An Abstract Proof Checker*

Fausto Giunchiglia
Mechanized Reasoning Group
IRST, Loc. Panté di Povo
I 38050 Trento
Italy
fausto@irst.it

Toby Walsh
Mathematical Reasoning Group
Dept of Artificial Intelligence
University of Edinburgh
Scotland
T.Walsh@ed.ac.uk

Mathematicians rarely present proofs in all their detail; usually they give just an outline or abstraction of the proof. This paper describes our attempt to reproduce such activity within a computer proof system.

1 Introduction

Abstraction has been proposed as a powerful heuristic for controlling search in AI. Despite some promising theoretical results (*eg.* Korf has demonstrated that abstraction can potentially reduce an exponential search to a linear one [Kor79]), abstraction has proved, in theorem proving at least, less useful than expected. Plaisted, for example, claims:

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

Abstraction is, we agree, too weak a heuristic for *unguided* search. It should therefore be integrated with more powerful heuristics; the approach we propose here is to use it 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 a proof in the abstract space which is an outline of the original proof. The details can then integrated back into the outline, again 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 second author is a SERC PostDoctoral Fellowship. All the members of the Mathematical Reasoning group in Edinburgh and the Mechanized Reasoning group in Trento are thanked for their contributions to this work. Both authors would especially like to thank Alan Bundy.

The main motivation underlying our work is that this kind of reasoning is used all the time by human mathematicians: they first build an outline of the proof, then refine this outline by adding in details which have been abstracted away. For instance, Polya [Pol45] proposes a four part strategy for solving mathematical problems: understanding the problem, devising a plan, carrying out this plan, and finally examining the solution.

> *"... The way from understanding a problem to conceiving a plan may be long and tortuous. In fact, the main achievement in the solution of a problem is to conceive the idea of a plan ... To carry out a plan is much easier; what we need is mainly patience. The plan gives a general outline; we have to convince ourselves that the details fit into the outline."* [Pol45][pages 8 and 12]

Abstraction is one way of tackling the difficult problem of finding the plan. There are other technical motivations for this proposal (which, in part, also motivate the strategies used by mathematicians themselves):

Complexity: abstraction can greatly reduce proof complexity; for example, in this paper we demonstrate how a proof of Gödel's First Incompleteness theorem can be approximately halved in size from 59 steps to 30.

Explanation: such reductions in proof complexity make proofs more comprehensible and more easily explained. Just as mathematicians often only give an outline to their proofs, a proof system should be able to describe proofs without all the inessential details.

Analogy: abstractions of proofs might also provide a way to tackle analogy since analogical proofs will often have similar abstractions.

Our abstract proof checker is built on top of `GETFOL` [GW91], which is an extension and re-implementation of the `FOL` proof checking system [Wey80] [1]. There are many reasons for this choice. Most important is the *conversational* nature of `GETFOL` [GW91]: the interface with `GETFOL` is designed to make the interaction with the user more a dialogue than a sequence of orders. Thus, the user engages in a *conversation* with the proof system in which he/she describes the abstraction, and the outline of the proof, and then progressively refines this abstraction. The two-way nature of this conversation is essential for abstraction to be of practical utility, and to overcome the criticisms of Plaisted et al.

[1] Actually, `GETFOL` is far more than a conventional proof checker. For instance, it includes derived inference rules and complex deciders. A single proof step in `GETFOL` can thus represent very complex reasoning. We could perhaps call it an "interactive theorem prover". However, we shall stick to "proof checker" as we wish to emphasize our interest in the interaction with the system rather than in the automation of the construction of proofs.

The other main reason for using GETFOL is that it provides facilities for multitheoretic reasoning. That is, GETFOL allows for multiple distinct logical theories and provides inference rules, so called bridge rules, to "link" reasoning between distinct theories. This feature is essential for an abstract proof checker as the ground and the abstract space need to be two distinct theories. Bridge rules determine the properties which hold between ground and abstract spaces.

2 Abstraction

In [GW89a] we argued that abstraction can be seen as the mapping of one representation of a problem, the ground representation onto a new and simpler representation, the abstract representation. Problems can be represented by **axiomatic formal systems**; an abstraction is then simply a mapping from one formal system to another [GW90].

> **Definition 1 (Abstraction)** *: An* **abstraction** f*, written "*$f : \Sigma_1 \Rightarrow \Sigma_2$*" is a triple consisting of the formal systems* Σ_1*,* Σ_2 *and a total and computable function, which maps the language of* Σ_1 *onto that of* Σ_2*.*

Σ_1 is the **ground** space, and Σ_2 the **abstract** space. Since our focus is on proving theorems, an interesting restriction is to abstractions that preserve provability. As we show in [GW90, GW89a], one of the most interesting classes of abstractions (and one which captures nearly all the abstractions proposed in the past), is the class of **TI-abstractions**[2].

> **Definition 2 (TI-abstraction)** *: An abstraction* $f : \Sigma_1 \Rightarrow \Sigma_2$ *is a TI-abstraction iff, for any wff* φ*, if* $\varphi \in TH(\Sigma_1)$ *then* $f(\varphi) \in TH(\Sigma_2)$*.*

TI-abstractions are complete but not correct. The abstraction of a ground theorem is a theorem of the abstract space but not vice versa; there may be theorems of the abstract space which do not correspond to any theorem of the ground space. The existence of an abstract theorem must be understood more as a "suggestion" than as an assurance. We need a further step (that of **mapping back**) where we take this suggestion and information about the abstract proof, and try to prove the ground theorem.

[2] "T" stands for theorem and "I" for increasing.

Our aim is therefore to find abstractions in which the structure of abstract proofs is similar to that of ground proofs. Usually, the similarity is that the ground and abstract proofs have the same global structure. Hopefully, the abstract proof will also be shorter and easier to construct. As the following sections will show, there is a very general monotonicity property, called **tree subsumption** which is preserved by a very large class of abstractions, and which captures the relationship between ground and abstract proofs for practically all the abstractions of which we are aware.

3 Mapping Back

To define this monotonicity property, we must first give some notions for describing the structure of proofs. Proofs are special types of **formulae trees**, trees with every node labelled by a wff. Arcs describe the local structure of a tree. They also induce a global structure on trees; we can define when one node is below another, when it is above another, or (when neither of these two cases is true) when it is adjacent to another.

> **Definition 3** ($\preceq$) *: For any two nodes n_1, n_2 in a formulae tree, n_1 is* **below** *n_2, in symbols, $n_1 \preceq n_2$ iff*
>
> - $n_1 = n_2$*;*
> - $\langle n_1, n \rangle \in arcs(\Pi)$ *and* $n \preceq n_2$*.*

The below relation is a weak partial order. We will say that n_1 is **above** n_2 iff $n_2 \preceq n_1$, and that n_1 is **adjacent** to n_2, written $n_1 \bowtie n_2$ iff n_1 is not below or above n_2. Adjacency is an incomparability relation.

The below, above and adjacency relations allow us to formalise a monotonicity property between ground and abstract proof trees. Such a property satisfies the following two conditions:

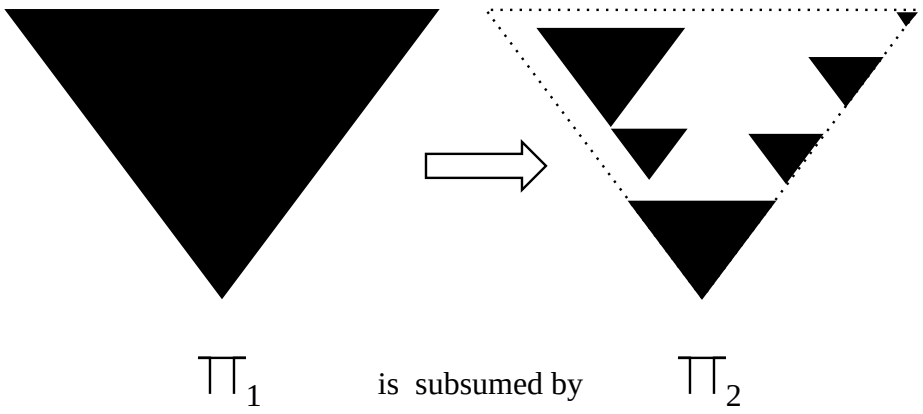
- **the preservation of the global structure of trees**; the below, above and adjacency relations should be maintained.
- **the preservation of the nodes**; all the nodes in the abstract tree (plus possibly more) should correspond to nodes in the ground tree.

These conditions motivate the following definition.

Definition 4 (Tree subsumption) *: If Π_1 and Π_2 are trees then Π_1* **subsumes** *Π_2, written $\Pi_1 \subseteq \Pi_2$, iff there exists an injective map, $\tau : nodes(\Pi_1) \mapsto nodes(\Pi_2)$ such that*

- $label(n) = label(\tau(n))$
- *if* $n_1 \preceq n_2$ *then* $\tau(n_1) \preceq \tau(n_2)$
- *if* $n_1 \bowtie n_2$ *then* $\tau(n_1) \bowtie \tau(n_2)$

The intuitive meaning of this definition is that the same wffs occur in Π_2 as in Π_1 (first condition) with the same global ordering (second and third conditions). The second condition guarantees that the relationship between wffs *within* a subtree remains the same, whilst the third condition guarantees that the relationship *between* wffs in distinct subtrees remains the same. Tree subsumption is a preorder being transitive, and reflexive. It is also a **monotonicity** property on the depth, the weight, the number of formulae occurrences, the ordering of wffs and the branches. We can represent tree subsumption graphically as follows:



Tree subsumption can be used to define a very general class of abstractions called **PI-abstractions**[3] which preserve the structure of proofs.

Definition 5 (PI-abstraction) *: An abstraction, $f : \Sigma_1 \Rightarrow \Sigma_2$ is said to be a* **PI-abstraction** *iff, for any proof Π_1 of a theorem φ in Σ_1, there exists a proof Π_2 of $f(\varphi)$ in Σ_2 with $\Pi_2 \subseteq f(\Pi_1)$.*

[3] "P" stands for proof, and "I" for increasing.

Note that it is the number of proofs in the abstract space (and not their weight or depth) that is increasing; indeed, their weight and depth is actually decreasing. By preserving the structure of proofs, a PI-abstraction preserves provability. That is, a PI-abstraction is also a TI-abstraction.

4 An Abstract Proof Checker

We can now outline how PI-abstractions can be used to guide proof checking. As a subset of the nodes of the proof in the ground space map onto the nodes of the proof in the abstract space, the individual steps in the abstract proof provide the islands to be bridged by proof checking in the ground space.

The process of abstract proof checking can thus be broken down into four steps:

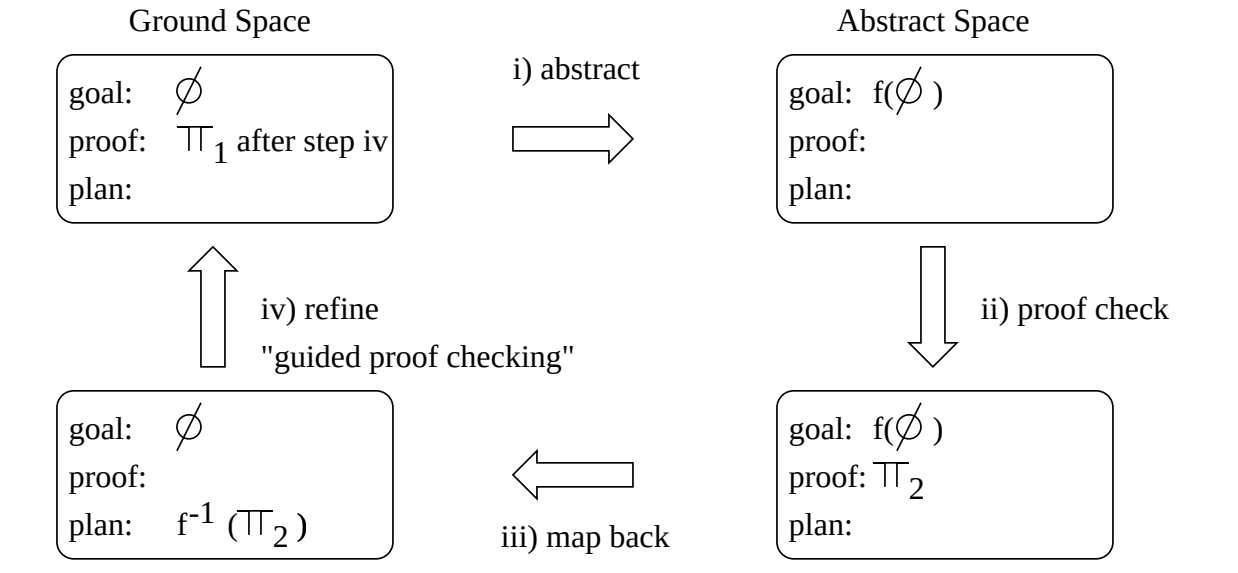
(i) we **abstract** the formula to be proved from the ground space onto the abstract space;

(ii) we "**proof check**" an abstract proof;

(iii) we **map back** (unabstract) the abstract proof; this gives us a plan that provides the major steps (or islands) we will jump between in the ground proof;

(iv) we **refine** the plan by filling in the gaps between the islands.

This process can be represented graphically as follows:



Notice that this process can be repeated for any single theorem or subgoal. The abstract proof checker provides a set of primitives for each of the four steps. GETFOL's context mechanism [GW91] allows for the *simultaneous* representation of ground and abstract spaces. A GETFOL context is a very complex notion, only part of which we need to use. It can be seen as a logical theory with its own language, axioms and deductive machinery. The abstract proof checker allows for the creation of and the switching between different contexts. Step **(i)**, the mapping from ground to abstract spaces, is built on top of the context mechanism; essentially it consists of a rewriting step plus a context switch. A lot of complexity arises from the need to keep track of the appropriate signature (either that of the abstract space or that of the ground space). We also need commands to abstract axioms, to abstract theorems and so on. For example, the following sequence of commands declares a ground and abstact space, defines an abstraction, declares an axiom in the ground space and then maps this onto an axiom of the abstract space (the axioms and abstraction are taken from the example in the next section).

```
GETFOL:: NAMECONTEXT maths;
You have named the current context: maths
GETFOL:: MAKECONTEXT absmaths;
You have created the context: absmaths
GETFOL:: DEFINE ABSTRACTION f:maths=>absmaths BY ...;
GETFOL:: AXIOM m0: provable(peano,diag(x) IFF ~ prf(formno(diag(x)),x));
GETFOL:: SHOW AXIOM m0;
   m0: provable(peano,diag(x) IFF ~ prf(formno(diag(x)),x))
GETFOL:: ABSTRACT m0 to absmaths by f;
GETFOL:: SWITCHCONTEXT absmaths;
You are now using context: absmaths
GETFOL:: SHOW AXIOM m0;
   m0: provable(peano,~d IFF d)
```

Steps **(iii)** and **(iv)**, unabstracting the outline and filling in the details, are more difficult to implement than the first two steps. The problem is that, with TI-abstractions, the result of mapping back an asbtract theorem is not in general a theorem. Even worse, abstractions are usually not bijective; as a consequence, we must chose what we map back to. We are considering a technique based on higher order unification, and "middle-out" reasoning (that is, reasoning not from the beginning or the end but the middle of the proof) for this task. This part will be discussed only briefly in the paper.

5 An Example

We consider a theorem closely related to Gödel's First Incompleteness theorem[4]. For simplicity, we will skip much of the proof, and assume that Gödel numbering has been defined and several of the key lemmas have been proved. In particular, we'll assume the diagonalisation lemma:

```
AXIOM m0:  provable(peano,diag(x) IFF ~ prf(formno(diag(x)),x));
```

where `prf(Fn,Pn)` is true iff `Pn` is the Gödel number of a proof of the formula whose Gödel number is `Fn`.

Additionally we'll need various properties of consistency, provability, validity and substitution, and of the theory `peano` and the model `arith`. For example:

```
AXIOM m2:  consistent(peano);
AXIOM m3:  consistent(Th) imp not provable(Th,F) or not provable(Th,~F);
AXIOM m4:  provable(peano,F) imp valid(arith,F);
AXIOM m7:  provable(Th,F) and provable(Th,F IFF G) imp provable (Th,G));
AXIOM m11: subst(el(M),X,F,G) and valid(M,G) imp valid(M,all(X,F)));
```

`el(M)` is a random element of the model `M`, and `subst(T,X,F,G)` is true if `G` is the result of substituting term `T` for variable `X` in the formula `F`.

We want to prove that:

```
exists F. valid(arith,F) and not provable(peano,F)
```

That is, there exists a formula `F` which is valid in the model `arith` but not provable in `peano` arithmetic. A complete proof of this theorem is very complicated, requiring some 59 steps. In his unpublished note, Alan Bundy has proposed a PI-abstraction which simplifies the proof greatly but still provides the key steps. The terms `diag(x)`, `diag(el(arith))`, `all(x,diag(x))`, `~prf(formno(diag(x)),x)`, and `~prf(formno(diag(x)),el(arith))` are abstracted onto `~d`. Similarly, the two terms `prf(formno(diag(x)),x)`, and `prf(formno(diag(x)),el(arith))` are mapped onto `d`. All the other terms (and all predicate names) are left unchanged. Formally, this mapping is a function abstraction (which collapses functions onto constants) [GW90, GW89a].

[4]This example is adapted from an (unpublished) note by Alan Bundy. We are greatly indebted to Alan Bundy for this contribution.

Under such an abstraction, more than half the axioms become redundant and the proof halves in size; more importantly, every step in the abstract proof corresponds to (the abstraction of) an important step in the ground proof; that is, the abstract proof tree subsumes (the abstraction of) the ground proof. In the long paper we will describe this proof in more detail. In the figure below, the key steps of the two proofs are illustrated side by side:

Abstract Proof	
15	`not provable(peano,~d)`
20	`not provable(peano,d)`
27	`provable(peano,~d)`
29	`valid(arith,~d)`
30	`valid(arith,~d) and`
	`not provable(peano,~d)`

Ground Proof	
32	`not provable(peano,all(x,diag(x))`
38	`not provable(peano,`
	`prf(formno(diag(x),el(arith)))`
53	`provable(peano,diag(el(arith)))`
58	`valid(arith,all(x,diag(x))`
59	`valid(arith,all(x,diag(x))) and`
	`not provable(peano,all(x,diag(x)))`

Note that the abstract space is inconsistent; indeed at line `15` we demonstrate that `not provable(peano,~d` whilst later on, at line `27`, we prove that `provable(peano,~d)`. In [GW89b] we demonstrate that inconsistency is inevitable with the use of abstractions like this. However, provided we are careful not to exploit this inconsistency in our proofs, this does not cause problems [GW89b].

6 Conclusions

We have proposed using abstraction to guide a proof checking system. This is very similar to the way abstraction is used by human mathematicians. We have demonstrated the utility of this strategy by means of a computer assisted proof of Gödel's First Incompleteness theorem. This use of abstraction seems to overcome many of the problems associated with the use of abstraction in automated theorem proving. Such a use of abstraction has the potential to transform traditional mechanized reasoning.

References

[GW89a] F. Giunchiglia and T. Walsh. Abstract Theorem Proving. In *Proceedings of the 11th IJCAI*, International Joint Conference on Artificial Intelligence, 1989.

Also available as DAI Research Paper No 430, Dept. of Artificial Intelligence, Edinburgh.

[GW89b] F. Giunchiglia and T. Walsh. Abstracting into inconsistent spaces (or the false proof problem). In *Proceedings of AI*IA 89*, Associazione Italiana per l'Intelligenza Artificiale, 1989. Also available as DAI Research Paper, Dept. of Artificial Intelligence, Edinburgh.

[GW90] F. Giunchiglia and T. Walsh. *A Theory of Abstraction.* Research Paper 516, Dept. of Artificial Intelligence, University of Edinburgh, 1990. Submitted to Journal of Artificial Intelligence.

[GW91] F. Giunchiglia and T. Walsh. *Abstract theorem proving: mapping back.* Research Paper 460a, Dept. of Artificial Intelligence, University of Edinburgh, 1991. This is a revised version of DAI Research Paper 460.

[Kor79] R.E. Korf. Planning as search: a quantitative approach. *Artificial Intelligence*, 33:65–88, 1979.

[Pol45] G. Polya. *How to Solve It.* Princeton University Press, Princeton, NJ, 1945.

[Wey80] R.W. Weyhrauch. Prolegomena to a theory of Mechanized Formal Reasoning. *Artificial Intelligence. Special Issue on Non-monotonic Logic*, 13(1), 1980.