Proving Theorems By Using Abstraction Interactively *

Roberto Sebastiani[1], Adolfo Villafiorita[1], Fausto Giunchiglia[2,3]

[1] Mechanized Reasoning Group, D.I.S.T., University of Genoa, Italy

[2] Mechanized Reasoning Group, I.R.S.T., 38050 Povo Trento, Italy.

[3] University of Trento, Via Inama 5, 38100 Trento, Italy.

rseba@dist.unige.it adolfo@dist.unige.it fausto@irst.it

Abstract

In this paper we show how an interactive use of abstraction in theorem proving can improve the comprehension and reduce the complexity of many significant problems. For such a task we present a fully mechanized example of the very well-known map colouring problem.

1 Introduction

By "abstraction" we informally mean the process by which, starting from a given representation of a problem (called "ground space"), we construct a new and simpler representation (called "abstract space"), we find a solution for it and hence we use such a simplified solution as an outline for the solution of the original problem. The abstract space is obtained by throwing away "details", i.e. inessential information which can be reconsidered in a second moment. This informal definition is very general and identifies as "abstraction" a large number of applications in areas like theorem proving, problem solving, planning, reasoning by analogy, learning, common sense reasoning, hardware and software verification.

Since our focus is on theorem proving (but also in problem solving and planning), by "problems" we mean goals to be proved in axiomatic formal systems. In this sense, the most common use of abstraction has been *to abstract the goal, to prove its abstracted (and hopefully simpler) version and then to use the structure of the resulting proof as an outline to help construct the proof of the original goal.* There are many different possible forms of abstraction in theorem proving, each corresponding

*Paolo Pecchiari, Marco Roveri and Alessandro Zorer are thanked for their contribution to the work underlying this paper.

to a particular notion of "detail": collapsing constants [Hob85], collapsing predicate or function symbols [Pla80, GW92a, Ten88], dropping arguments to function or predicate symbols [Mel87], deleting preconditions in operators [Sac74, BGW91], etc. This is why any attempt to use abstraction in general-purpose theorem proving must be based on a very wide notion of abstraction.

A satisfactory definition of abstraction is given in [GW92a, GW92b], which provide the basic theoretical foundation to this paper. In [GW92a] an abstraction f, written "$f : \Sigma_1 \Rightarrow \Sigma_2$ ", is formally defined as a triple $\langle \Sigma_1, \Sigma_2, f \rangle$ consisting of the formal systems Σ_1 and Σ_2 and a total and computable function, f, which maps the language of Σ_1 onto that of Σ_2 [1]. This definition has been proved to be sufficiently general to capture most abstractions defined in the literature. The framework also gives the definitions and proves the properties necessary to provide the theoretical background for an effective use of abstraction in theorem proving.

The goal of this paper is to show how an interactive use of abstraction in theorem proving can both improve the comprehension and reduce the complexity of many significant problems. In [BGVW93] a proof of Gödel's First Incompleteness theorem has been halved in size by interactive abstract theorem proving. In this paper, we will consider as an example the very well-known problem of colouring a plane map using only four colours. We will show how the main steps of Kempe's algorithm [Kem79] can be represented as an abstraction, so that a general-purpose theorem prover can pass from an exponential search to a linear one to solve many instances of a map coloring problem. For such a task, in the following sections we will make use of the prototype of `ABSFOL` [GSW93], an interactive theorem prover via abstraction on development [2]. The goal of the `ABSFOL` project has been to use the theory of abstraction described in [GW92a, GW92b] as the theoretical reference to develop a general purpose environment for using abstraction in automated deduction, i.e. both for proving theorems in various branches of mathematics and logic and for formalizing common sense reasoning.

In Section 2 we present and motivate the interactive approach to abstract theorem proving. In Section 3 we introduce the example of map colouring problem, showing that a skilled usage of abstraction can guide a theorem prover to linear solutions. In Section 4 we describe in details the five steps of interactive abstract theorem proving. For such a task, we present a full mechanization of a map colouring example.

2 Using abstraction interactively

Experimentally, a *totally automatic* use af abstraction in theorem proving has been shown less useful than expected [Pla90, GW92a]. We can see two main motivations for such a failure. First, it seems very hard, if not impossible, to conceive an algorithm

[1] From now on, we will often use the words *space*, *context* and *formal system* as synonyms.

[2] `ABSFOL` is implemented on top of `GETFOL` [Giu92], an interactive theorem prover which is a re-implementation/extension of the `FOL` system [Wey80]. `GETFOL` allows the user to perform theorem proving simultaneously in many distinct logical theories ("contexts"), and it also provides inference rules ("bridge rules") which link proofs among distinct theories. This feature is essential for `ABSFOL`, as the ground and the abstract space must be two distinct interacting theories.

which chooses the appropriate abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ "ad hoc" for any possible goal φ of any possible formal system Σ_1. A definition "ad hoc" of the abstraction f is absolutely essential for a satisfactory solution of any particular problem. In fact, f must keep *all and only* the information which is essential for the proof of the given theorem φ, throwing away the inessential details (note that f can eliminate information necessary to prove many other possible goals). A finely-chosen abstraction can introduce a strong simplification in the representation of the problem [GW92a] while preserving sufficient information for the mapping back of the abstract solution. By comparison, a bad choice can either complicate and hinder the abstract proving step, or lead to an abstract proof which is useless. Secondly, it seems impossible to conceive a mechanization of the idea of "using the abstract proof as an outline for the proof of the original goal" general enough to work automatically for any formal system. Only very specific-purpose applications have been actually implemented (e.g. [Sac74]).

In order to overcome such problems, in [GSW93] an *interactive* approach to abstract theorem proving has been proposed. Following such approach, an external guide (either human or mechanized) is required in those steps where a "clever" choice is essential for avoiding complexity explosions, while the repetitive and dumb steps are performed by automatic subsystems. This means that the user can *define* his (or its) abstraction "ad hoc" for the problem to solve, and then use the system to find a proof in the abstract space which is hence "mapped back" into an outline of the original proof. The details can then be integrated back into the outline, till it is automatically checked to be a correct proof. In the most general case, this is represented by the following five step schema:

(i) **choose** and **define** a suitable abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$;

(ii) **abstract** the goal φ from Σ_1 to Σ_2, i.e. apply the abstraction f to φ ($f(\varphi)$); the axioms of Σ_1 can be abstracted too;

(iii) **prove** the abstract goal, i.e. find in Σ_2 a proof Π_2 of the abstract goal $f(\varphi)$;

(iv) **map back** the abstract proof; this means create in Σ_1 an *outline* Π^* of the future ground proof, such that $f(\Pi^*) = \Pi_2$; as f is generally not injective, Π^* can be parametric;

(v) **refine** the outline Π^* by instantiating the (eventual) parameters and "filling in the gaps" between individual steps of the outline, till it is checked to be a correct proof Π_1 of φ.

A formal definition of proof outline has been given in [GW92b]. Roughly speaking, an outline of a proof Π_1 is given by a tree connecting the key wffs of Π_1, whose arcs in general do not represent direct inferences. The former schema suggests to use the proof of the abstract goal to generate an outline for the proof of the original goal. Of course, this forces to restrict the set of the suitable abstractions to those for which such operation is possible. Fortunately, in [GW91, GW92a, GW92b] such a set is proved to include most abstractions used in AI [3]. This means that, given an

[3] Actually, such a proof uses the definition of a class of abstractions, called *proof increasing* (PI) [GW91]. In particular, a very common type of abstractions, called *atomic*, are proved to be PI on very weak conditions [GW91, GW92a].

abstraction f belonging to such a set, for any ground theorem φ, exists in Σ_2 a proof Π_2 of the abstracted theorem $f(\varphi)$; moreover exists (at least) one way to "map back" (at least one of the possible) Π_2 to Σ_1 which gives rise to an outline Π^* of the ground proof. Π^* can be subsequently refined.

It is important to observe that in interactive abstract theorem proving there is no need to interact only with human beings. Thus, in order to achieve automatic theorem proving in limited domains, an interactive abstract theorem prover, like `ABSFOL`, could be interfaced with external control programs based on euristic strategies [Kno91, BGW91]. Moreover, such system can be provided with an internal ML-like metalanguage for writing programs which implements control strategies, called *tactics* [GT94]. Automated theorem proving can then be implemented by writing complex tactics.

3 The map colouring problem

As an example of interactive abstract theorem proving, in this paper we will consider the very well-known problem of colouring the various regions of a planar map using only four different colours. In Figure 1 we find a very simple example taken from [McC90]. According to McCarthy, we can formalize the problem in terms of theorem proving as follows. We need a set of variables (like `Albania, Andorra, Austria,` ... or `r1,r2,` ...) each representing a region of the given map, a binary predicate `n()` "*next*", representing the frontier constraints (e.g. `n(Austria,Italy)`), and a set of the four individual constants `y`, `b`, `g`, `r` (yellow, blue, green, red). Then we define an axiom "`colours`", written as a conjunction of colouring constraints like `n(y,b)` ("a yellow-coloured state can border a blue-coloured one"). Using `ABSFOL`, the problem can be initialized as follows: [4]

```
NONAME:: NAMECONTEXT gmap;
GMAP:: DECLARE INDCONST y b g r;
GMAP:: DECLARE INDVAR r1 r2 r3 r4 r5 r6;
GMAP:: DECLARE PREDCONST n 2;
GMAP:: AXIOM colours :
n(y,b) and n(y,g) and n(y,r) and n(b,y) and n(b,g) and n(b,r) and
n(g,y) and n(g,b) and n(g,r) and n(r,y) and n(r,b) and n(r,g);
```

The goal is given by the conjuction of all and only the frontier constraints of the map:

```
exists r1 ... r6. ( n(r1,r2) and n(r1,r3) and ... and n(r5,r6) )
```

[4] In `ABSFOL`, `Teletype` font is used to write the input and output of `ABSFOL`. `UPPERCASE TELETYPE` is used for key-words. "`<string>::`" is the `ABSFOL` prompt: the string before "`::` " is the name of the current context, that is the theory we are working in. `NAMECONTEXT` names the current context. `DECLARE` adds new symbols to the current context. `AXIOM` adds an axiom to the current context. Both the input and the output have been slightly edited to make them more readable.

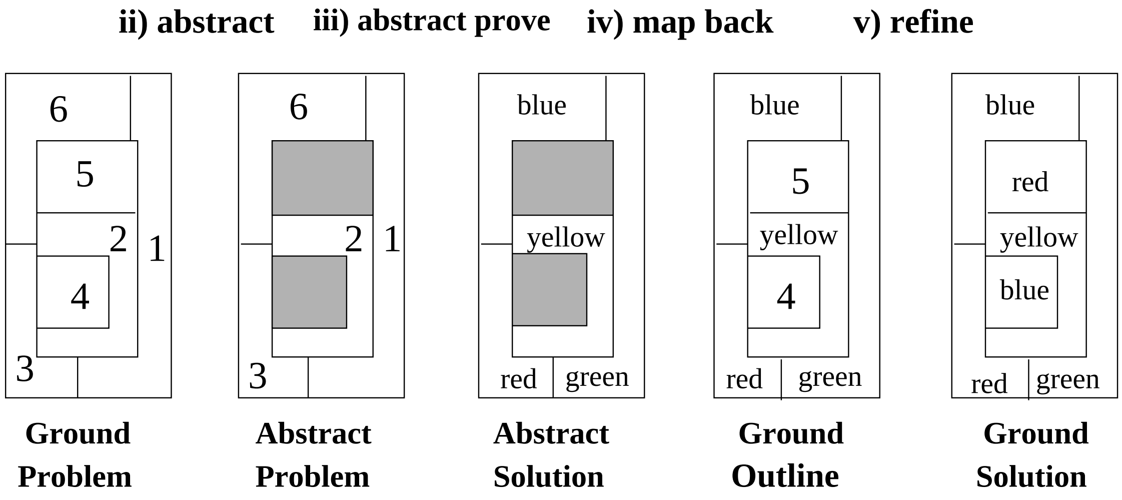

Figure 1: A solution to a map colouring problem via abstraction.

Using a formalization like this, the solution is usually very computationally explosive for any automatic theorem prover [McC90], as it involves a lot of backtracking. According to Mozetič [Moz90], for a map of 32 European countries a prolog program implementing a chronological backtracking algorithm needs 12 CPU-time days. We want to show that abstraction can guide a theorem prover to solve many instances of the problem in linear time, eliminating any backtrack [5].

Following Kempe [Kem79] we first note that any region with three or fewer neighbours presents no problem, as there is always a colour available. Thus we can abstract the original map by eliminating all such countries (see Fig 1). In our formalization, this means mapping any frontier constraint involving such less-relevant countries (e.g. `n(r3,r4)`) into `TRUE` (step (ii)). This gives rise to a simpler representation of the problem, as the number of countries will have decreased and, most important, many of the countries left will surely loose some neighbours. Thus we can iterate this process until we have no region with more than three neighbours: this final representation is automatically solvable at the first attempt in linear time, without backtracking (step (iii)). Each level's solution becomes a plan for the next lower level (step (iv)), and can also be refined without backtracking as it also needs colouring only less-then-four-bordered regions (step (v)).

We can use this process with complicate maps. For instance, we can colour a 1989 Europe map [6] using only three consequent abstractions. The first abstraction introduces the greatest simplification of the problem, as it eliminates the following countries: *Spain, Portugal, Andorra, Ireland, United Kingdom, Iceland, Norway, Sweden, Finland, Holland, Luxembourg, Lichtenstein, Denmark, Poland, Turkey, Monaco,*

[5] Even if a very simple input problem is given, a standard algorithm for automatic theorem proving could need some backtracking to solve it [McC90]. For instance, if we try to colour the map in Figure 1 starting from countries `r4,r5` by using an ordinary chronological backtracking algorithm, we will need more than 24 steps instead of 6.

[6] In order to make comparisons with [Moz90], we refer to a 1989 Europe map, with still Czechoslovakia, Yugoslavia and two distinct german nations.

S.Marino, Vatican City, Albania, East Germany. As there are still some countries with more than three neighbours, we must iterate the process. The second and third abstraction remove respectively *Belgium, Czechoslovakia, U.S.S.R., Bulgaria, Greece,* and *France, West Germany, Romania, Hungary.* Now the abstract problem is sufficiently reduced to be solved directly, as there are only four countries left: *Switzerland, Italy, Austria, Yugoslavia.* Now the whole colouring process follows easily, by colouring at each step all the nations of a different abstract level (in linear time). As a whole, three concatenated abstractions have guided the system to a complete colouring, *with no backtracking.*

4 The five steps

In this section we describe some of the details of the five steps of interactive abstract theorem proving, as they are listed in the schema of Page 3. For such a task, we present a full mechanization of the map colouring example of Figure 1.

4.1 Step (i): abstraction definition

As we said in Section 2, in order to achieve satisfactorily an abstract-checked proof for a given goal φ in Σ_1, the abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ must be very specific for φ. For such a task `ABSFOL` provides some tools for defining abstractions [Seb93]: a *language*, general enough to preserve all the expressive power of the formal definition of abstraction [GW92a]; an *interpreter* for such a language, which allows for the automatic (partial or total) generation of the abstract space; a hardwired library of context-independent abstractions (*abstraction prototypes*), which can be instantiated into the desired application.

In order to solve the map colouring problem of Figure 1, the user can define the abstraction `kempe`, which eliminates the regions `r4` and `r5` [7]:

```
GMAP:: MAKECONTEXT amap;
GMAP:: DEFINE ABSTRACTION kempe : gmap ==> amap
BY PROTOTYPE logic_pres
   ....
NEWRULES
        ABS( n(r4,$x) )    := TRUE
        ABS( n($x,r4) )    := TRUE
        ABS( exists r4. $A ) := ABS($A)
        ABS( n(r5,$x) )    := TRUE
        ABS( n($x,r5) )    := TRUE
        ABS( exists r5. $A ) := ABS($A);
```

[7] `MAKECONTEXT` creates a new empty context. `gmap` and `amap` are the names of the ground and abstract contexts. The symbols beginning with "$" are meta-variables. As `ABSFOL` is still at prototype level, the syntax of the commands can change from time to time.

An abstraction is defined by a set of (terminating) rewrite rules, plus a description of the mapping of the alphabet symbols. The declaration of `kempe` can be easily performed by instantiating an abstraction prototype, that is a pre-compiled set of rules: "similar" abstractions are obtained from the same prototype. In particular `logic_pres` consists of rules like:

```
ABS(forall $x. $A) := forall $x. ABS($A)
ABS($A and $B)     := ABS($A) and ABS($B)
```

which preserve the logical stucture of a wff. We can note that it is easy to write an external program able to generate automatically the suitable abstraction for a given map colouring problem:

{*initialize abstraction* $f : \Sigma_1 \Rightarrow \Sigma_2$ }
for *any country* r_i **do**
  if {r_i *occurs less then four times in the goal*}
  then {*add to* f *the rules for eliminating* r_i}

The computational cost of such programs is $O(n^2)$.

4.2 Step (ii): abstraction application

Step (ii) consists of the generation of an abstracted version of a given problem (an abstract context with new axioms and a new goal). This is performed by means of *bridge rules*, i.e. rules which allow us to derive facts in a context from one in another context: any application can be considered to be a form of inference in a multi-context system [Giu93]. In our example: [8]

```
GMAP:: ABSTRACT ALPHABET BY kempe;
'r' is declared to be a new indconst in 'amap'.
....
GMAP:: ABSTRACT ALLAXIOMS BY kempe;
colours:
n(y,b) and n(y,g) and n(y,r) and n(b,y) and n(b,g) and n(b,r) and
n(g,y) and n(g,b) and n(g,r) and n(r,y) and n(r,b) and n(r,g);
is a mapped axiom in 'amap'.
GMAP:: ABSTRACT GOAL
exists r1 r2 r3 r4 r5 r6.(
    n(r1,r2) and n(r1,r3) and n(r1,r5) and n(r1,r6) and
    n(r2,r3) and n(r2,r4) and n(r2,r5) and n(r2,r6) and
    n(r3,r4) and n(r3,r6) and n(r5,r6)) BY kempe;
exists r1 r2 r3 r6.
   (n(r1,r2) and n(r1,r3) and TRUE and n(r1,r6) and
    n(r2,r3) and TRUE and TRUE and n(r2,r6) and TRUE and
    n(r3,r6) and TRUE)
is the mapped goal from 'gmap'.
```

[8] The lines which follow ";" and are before the prompt are ABSFOL's output.

The first command completely generates the language of amap; the second abstracts all the ground axioms by adding the mapped axioms to amap; the last abstracts the goal. The whole action could have been achieved in only one step:

```
GMAP:: ABSTRACT CONTEXT BY kempe;
```

In such case the action becomes fully automatic. The simplified representation of the problem we have obtained is equivalent to the second pattern of Figure 1.

4.3 Step (iii): theorem proving in the abstract space

By step (iii) we mean "standard" theorem proving in a (hopefully simpler) abstract context. Colouring a map is a typical form of backward theorem proving, from the goal to the axioms. Unfortunately, the implementation of backward inference rules inside ABSFOL is still ongoing [Ago93, Zor93]. Thus we have to perform a "forward" proof, by applying a tautological decider [Giu92] and an "exist introduction" rule [Pra65].

```
AMAP:: TAUT
 n(g,y) and n(g,r) and TRUE and n(g,b) and n(y,r) and
 TRUE and TRUE and n(y,b) and TRUE and n(r,b) and TRUE
BY colours;
1   n(g,y) and n(g,r) and TRUE and n(g,b) and n(y,r) and
    TRUE and TRUE and n(y,b) and TRUE and n(r,b) and TRUE
AMAP:: existi 1  b:r6  r:r3  y:r2  g:r1;
2 exists r1 r2 r3 r6.(
   n(r1,r2) and n(r1,r3) and TRUE and n(r1,r6) and n(r2,r3) and
   TRUE and TRUE and n(r2,r6) and TRUE and n(r3,r6) and TRUE)
```

This proof must be conceived backward, from the goal 2 to the axiom colours. In this sense, the colouring is performed (backward) by the "exist introduction" rule, which passes from the goal 2 to the "coloured map" 1, which is proved to be correctly coloured by the tautological decider TAUT. The proof represents the third pattern of Figure 1.

4.4 Step (iv): map back

Mapping back an abstract proof Π_2 into the ground context means creating an outline Π^* in the ground context such that $f(\Pi^*) = \Pi_2$. As we said in Section 2, a clever previous choice of the abstraction $f : \Sigma_1 \Rightarrow \Sigma_2$ ensures that such an outline exists [GW92b]. The single elements of an outline are called *tries*. Abstractions are usually not injective, so that the outline Π^* resulting from mapping back the abstract proof Π_2 is not uniquely determined. The choice of Π^* is as critical as the choice of the abstraction itself, so that step (iv) must be performed interactively. In ABSFOL the mapping back step generates a parametric outline [9]:

```
AMAP:: MAPBACK PROOF BY kempe;
1.0  n(g,y) and n(g,r) and TRUE and n(g,b) and n(y,r) and
     TRUE and TRUE and n(y,b) and TRUE and n(r,b) and TRUE
2.0  exists r1 r2 r3 r6.(
   n(r1,r2) and n(r1,r3) and TRUE and n(r1,r6) and n(r2,r3) and
   TRUE and TRUE and n(r2,r6) and TRUE and n(r3,r6) and TRUE)
GMAP::
```

This must be subsequently instantiated. In our example, try `2.0` is instantiated into the goal wff [10], while try `1.0` is instantiated in such a way to preserve the "existi" inference relation with try `2.0`. These steps are performed by the command `INSTWFF`:

```
GMAP:: INSTWFF 1.0 ... :  exists r4 r5. ... and n(r5,b))  BY kempe;
1.0  exists r4 r5. (
  n(g,y) and n(g,r) and n(g,r5) and n(g,b) and n(y,r) and n(y,r4)
  and n(y,r5) and n(y,b) and n(r,r4) and n(r,b) and n(r5,b))
GMAP:: INSTWFF 2.0 ... :  exists r1 r2 ... and n(r5,r6))  BY kempe;
2.0  exists r1 r2 r3 r6 r4 r5. (
     n(r1,r2) and n(r1,r3) and n(r1,r5) and n(r1,r6) and
     n(r2,r3) and n(r2,r4) and n(r2,r5) and n(r2,r6) and
     n(r3,r4) and n(r3,r6) and n(r5,r6))
```

The result is a non-parametric ground outline which corresponds to the fourth pattern in Figure 1. `INSTWFF` checks whether the resulting wff is mapped by $f : \Sigma_1 \Rightarrow \Sigma_2$ into the original abstract wff. If not, the instantiation fails. This ensures that $f(\Pi^*) = \Pi_2$. The system keeps track of the inference story of any proof, so that it "supposes" by default that the inferences relating the abstract facts are preserved between the ground tries. In our example, try `2.0` still follows from `1.0` by applying `existi r1...`, but `1.0` can be no more inferred by applying `TAUT` to `colours`. Thus we need some extra try to "fill the gap" between `colours` and `1.0`.

4.5 Step (v): Refine

ABSFOL provides an environment for manipulating proof outlines, and a tool able to check whether a refined outline is a proof.

In our example, we still have to find a colour for regions `r4,r5`, i.e. we must look inside the wff `1.0` for a suitable instantiation of variables `r4,r5`. It is very easy to see that `b,r` is a correct choice. Thus, we can infer the "coloured" formula as a tautological conseguence of the axiom `colour` [11]:

[9] Each formula displayed as "`n.0`" can be parametric as it is meant to represent a set of ground proof lines ("facts") which abstract onto proof line `n`. Note that the parametric wff of any try `n.0` looks the same as the wff of abstract proof line `n`.

[10] This is a very general instantiation criteria: the abstract image of the goal must be mapped back into the goal itself.

[11] `TRYTAUT` is similar to `TAUT`: it generates a try instead of a proof line.

```
GMAP:: TRYTAUT n(g,y) and n(g,r) and ... and n(r,b) BY colors;
0.1  n(g,y) and n(g,r) and n(g,r) and n(g,b) and n(y,r) and
     n(y,b) and n(y,r) and n(y,b) and n(r,b) and n(r,b) and n(r,b)
```

Now we must bridge up tries `0.1` and `1.0`. For such a task, we can infer a new try `0.2` (identical to try `1.0`) by "exist introduction" on `0.1`:

```
GMAP:: TRYEXISTI 0.1  r:r5 ... b:r4 ... ;
0.2  exists r4 r5. (
     n(g,y) and n(g,r) and n(g,r5) and n(g,b) and n(y,r) and n(y,r4)
     and n(y,r5) and n(y,b) and n(r,r4) and n(r,b) and n(r5,b))
GMAP:: TRYMATCH 0.2 1.0;
1.0 has been bound to 0.2.
```

Now the outline `0.1 - 0.2/1.0 - 2.0` should be a correct proof for the goal. The command `OL2PROOF` checks whether the current outline is a proof of the last try. If so, the corresponding proof is generated:

```
GMAP:: OL2PROOF;
The current outline is a proof.
1   n(g,y) and n(g,r) and n(g,r) and ...
                                       ... and n(r,b)
2   exists r4 r5. (n(g,y) and n(g,r) and ...
                                        ... and n(r5,b))
3   exists r1 r2 r3 r6 r4 r5. (n(r1,r2) and ...
                                        ... and n(r5,r6))
```

This corresponds to the final pattern of Figure 1.

5 Conclusions

We have shown how an interactive use of abstraction in theorem proving can improve the comprehension and reduce the complexity of many significant problems. For such a task we have considered (a fully mechanized example of) McCarthy's formalization of the map-colouring problem. We have shown how the main steps of Kempe's algorithm for map-colouring problem can be represented as abstractions, which can guide a general-purpose interactive theorem prover like `ABSFOL` to a quicker solution. By this way, we can avoid any backtracking and pass from an exponential search to a linear one.

References

[Ago93] A. Agostini. Bidirectional natural deduction. *AI*IA Notizie*, 4, 1993.

[BGVW93] A. Bundy, F. Giunchiglia, A. Villafiorita, and T. Walsh. Gödel's Incompleteness Theorem via Abstraction. Technical Report 9302-15, IRST, Trento, Italy, 1993. Also DAI Research Paper, University of Edinburgh.

[BGW91] A. Bundy, F. Giunchiglia, and T. Walsh. Calculating criticalities. Technical report, Dept. of Artificial Intelligence, University of Edinburgh, University of Edinburgh, 1991. Submitted to Artificial Intelligence. Also IRST-Technical Report 9112-23, December 1991.

[Giu92] F. Giunchiglia. The `GETFOL` Manual - `GETFOL` version 1. Technical Report 92-0010, DIST - University of Genova, Genoa, Italy, 1992.

[Giu93] F. Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, XVI:345–364, 1993. Short version in Proceedings IJCAI'93 Workshop on Using Knowledge in its Context, Chambery, France, 1993, pp. 39–49. Also IRST-Technical Report 9211-20, IRST, Trento, Italy.

[GSW93] F. Giunchiglia, R. Sebastiani, and T. Walsh. A general purpose reasoner for abstraction. Technical Report 9301-08, IRST, Trento, Italy, 1993.

[GT94] F. Giunchiglia and P. Traverso. Program Tactics and Logic Tactics. In *Proceedings 5th Intnl. Conference on Logic Programming and Automated Reasoning (LPAR'94)*, Kiev, Ukraine, July 16-21, 1994. Also IRST-Technical Report 9301-01, IRST, Trento, Italy. Presented at the Third International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, January 1994.

[GW91] F. Giunchiglia and T. Walsh. Mapping back. Technical Report 9111-02, IRST, Trento, Italy, 1991. Revised version of IRST-Technical Report no. 8911-16. Also DAI Research Paper, University of Edinburgh.

[GW92a] F. Giunchiglia and T. Walsh. A Theory of Abstraction. *Artificial Intelligence*, 56(2-3):323–390, 1992. Also IRST-Technical Report 9001-14, IRST, Trento, Italy.

[GW92b] F. Giunchiglia and T. Walsh. Tree subsumption: Reasoning with outlines. In *Proc. 10th European Conference on Artificial Intelligence ECAI-92*, pages 77–81, Vienna, Austria, 1992. Also IRST-Technical Report 9205-01, IRST, Trento, Italy.

[Hob85] J.R. Hobbs. Granularity. In *Proc. of the 9th International Joint Conference on Artificial Intelligence*, pages 432–435, 1985.

[Kem79] A. B. Kempe. On the geometrical problem of four colors. *American Journal of Mathematics*, 2:193–204, 1879.

[Kno91] C.A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, 1991.

[McC90] J. McCarthy. Coloring Maps and the Kowalski Doctrine. In John. McCarthy, editor, *Formalizing Common Sense - Papers by John McCarthy*, pages 167–174. Ablex Publishing Corporation, 1990. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 167–174.

[Mel87] T.F. Melham. Abstraction mechanisms for hardware verification. Technical Report 106, University of Cambridge, Computer Laboratory, 1987.

[Moz90] I. Mozetic. Abstractions in model-based diagnosis. In *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, pages 64–75. AAAI, 1990.

[Pla80] D.A. Plaisted. Abstraction mappings in mechanical theorem proving. In *5th Conference on Automated Deduction*, pages 264–280, 1980.

[Pla90] D. Plaisted. Mechanical Theorem Proving. In *Formal Techniques in Artificial Intelligence.* Elsevier Science Publishers B.V., 1990.

[Pra65] D. Prawitz. *Natural Deduction - A proof theoretical study.* Almquist and Wiksell, Stockholm, 1965.

[Sac74] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Seb93] R. Sebastiani. Astrazione: dalla Teoria alla Realizzazione di un Abstract Proof Checker. *AI*IA Notizie*, 2:41–53, 1993.

[Ten88] J.D. Tenenberg. *Abstraction in Planning.* PhD thesis, Computer Science Department, University of Rochster, 1988. Also TR 250.

[Wey80] R.W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artificial Intelligence*, 13(1):133–176, 1980.

[Zor93] A. Zorer. Definizione ed implementazione di un sistema formale per la meccanizzazione del ragionamento integrato su fatti ed obiettivi. Technical Report 9310-15, IRST, Trento, Italy, 1993. IRST-Thesis.