpler abstract spaces, it is actually possible to generate indefinitely long chains of abstractions. Putting an upper bound on the depth of the

chain is one solution to this problem. A better solution perhaps is to require that there is a point after which all the Σ_i are decidable (an easy way to generate a decidable abstract space Σ_i is to use an propositional abstraction, that is an abstraction whose abstract space is propositional).

In the second case, Σ_i is consistent but abstracting it further does not increase the number of theorems. This happens when the abstraction tries to forget details which have already been forgotten. This might be the case with most `abstractions` which, at each step, decrease the complexity of the search space. Possible situations are: trying to delete preconditions which do not occur, applying a propositional abstraction to a propositional ground space, or collapsing predicates already collapsed. In this situation, the fixed point is different for each different abstraction. We are, however, always able to recognize when we are at such a fixed point.

Abstraction iteration can be used to tackle the problem of inconsistent abstract spaces (see Section 9). Because abstract spaces are usually undecidable, no general methods for building consistent abstract spaces independent of the (consistent) ground space can exist. In Section 9 we have argued that an inconsistent abstract space should be avoided. One way around the problem of inconsistent abstract spaces is to place strong restrictions on the types of abstraction and/or axioms allowed. For example, we might only work with TD* abstractions but this loses completeness. Or we might fix in advance the formal system or the class of formal systems so that the abstraction is guaranteed to construct a consistent abstract space; for instance, we might place a syntactic restriction on the axioms allowed in the ground space.

Many attempts in this direction have been tried. We will briefly repeat some observations from the examples reported in Section 8. Tenenberg [73,74] presents two solutions to this problem for predicate abstractions. The first keeps in the abstract space only those axioms from the ground space that do not distinguish between the predicates which are mapped together. Unfortunately deciding indistinguishability is an undecidable property requiring an arbitrary amount of theorem proving in the ground space. Additionally, the abstractions in this class are not TI as the abstract space has fewer axioms than the ground space (since we don't map all of them) and therefore fewer theorems. Tenenberg's second solution [74] overcomes the objection to undecidability. In this proposal, axioms are kept in the abstract space provided they can be trivially shown not to distinguish in the ground space between predicates which are conflated together; thus, instead of performing an arbitrary amount of theorem proving in the ground space to determine indistinguishability, we insist that it is an immediate consequence of the axioms. While being decidable, this solution is again not TI. Hobbs [31] and Imielinski [32] also suggest solutions to the false proof problem for domain abstractions in which the objects in the language (and

not the predicate symbols) are mapped together. Hobbs' proposal, however, requires arbitrary theorem proving in the ground theory, while Imielinski's proposed solution is not TI.

One solution (discussed in detail in [21]), suggested by Theorem 11.8 in Section 11, is to use an ordered chain of TI* abstractions, the strongest of which gives a decidable abstract space. If this abstract space is consistent, then all the intermediate abstract spaces back to the ground space will be also. Thus we can iterate back through the chain of abstractions safe in the knowledge that all the intermediate (and possibly undecidable) abstract spaces are consistent. Of course, we can't escape undecidability so this trick is inevitably cautious; for instance there will be cases where the strongest abstract space is inconsistent but the intermediate abstract spaces are consistent.

13. Building abstractions

We have argued at length that abstractions should preserve provability. Unfortunately, it is difficult to predict in advance how an abstraction will affect the "global" property of provability.

The goal of this section is to define some "local" properties of an abstraction we can test, that is a set of restrictions on f_A and on the relations between axioms and deductive machinery in the ground and abstract spaces, which guarantee that it is TI*. One step in this direction was done by Plaisted, who provided a set of properties, built inside his definition of ordinary and weak abstractions, for NTI abstractions between resolution systems. However his definitions are still based on the preservation of provability and not on the more primitive notions of f_A, Σ_1, and Σ_2. This means that, given a definition of abstraction we still have to prove that it satisfies these properties. (Plaisted, in fact, lists, as examples, a set of abstractions which satisfy those properties [60].)

A first result is that all Σ-invariant abstractions are TI. (A Σ-invariant abstraction is an abstraction in which the language, axioms, and inference rules are all abstracted using the same mapping function.)

Theorem 13.1. *If $f : \Sigma_1 \Rightarrow \Sigma_2$ is a Σ-invariant abstraction, then it is a TI abstraction.*

The proof follows trivially from the consideration that we can map a proof tree Π_1 of φ onto a proof tree Π_2 of $f(\varphi)$ merely by applying f to every wff in the proof tree. Notice that Theorem 13.1 holds even if the abstraction is Δ-variant, that is, even if the deductive machinery in the abstract space is different from that in the ground (as in GPS).

This is, of course, not the only way to build TI abstractions. Indeed, there are TI abstractions that are not Σ-invariant. In most cases, we inherit a fixed language and inference engine for the abstract and ground theories. We cannot therefore use Σ-invariant abstractions which change the inference rules. Instead we want to find Δ-invariant abstractions as this saves us implementing a new inference engine for the abstract theory, and allows us to use hierarchies of abstractions. Plaisted proves [61, Theorem 2.1] some local properties that are sufficient but not necessary to make such mappings between resolution systems NTI abstractions. We can generalize this result to find conditions on a Δ-invariant abstraction between first-order languages with complete deductive machinery that make the abstraction TI/NTI.

As we noted before, most useful `abstractions` change the theory not the logic. That is, they map the predicates but not the logical structure of the wff. In general, the logic is well-behaved and it is the theory that needs to be simplified. There is another good reason for using theory abstractions; if the abstraction is to be TI, then it needs to preserve the meaning of the connective introduction and elimination rules. For example, since we can derive $p \wedge q$ from p and q, we need to be able to derive $f(p \wedge q)$ from $f(p)$ and $f(q)$. A theory abstraction will guarantee that deductions using the connective rules remain valid deductions in the abstract theory. The majority of useful `abstractions` are also Λ/Ω-invariant abstractions. For an abstraction to be TI, the abstraction of the axioms of the ground theory must also be theorems of the abstract theory. This is easily achieved by making the abstraction Λ/Ω-invariant.

We have thus reduced the problem of constructing a Δ-invariant abstraction to the much easier problem of deciding on a suitable mapping of atomic formulas for a Λ/Ω-invariant theory abstraction. The question now becomes: can we come up with any syntactic test on this mapping which guarantees that the abstraction is TI? Given the undecidability of provability, it is impossible for us to find a test that captures the whole class of TI abstractions. However, it is possible to come up with a test that captures a very large subclass. Indeed, the test captures most of the `abstractions` listed in Section 8.

Theorem 13.2. *If $f : \Sigma_1 \Rightarrow \Sigma_2$ is a Λ/Ω-invariant theory abstraction such that its mapping function, f_Λ, preserves the names of the occurrences of the free and bound variables which occur in the abstract space and preserves substitution instances, that is, $f_\Lambda(p[a]) = f_\Lambda(p[x])\{f_\Lambda(a)/x\}$, then f is a TI* abstraction.*

Proof (*Outline*). (Remember that we are restricting ourselves to first-order systems). Since f is a theory abstraction it is negation preserving. Thus it is sufficient to prove that f is a TI abstraction. We show how, given a

proof tree Π_1 of φ in Σ_1, we can construct a proof tree Π_2 of $f(\varphi)$ in Σ_2. The argument proceeds by induction on the length of the proof tree Π_1. If the proof tree is of length 1, then φ must be an axiom. But, as f is Λ/Ω-invariant, $f(\varphi)$ is also an axiom. We then assume that we can show it for all proof trees up to length N, and prove it is true of all proof trees Π_1' of length $N+1$. We consider the last inference in the proof tree Π_1'. If it is a connective introduction or elimination, a universal elimination or an existential introduction, then we apply the same rule at the bottom of the proof tree we construct in Σ_2. For a reductio ad absurdum, the preservation of substitution instances guarantees $f(\bot) = \bot$. We can therefore apply an application of reductio ad absurdum at the bottom of Σ_2. If it is a universal introduction or an existential elimination, then the same rule can also be applied in Σ_2 since the conditions on applying the rule (that the assumptions do not mention the free variable being universally quantified, etc.) still hold as variable names are preserved. □

Notice that we can drop variables in the abstract space, for instance by applying a propositional abstraction (this is why the theorem is restricted to the variables which occur in the abstract space). Actually, we can drop the requirement on preserving the name of bound variables provided we are carefully to avoid any naming clashes. If we restrict ourselves to abstraction mappings between resolution systems, Theorem 13.2 is similar to [61, Theorem 2.1]; for Plaisted's abstractions, our extra condition on the name of variables is redundant since all wffs are already skolemized and no variable naming problems can arise. Note that preserving variable names and substitution instances does not capture all TI theory abstractions. For example, the theory abstraction that maps $p(a)$ onto $\top$ and $p(x)$ onto $p(x)$ is TI if $p(a)$ is a theorem of the ground space; however, this abstraction does not preserve substitution instances.

Thus we have found a useful syntactic condition on the mapping that guarantees that the resulting abstraction be TI. The question now becomes: what sort of mappings preserve substitution instances and preserve (or drop) variables? Historically, most abstractions fall into four main types (see Sections 8.1–8.4):

- *predicate abstractions* where we map the predicate names in some uniform way;
- *domain abstractions* where we map the constants or function symbols in some uniform way;
- *propositional abstractions* where we drop some or all of the arguments to predicates;
- *ABSTRIPS abstractions* where we map some of the preconditions onto $\top$ (the condition on preserving substitution is vacuously satisfied).

All the abstractions in these four classes satisfy the hypotheses of Theorem 13.2.

Taking a closer look at the classes of `abstractions` defined above it is easy to notice that all of them work on atomic formulas and that each abstracts different parts. This observation can be exploited to make a systematic classification of abstractions which satisfy the hypotheses of Theorem 13.2. This characterization can be given following the recursive definition of atomic formulas. Thus theory abstractions that preserve substitution instances can be characterized as follows:

(1) *term* abstractions which map the terms, themselves classified into:
   - *constant symbol* abstractions which map constants,
   - *function symbol* abstractions which map function symbols;

(2) *predicate symbol* abstractions which map predicate symbols.

Notice that this last classification is exhaustive in the sense that it considers *all and only* the possible ways to build TI abstractions by manipulating the parts of the atomic wffs.

The only examples of constant symbol abstractions are domain abstractions. In fact the only syntactic information that a constant carries is its name, which is thus the only thing which can be forgotten. To forget details of a constant necessarily means to collapse it with another one (to introduce a new name is useless). We call this *collapsing*.

With function symbol abstractions, it is possible to collapse the function name, or to change the arity. Thus the arity can be decreased by any number of arguments (in the limit, functions become constants), or the order of the arguments can be changed (increasing the number of arguments is not considered here since it corresponds to an increase of complexity). We call this *argument manipulation*.

With predicate symbol abstractions it is possible to collapse the predicate name (i.e. predicate abstractions) and argument manipulation (e.g. propositional abstractions). The one type of mapping which distinguishes function symbol abstractions from predicate symbol abstractions is that predicate symbols can be (selectively) mapped, after having deleted all the arguments, onto the special symbols $\top$ or $\bot$. We call this a *truth* mapping. ABSTRIPS is an example of a truth mapping.

Finally we note that Theorems 13.1 and 13.2 provide us with a way to build abstractions which are TI irrespective of the particular axioms of the ground space. Theorems 13.1 and 13.2, in fact, do not place any restrictions on the axioms of the ground space, *all Σ-invariant abstractions, and Δ-invariant abstractions which preserve substitution instances are TI irrespective of the axioms of the ground space*. Notice that it is not sufficient that we abstract the axioms in the same way as the wffs since not all Λ/Ω-invariant abstractions are TI. For example, let Σ_1 and Σ_2 be complete

propositional theories. If $f(p) = \top$, $f(q) = p \wedge q$, $f(\varphi) = \varphi$ otherwise, and $\Omega_1 = \{p, q, r\}$, then it is TI. However, if $\Omega_1 = \{p, r\}$, then f is not TI (since we cannot show $p \rightarrow r \in \mathrm{TH}(\Sigma_2)$).

Once we have built up a collection of abstractions, we can use the various operations on abstractions to construct yet more abstractions. In particular, we can use the fact that the composition of two T* (NT*) abstractions is itself a T* (NT*) abstraction, and that the inverse of a TI* abstraction is a TD* abstraction (and vice versa). Finally, to construct NT* abstractions, we can call upon the fact that a T* abstraction that is negation preserving is also an NT* abstraction, and vice versa to construct T* abstractions.

14. Summary and conclusions

We have presented the beginnings of a *theory* of abstraction. Abstraction is defined as a mapping between formal systems. We consider such mappings in the light of how they preserve provability and, for refutation systems, inconsistency. This framework is very general. Indeed it captures a lot of the work on abstraction done in the past (see the references in Section 1) and, as a particular case, `abstraction`. Abstractions often satisfy much stronger requirements than just the preservation of provability; for example, there is frequently a correspondence between the structure of proof in the abstract formal system and that in the ground. However, even restricting ourselves to the weak property of preserving provability, we are able to prove some very interesting results.

We have used this theory of abstraction to capture and generalize previous work on abstraction and, in particular, because of our interests, on `abstraction`. This has allowed us to formalize work often informally described in an uniform way, and to classify the different types of `abstractions`. Many, at first sight, seemingly different abstractions are in fact related; for example, we have shown that Hobbs' granularity [31] is an example of one of Plaisted's abstractions [61].

The other main use of this theory of abstraction has been to study the formal properties of abstractions and the operations like composition and ordering which can be defined upon them. In particular, we have investigated the properties of the four main classes of abstraction (TI, TD, NTI, NTD), and the relationships that exist between them. We have also considered the different ways to use abstractions. Finally we have presented some results which tackle the problem of how to build "useful" abstractions.

Three individual results are worth highlighting. Firstly, preserving provability seems a good way to characterize abstractions (and in particular `abstractions`). Indeed, with very few exceptions, abstractions fall into one of two classes, those in which proof (inconsistency) in the abstract space

implies proof (inconsistency) in the ground, and those in which the opposite holds. This has immediate implications on the different ways we can use abstractions (that is, deductively or abductively). Secondly, the problem of inconsistent abstract spaces, where a consistent space maps onto an inconsistent abstract space, is inevitable for almost all abstractions. We have proposed a solution which exploits the fact that abstractions can ordered. Thirdly, there are very few choices to be made in building abstractions. Since in general it is the theory not the logic that is introducing complexity into the problem solving, one needs merely to decide how to map the atomic formulas. For an abstraction to preserve provability, it is sufficient that the mapping of atomic formulas preserve substitution instances. And the sorts of mappings that preserve substitution instances (that is, collapsing, argument manipulation, and truth mapping abstractions) capture the four main types of abstractions (that is, predicate, domain, propositional, and ABSTRIPS abstractions) identified in Section 8.

15. Further work

This paper describes the basic underlying theory we are currently using in our work on abstraction. At the moment we are working on the following topics:

- Firstly, we want to refine our formal definition of abstraction to capture more properties of abstractions. This result is a direct consequence of the development of a theory of mapping an abstract proof back into the ground space. This requires the definition of the class of abstractions which preserve (as well as provability) the structure of the proof between the ground and the abstract spaces. Some preliminary results can already be found in [24].
- Secondly we need to study what is meant by the abstract representation being "simpler to handle". Ultimately this will involve a complexity analysis of the process of solving an abstract problem and using the abstract solution to aid the proof in the ground space. Some preliminary results can already be found in [25].
- Thirdly, we want to define and implement a theorem prover for using abstraction. As hinted in Section 1, we want to implement abstraction inside an abstract proof checker, that is, inside a system which allows us to use abstraction interactively. The abstract proof checker is under development (some preliminary ideas can be found in [26]) and it is being implemented on top of GETFOL [20], an interactive theorem prover which runs on top of a re-implementation of the FOL system [28,79]. Some preliminary testing has been made; for instance we have proved

with GETFOL a (very) simplified version of Gödel's theorem. The results are encouraging.

- Fourthly, if such a theorem prover is going to be used on real problems, it will have to construct automatically (or, possibly, suggest in the case of interactive theorem proving) abstractions. We have done some work in the case of ABSTRIPS abstractions [7,8].

There are also some areas where we plan on developing our work. In particular, we are interested in broadening our definition of abstraction to look at analogy. Abstraction is closely related to analogy and we would like to study (and implement) them in a uniform framework. Finally, we would like to combine abstraction with other theorem proving techniques like proof plans [9]; a successful synthesis of such techniques would help us towards the dream of creating artificial (mathematical) reasoners.

Acknowledgement

The idea of describing abstraction as a mapping between formal systems originated when the first author was working inside the FOL group, at the CS Department of Stanford University. This work started when the first author was at the AI Department of Edinburgh University. In Edinburgh, financial support for the first author was provided by SERC grant GR/E/4459.8. Currently the first author's research at IRST is funded by ITC (Istituto Trentino di Cultura). The second author is supported by a SERC postdoctoral fellowship.

The research described in this paper owes a lot to the openness and sharing of ideas which exists in the Mathematical Reasoning Group in Edinburgh and Mechanized Reasoning Group in Trento. The authors thank Alan Bundy, Alessandro Cimatti, Craig Knoblock, Paolo Pecchiari, Luciano Serafini, Alex Simpson, Carolyn Talcott, and Adolfo Villafiorita for their careful reading of earlier versions of this paper. Gregori Mints and Alan Smaill have helped in the solution of two technical problems. The referee has provided very detailed and useful feedback which has allowed us to improve substantially the quality of the paper.

References

[1] S. Abramsky and C. Hankin, An introduction to abstract interpretation, in: *Abstract Interpretation of Declarative Languages* (Ellis Horwood, Chicester, England, 1987).

[2] W. Bledsoe, Using examples to generate instantiations of set variables, in: *Proceedings IJCAI-83*, Karlsruhe, Germany (1983).

[3] W. Bledsoe, A precondition prover for analogy, CS Department Memo, University of Texas at Austin, TX (1990).

[4] W. Bledsoe and M. Tyson, The UT interactive prover, Tech. Rept. ATP-17, Mathematics Department, University of Texas at Austin, TX (1975).
[5] B. Brock, S. Cooper and W. Pierce, Analogical reasoning and proof discovery, in: *Proceedings CADE-9* Argonne, IL (1988) 451–468.
[6] A. Bundy, *The Computer Modelling of Mathematical Reasoning* (Academic Press, NY, 1983).
[7] A. Bundy, F. Giunchiglia and T. Walsh, Building abstractions, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990); also: DAI Research Paper No. 506, University of Edinburgh (1990); also Tech. Rept. 9007-02, IRST, Trento, Italy (1990).
[8] A. Bundy, F. Giunchiglia and T. Walsh, Calculating criticalities, Tech. Rept., DAI University of Edinburgh (1991); also: Tech. Rept. 9112-23, IRST. Trento, Italy (1991); also: *Artif. Intell.* (submitted).
[9] A. Bundy, D. Sannella, F. Giunchiglia, F. van Harmelen, J. Hesketh, P. Madden, A. Smaill, A. Stevens and L. Wallen, Proving properties of logic programs: a progress report, in: *1988 Alvey Conference* (1988) 131–133; also: DAI Research Paper No. 361, Department Artificial Intelligence, University of Edinburgh (1988).
[10] C. Chang, Resolution plans in theorem proving, in: *Proceedings IJCAI-79*, Tokyo (1979) 143–148.
[11] C. Chang and J. Slagle, Using rewriting rules for connection graphs to prove theorems, in: B.L. Webber and N.J. Nilsson, eds., *Readings in Artifical Intelligence* (Morgan Kaufmann, San Mateo, CA, 1981) 109–118.
[12] R. Cremonini, K. Marriott and H. Sondergaard, A framework for abstraction based on abstract interpretation, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990) 233–244.
[13] R. Doyle, Constructing and refining causal explanations from an inconsistent domain theory, in: *Proceedings AAAI-86*, Philadelphia, PA (1986).
[14] B. Dreben and W. Goldfarb, *The Decision problem—Solvable Classes of Quantificational Formulas* (Addison-Wesley, Reading, MA, 1979).
[15] T. Ellman, Mechanical generation of heuristics through approximation of intractable theories, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990) .
[16] H. Gelernter, Realization of a geometry theorem-proving machine, in: *Proc. IFIP Congress* (1959) 273–282.
[17] A. Giordana and L. Saitta, Abstraction: a general framework for learning, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990).
[18] E. Giunchiglia, A set of hierarchically structured decision procedures for some subclasses of First Order Logic, in: *Proceedings 3rd Scandinavian Conference on Artificial Intelligence* Roskilde University, Denmark (1991); also: MRG-DIST Tech. Rept. 9101-01, University of Genova, Italy (1991).
[19] F. Giunchiglia and E. Giunchiglia, Building complex derived inference rules: a decider for the class of prenex universal-existential formulas, in: *Proceedings 7th European Conference on Artificial Intelligence*, Munich, Germany (1988); extended version: DAI Research Paper 359, Department of Artificial Intelligence, University of Edinburgh (1980).
[20] F. Giunchiglia and P. Traverso, GETFOL user manual—GETFOL version 1, DIST Tech. Rept. 9107-01, University of Genova, Italy (1991).
[21] F. Giunchiglia and T. Walsh, Abstract theorem proving, in: *Proceedings IJCAI-89*, Detroit, MI (1989); also: Tech. Rept. 8902-03, IRST, Trento, Italy (1989); also: DAI Research Paper No. 430, University of Edinburgh (1989).
[22] F. Giunchiglia and T. Walsh, Abstracting into inconsistent spaces (or the false proof problem), in: *Proceedings AI*IA 89* (1989); also: Tech. Rept. 8904-02, IRST, Trento, Italy (1989); also: DAI Research Paper No. 514, University of Edinburgh (1989).
[23] F. Giunchiglia and T. Walsh, Theorem proving with definitions, in: *Proceedings 7th Conference of the Society for the Study of Artificial Intelligence and Simulation of*

Behaviour (1989); also: Tech. Rept. 8901-03, IRST, Trento, Italy (1989); also DAI Research Paper No. 429, Department of Artificial Intelligence, University of Edinburgh (1989).

[24] F. Giunchiglia and T. Walsh, Abstract theorem proving: mapping back, Tech. Rept. 8911-16 IRST Trento, Italy (1989); also: DAI Research Paper No. 460a, University of Edinburgh (1989).

[25] F. Giunchiglia and T. Walsh, Using abstraction, in: *Proceedings 8th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, Leeds, England (1991); also: Tech. Rept. 9010-08, IRST, Trento, Italy (1990); also DAI Research Paper No. 515, University of Edinburgh (1990).

[26] F. Giunchiglia and T. Walsh, An abstract proof checker, in: *Proceedings 2nd International Conference on Artificial Intelligence and Mathematics*, Fort Lauderdale, FL (1992).

[27] F. Giunchiglia and T. Walsh, The inevitability of inconsistent abstract spaces, Tech. Rept. 9006-16, IRST, Trento, Italy (1990); also: DAI Research Paper, University of Edinburgh (1990); also: *J. Autom. Reasoning* (to appear).

[28] F. Giunchiglia and R. Weyhrauch, FOL user manual—FOL version 2, Tech. Rept. 9107-02, DIST, University of Genova, Genova, Italy (1991).

[29] C. Green, Application of theorem proving to problem solving, in: *Proceedings IJCAI-69*, Washington, DC (1969) 219–239.

[30] J. Herbrand, Investigations in proof theory: the properties of true propositions, Ph.D. Thesis (1930), in: J. Heijenoort, ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931* (Harvard University Press, Cambridge, MA, 1967).

[31] J. Hobbs, Granularity, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985).

[32] T. Imielinski, Domain abstraction and limited reasoning, in: *Proceedings IJCAI-87*, Milan, Italy (1987) 997–1003.

[33] B. Indurkhya, Approximate semantic transference: a computational theory of metaphors and analogies, *Cogn. Sci.* **11** (1987) 445–480.

[34] J.N. Johnson-Laird, *Mental Models* (Harvard University Press, Cambridge, MA, 1983).

[35] L. Joskowicz, Approximation and abstraction in spatial reasoning: a case study, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990) 134–140.

[36] M. Keane, Analogical mechanisms, *Artif. Intell. Rev.* **2** (1988) 229–250.

[37] S.C. Kleene, *Introduction to Metamathematics* (North-Holland, Amsterdam, 1952).

[38] C.A. Knoblock, Learning hierarchies of abstraction spaces, in: *Proceedings Sixth International Workshop on Machine Learning*, Ithaca, NY (1989).

[39] C.A. Knoblock, A theory of abstraction for hierarchical planning, in: P. Benjamin, ed., *Proceedings of the Workshop on Change of Representation and Inductive Bias* (Kluwer, Boston, MA, 1989).

[40] C.A. Knoblock, Learning abstraction hierarchies for problem solving, in: *Proceedings AAAI-90*, Boston, MA (1990).

[41] C.A. Knoblock, Search reduction in hierarchical problem solving, in: *Proceedings AAAI-91*, Anaheim, CA (1991).

[42] C.A. Knoblock, S. Minton and O. Etzioni, Integrating abstraction and explanation-based learning in PRODIGY, in: *Proceedings AAAI-91*, Anaheim, CA (1991).

[43] C.A. Knoblock, Abstracting the tower of hanoi, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990) 13–23.

[44] C.A. Knoblock, J. Tenenberg and Q. Yang, Characterizing abstraction hierarchies for planning, in: *Proceedings AAAI-91*, Anaheim, CA (1991).

[45] R. Korf, Planning as search: a quantitative approach, *Artif. Intell.* **33** (1987) 65–88.

[46] R. Kowalski, A proof procedure using connection graphs, *J. ACM* **22** (4) (1975) 227–260.

[47] H.R. Lewis, *Unsolvable Classes of Quantificational Formulas* (Addison-Wesley, Reading, MA, 1979).

[48] J. McCarthy and P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer and D. Michie, eds., *Machine Intelligence* **4** (Edinburgh University Press, Edinburgh, Scotland, 1969) 463–502.

[49] T. Melham, Abstraction mechanisms for hardware verification, Tech. Rept. 106, University of Cambridge, Computer Laboratory (1987).
[50] C. Mellish, Abstract interpretation of PROLOG programs, in: *Abstract Interpretation of Declarative Languages* (Ellis Horwood, Chicester, England, 1987) 181–198.
[51] E. Mendelson, *Introduction to Mathematical Logic* (Van Nostrand Reinhold, New York, 1964).
[52] A. Modi, D. Steier and A. Westerberg, Learning to use approximations and abstractions in the design of chemical processes, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990) 53–63.
[53] I. Mozetic, Abstractions in model-based diagnosis, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990) 64–75.
[54] I. Mozetic, Reduction of diagnostic complexity through model abstractions, in: *Proceedings 1990 International Workshop on Principles of Diagnoses*, Stanford, CA (1990).
[55] I. Mozetic and C. Holzbaur, Extending EBG by abstraction operators, in: *Proceedings EWSL-91*, Porto, Portugal (1991).
[56] J. Munyer, *Analogy as a means of discovery in problem solving and learning*, Ph.D. Thesis, University of California, Santa Cruz, CA (1991).
[57] A. Newell and H. Simon, *Human Problem Solving* (Prentice-Hall, Englewood Cliffs, NJ, 1972).
[58] N.J. Nilsson, Logic and artificial intelligence, *Artif. Intell.* **47** (1991) 31–56.
[59] P. Pecchiari, Meccanizzazione del concetto di modello di un dimostratore interattivo, IRST, Trento, Italy, Thesis 9009-15 (1990).
[60] D. Plaisted, Abstraction mappings in mechanical theorem proving, in: *Proceedings Fifth Conference on Automated Deduction*, Les Arcs, France (1980) 264–280.
[61] D. Plaisted, Theorem proving with abstraction, *Artif. Intell.* **16** (1981) 47–108.
[62] D. Plaisted, Abstraction using generalization functions, in: *Proceedings 8th Conference on Automated Deduction* (1986) 365–376.
[63] D. Plaisted, Mechanical theorem proving, in: R.B. Banerji, ed., *Formal Techniques in Artificial Intelligence: A Sourcebook* (Elsevier Science Publishers, Amsterdam, 1990).
[64] D. Plummer, Gazing: controlling the use of rewrite rules, Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh (1987).
[65] G. Polya, *How to Solve It* (Princeton University Press, Princeton, NJ, 1945).
[66] D. Prawitz, *Natural Deduction–A Proof Theoretical Study* (Almquist and Wiksell, Stockholm, 1965).
[67] A. Prieditis, Machine discovery of effective admissible heuristics, in: *Proceedings IJCAI-91*, Sydney, Australia (1991).
[68] R. Reiter, A semantically guided deductive system for automatic therem proving, in: *Proceedings IJCAI-73*, Stanford, CA (1973).
[69] E. Sacerdoti, Planning in a hierarchy of abstraction spaces, *Artif. Intell.* **5** (1974) 115–135.
[70] A. Simpson, Grazing: A stand alone tactic for theoretical inference, Master's Thesis, Department of Artificial Intelligence, University of Edinburgh, (1988).
[71] A. Simpson, Developing an abstraction for planning the unfolding of definitions, in: *Proceedings AI*IA International Conference* (1989); also: Tech. Rept. 8904-05, IRST, Trento, Italy (1989).
[72] C. Talcott and R. Weyhrauch, Towards a theory of mechanizable theories, 1. fol contexts—the extensional view, in: L.C. Aiello, ed., *Proceedings 8th European Conference on Artificial Intelligence*, Stockholm (1990) 634–639.
[73] J.D. Tenenberg, Preserving consistency across abstraction mappings, in: *Proceedings IJCAI-87*, Milan, Italy (1987) 1011–1014.
[74] J.D. Tenenberg, Abstraction in planning, Ph.D. Thesis, TR 250, Computer Science Department, University of Rochester, Rochester, NY (1988).
[75] A. Unruh and P. Rosenbloom, Abstraction in problem solving and learning, in: *Proceedings IJCAI-89*, Detroit, MI (1989).

[76] A. Unruh and P. Rosenbloom, Two new weak method increments for abstraction, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, boston, MA (1990) 134–140.

[77] R. Wang, The use of analogy in the PC system, unpublished.

[78] D.S. Weld, Reasoning about model accuracy, Tech. Rept. 91-05-02, Department of Computer Science and Engineering, University of Washington, Seattle, WA (1991).

[79] R. Weyhrauch, Prolegomena to a theory of mechanized formal reasoning, *Artif. Intell.* **13** (1) (1980) 133–170.

[80] B. Williams, Capturing how things work: constructing critical abstractions of local interactions, in: *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, MA (1990) 163–174.

[81] L. Wos, G. Robinson and D.Carson, The automatic generation of proofs in the language of mathematics, in: *Proceedings IFIP Congress 65*, Volume 2 (Barton Books, 1965) 325–326.