ISTITUTO PER LA RICERCA SCIENTIFICA E TECNOLOGICA
I 38100 TRENTO – LOC. PANTÉ DI POVO – TEL. 0461–814444
TELEX 400874 ITCRST – TELEFAX 0461–810851

A GENERAL PURPOSE REASONER FOR ABSTRACTION

Fausto Giunchiglia
Roberto Sebastiani
Adolfo Villafiorita
Toby Walsh

January 1993
Technical Report # 9301-08

Publication Notes: Also DIST-Technical Report 93-0016.

ISTITUTO TRENTINO DI CULTURA

A General Purpose Reasoner for Abstraction*

Fausto Giunchiglia[1,2] Roberto Sebastiani[3] Adolfo Villafiorita[3,4] Toby Walsh[1]

[1] I.R.S.T., 38050 Povo, Trento, Italy.

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

[3] D.I.S.T., viale Causa 13, 16146 Genoa, Italy

[4] University of Ancona, via Brecce Bianche, 60131 Ancona, Italy

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

tel: ++39.(0)461.314359

fax: ++39.(0)461.314591

Content areas: abstraction,
automated reasoning, interactive theorem proving

Abstract

The goal of the work described in this paper is the development of a system, called ABSFOL, which allows the user to state declaratively abstractions and to use them according to the desired control strategy. ABSFOL has been successfully tested on many examples. So far we have failed to find an interesting abstraction whose implementation requires a major programming effort.

*Alan Bundy provided much of the inspiration for the solution of the Goedel example. The authors' names are purely alphabetical.

1 Introduction

By "reasoning with abstraction" we mean the process by which, starting from a representation of problem (called the "ground space"), we construct a new and simpler problem representation (called the "abstract space") and use it to help the solution of the original problem. This informal definition is very general and identifies as "abstraction" a large number of applications in areas like problem solving, planning, reasoning by analogy, learning, qualitative and model based reasoning, common sense reasoning, approximate reasoning and hardware and software verification. In [GW92b] we have developed a theory of abstraction and used it to capture all these different applications inside the same framework.

The goal of the work partially described in this paper is to develop a general reasoner with abstraction, called `ABSFOL`. [1] By "general" we mean that all the forms of abstraction must be definable and executable inside `ABSFOL`. Any user wanting to reason by abstraction will be able to state it declaratively without going through the pain of another re-implementation from first principles (*e.g.* the Lisp primitives). The definition and implementation of such a general reasoner presents many problems. The uniformity highlighted by the theory presented in [GW92b] hides a lot of differences which made all previous implementations very different. The following is a list of the possible different abstractions and uses of abstraction.

- Almost all abstractions preserve provability [GW92b, GW92a]. However provability can be preserved in many different ways [Wel91, GW92b].
- There are many different techniques to forget details and still preserve provability in the same way. As proved in [GW92b], one set of such techniques is: deleting preconditions in operators [Sac73], collapsing constants [Hob85], collapsing predicate or function symbols [Pla80, GW89, Ten88], dropping arguments to function or predicate symbols [Mel87].
- For each such technique there are many ways of deciding exactly which details to forget. For instance, [BGW91, Kno91] discuss various ways for automatically forgetting preconditions.
- Finally, there is a choice of how to intermix reasoning in the ground space and reasoning in the abstract space. For instance, in SOAR, abstraction is used only when the system is at an impasse [UR89].

[1] `ABSFOL` is implemented on top of `GETFOL` [Giu92], an interactive theorem prover which is a re-implementation/ extension of the `FOL` system [Wey80].

ABSFOL allows for all the possible abstractions and uses of abstractions listed above and has been successfully tested on many abstractions defined in the past (all those listed in [GW92b]). In section 2 we describe the main idea underlying ABSFOL, namely that abstraction is implemented as logic plus control. In section 3 we describe the logic of abstraction. This is done by showing, as an example, the proof of a statement closely related to Gödel's incompleteness theorem.

2 Abstraction = Logic + Control

We formalize abstraction as a mapping of a representation of a problem onto a new representation. An abstraction f from a ground space Σ_1 to an abstract space Σ_2, written $f : \Sigma_1 \Rightarrow \Sigma_2$, is defined as a triple $\langle \Sigma_1, \Sigma_2, f_\Lambda \rangle$ consisting of two axiomatic formal systems, Σ_1 and Σ_2 and a total function, f_Λ, which maps the language of Σ_1 onto that of Σ_2. (We use the words "space", "theory", "context" and "formal system" as synonyms).

In order to build a general reasoner for abstraction we must somehow make sure that the implementation does not depend upon assumptions which prevent any of the possibilities listed in the previous section. This is achieved by making a sharp distinction between *what* we do with abstraction and *how* we do it; that is, we make a clear distinction between the calculus and the control. The calculus defines all the forms of abstraction that can be used, whilst the control defines all the possible uses and control strategies. In this perspective, a particular algorithm for using abstraction becomes "algorithm = logic + control" [Kow79]. The situation is similar to that of first order logic or production rules. It is more complex because of the inherent complexity of reasoning with abstraction.

Let us start with the logic. We must have an operation of *abstraction declaration*, which allows to define an abstraction f as $f = \langle \Sigma_1, \Sigma_2, f_\Lambda \rangle$. An operational definition of f_Λ allows the construction of Σ_2 from Σ_1. We call the operation of generating Σ_2, *abstraction application*. In general we need to construct the deductive machinery and the axioms of Σ_2. In this paper, for lack of space, we consider only abstractions where Σ_2 has the same inference rules as Σ_1 and its axioms are mapped from those of Σ_1 via f_Λ. We shall prove goals by theorem proving. As we have ground and abstract goals we need to perform theorem proving in both spaces. We call these two operations *ground* and *abstract reasoning*. It must also be possible to make the reasoning in the two spaces interact, *i.e.* to make assertions in Σ_1 just because certain facts hold in Σ_2. We call this operation, *mapping back*. The information that is mapped back strongly influences the ground reasoning. The ground reasoning can be seen as the process of "refining"

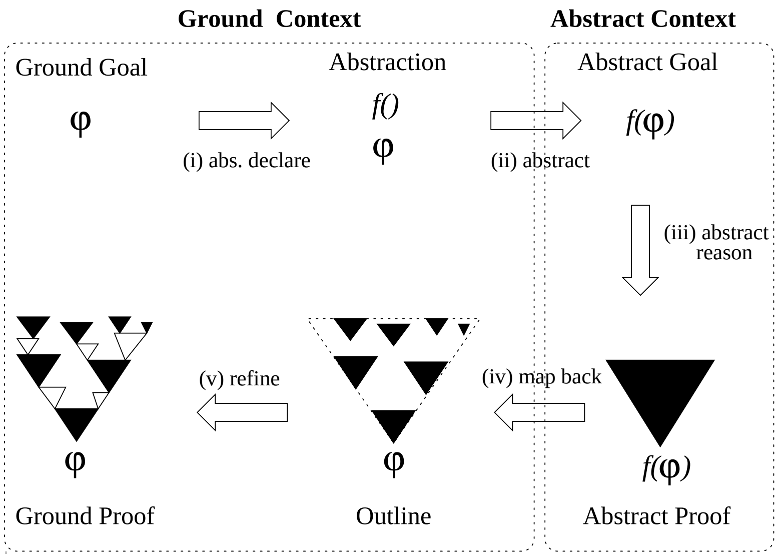

Figure 1: A calculus for abstraction.

the information obtained from the abstract space until the goal is proved. This is why we also call it, *refinement.* These five steps are graphically represented in Figure 1. The arrows describe the temporal order of activation in the (simplest) case of reasoning without bactracking. That is, we define the abstraction, we use it to build the abstract space, we prove the abstract goal which we then map back and refine until the original goal is proved.

However the simplicity of Figure 1 should not mislead the reader. Each of the five steps requires the application of many simpler operations. Thus, declaring an abstraction requires the declaration of the two spaces involved and of how the logic and the alphabet are translated. Generating Σ_2 means generating its alphabet, axioms and inference rules, the goal and often some of its theorems. Finding the abstract proof requires the application of the inference rules of the abstract space. The mapping back is particularly complicated as there are many possible strategies for extracting the information contained in an abstract proof. At one extreme only the abstract theorem is mapped back (as in *e.g.* [GG88]). In some cases no mapping back is performed at all, and the proof (or disproof) of the abstract theorem is taken as an approximation of the proof (disproof) of the ground theorem (as in *e.g.* [KS92]). At the opposite extreme, the entire abstract proof is mapped back (as in *e.g.* [Kno90]). There are also intermediate situations, for instance in [Plu87, GW89] only the partial order of unfoldings of definitions in the abstract proof is used. All these possible ways to perform the mapping back strongly influence the refinement as well. For instance the refinement of an abstract proof requires bridging the gaps among its nodes. This process is quite complicated to formalize as it involves reasoning from the middle out (see the example in section 3).

For each step we must therefore define a set of operations, *i.e.* a set of inference rules which define the calculus for that step. The logic of abstraction is the union of these five sub-calculi plus the prescription for when the inference rules of each calculus can be applied. One problem is that the inference rules which can be used depends on the abstraction and in particular on how provability is preserved across abstraction spaces. For instance, an abstract theorem in general cannot be used to assert its unabstraction as a ground theorem. It can only be taken as a "suggestion" of this fact. This is the case with TI abstractions. An abstraction $f = \langle \Sigma_1, \Sigma_2, f_\Lambda \rangle$ is TI iff

$$\vdash_{\Sigma_1} \varphi \implies \vdash_{\Sigma_2} f_\Lambda(\varphi) \quad \text{for any wff } \varphi \text{ in } \Sigma_1.$$

However with TD abstractions,

$$\vdash_{\Sigma_2} f_\Lambda(\varphi) \implies \vdash_{\Sigma_1} \varphi \quad \text{for any wff } \varphi \text{ in } \Sigma_1,$$

and the unabstraction of an abstract theorem can be directly asserted as a theorem. There is no need for refinement.

The control of abstract reasoning rises many problems as well. The key fact is that the temporal order suggested by Figure 1 is rarely followed in practice. For instance the abstract space may be generated only partially, this may require that we apply the rules of step (ii) after having done some abstract reasoning. It may happen that steps (iii) and (iv) are iterated. This kind of backtracking happens very often, *e.g.* any time we realize that we do not know how to refine the unabstraction of the abstract proof into a ground proof. We may iterate the entire process, *e.g.* perform step (i) (if we want to use a new abstraction) or step (ii) after step (v). In practice it is possible to envisage cases where the inference rules of the calculi of steps (i) - (v) are applied in any order.

The main consequence is that no ad hoc control strategy can be hardwired in ABSFOL. Our solution is to move from a completely automatic use of abstraction (as it has always been in the past) to one where abstraction is used interactively and the user can tell the system the exact sequence of operations to perform. ABSFOL is therefore built as an interactive system. However this is not an optimal solution as it still leaves the user with too much, often tedious, work to do. As is very common in interactive theorem proving (see *e.g.* [GMW79, Pau89, CAB$^+$86]), we propose a solution where ABSFOL is provided with an ML-like metalanguage for writing programs which implement search control strategies [HMM86, GMW79]. We call such programs, tactics. Automated theorem proving can then be implemented by writing complicated tactics. The user does not have to write the control strategy from scratch as the system is provided with a library of tactics implementing (some of) the most useful strategies. For a list of the advantages of this approach see [GMW79] and also [Bun88]. This issue is not further described in this paper.

Two observations are worth making. First, the implementation of a calculus and a control metalanguage for abstraction is quite complex and rises many theoretical and practical problems. The main point is one of correctness. This means not only that the underlying code does the right things but also that the user interface prevents the user from doing bad things, *e.g.* from asserting non-theorems. This implies a lot of code structuring and hiding that it is not necessary in ad hoc implementations. Second, the interactive use of abstraction, as well as allowing the construction of a general reasoner also has another important advantage. Despite some promising theoretical and experimental results (see for instance [Kor87, Kno91, GW91]) abstraction has proved, in automatic theorem proving at least, less useful than expected [GW91, Pla90]. The integration of user interaction and sophisticated heuristics can lessen some of the problems which have been found in the use of abstraction. For instance we think that it is very hard (if not impossible) to find a general heuristic which would automatically generate the abstraction used in section 3.

3 A calculus of abstraction

The goal of this section is to describe some of the details of the five sub-calculi of the logic of abstraction. We do this by describing the use of a TI abstraction according to the strategy where "*we first abstract the goal, we prove its abstracted version and then use the structure of the resulting proof as an outline (or plan) to help construct the proof of the original goal.* This is the hardest use of abstraction to provide a calculus for. It is also the most common use of abstraction in theorem proving, problem solving and planning" [GW92b].

We describe this use of abstraction via the proof of a theorem closely related to Gödel's First Incompleteness theorem. [2] This is an interesting example per se, for at least two reasons. First, the proof of this theorem is a subtle piece of mathematics well beyond the reach of current automatic theorem provers. Second, the proof shows how the interactive use of abstraction allows the construction of proofs which are much easier to understand and to explain. The precise statement of the goal is "$\exists F \; . \; valid(F) \; \wedge \; \neg provable(F)$". That is, there exists a

[2]The complete proof hinted in this paper can be found in [BGVW93].

formula F which is valid in the standard model of arithmetic but not provable in the standard axiomatization of Peano arithmetics (PA from now on). This problem is posed in a context called `maths` which knows of *e.g.* validity, provability, consistency of PA. Consider for instance the following sequence of `ABSFOL`'s commands.

```
noname:: NAMECONTEXT maths;
maths:: AXIOM m4:  forall F. provable(F) imp valid(F);
```

`Teletype` font is used to write input and output to `ABSFOL`. (Input and output have been slightly edited to make them more readable.) "`<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. `AXIOM` adds an axiom to the current context. Axiom `m4` says that any theorem of PA is true in the standard model of arithmetics. For simplicity, we assume that the Gödel numbering has been defined and that the diagonalisation lemma has been proved.

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

where `formno(F)` is the Gödel number of the formula `F`, `diag(x)` is the Gödel formula constructed by diagonalization (which asserts its own unprovability), `prf(Fn,Pn)` is true iff `Pn` is the Gödel number of a proof of the formula whose Gödel number is `Fn`. Although these assumptions simplify the theorem greatly, the proof is still long and complicated. Indeed, our experiments suggest that the proof is well beyond the reach of a state of the art resolution theorem prover like Otter [McC90].

3.1 Declaring an abstraction

We have provided `ABSFOL` with the following tools for defining abstractions: a language expressive enough to allow the definition of all the abstractions defined in the past (plus more), an interpreter for such a language which allows for the automatic (partial or total, see next section) generation of the abstract space, and a library of abstraction prototypes which can be easily instantiated to the desired application [Seb93]. At the moment the library contains all the most important abstractions defined in the past. The abstraction we consider here is adapted from a proposal by Alan Bundy. The main idea is to collapse the distinction between (the name of a) formula and its diagonalizing term. This is performed via the following many-to-one mapping on terms which collapses together terms which are semantically similar,

```
  diag(x), diag(el), all(x,diag(x)),
          ~prf(formno(diag(x),x)),    ⇒ ~d
         ~prf(formno(diag(x),el))

  prf(formno(diag(x),x)), prf(formno(diag(x),el))    ⇒  d
```

In ABSFOL this abstraction is defined with the following command

```
maths:: MAKECONTEXT absmaths;
maths:: ABSTRACTION goedel:maths => absmaths BY
 ...
 f(A and B)         := f(A) and f(B)
 f(A or B)          := f(A) or f(B)
 f(A imp B)         := f(A) imp f(B)
 f(A iff B)         := f(A) iff f(B)
 f(not A)           := not f(A)
 f(forall X.A)      := forall X. f(A)
 f(exists X.A)      := exists X. f(A)
 f(P(T1,...Tn))     := P(f(T1),...f(Tn))
 f(diag(T))         := ~d
 f(all(x,diag(x))   := ~d
 ...;
```

goedel is the name of the abstraction. MAKECONTEXT creates a new context of name its first argument. maths and absmaths are the names of the ground and the abstract spaces. The second line onwards define the abstraction. f_Λ is declaratively defined using a set of (terminating) rewrite rules. Notice that goedel preserves the logical structure of expressions and only abstracts atomic wffs (in particular, on terms). In fact goedel is an *atomic* abstraction and is TI [GW92b]. Notice that most (statistically more than ninety per cent) of the abstractions proposed in the past are atomic.

3.2 Applying an abstraction

A first option is to give ABSFOL the following command

```
maths:: ABSTRACT CONTEXT BY goedel;
```

This would cause the complete generation of the language, the axioms and the inference rules of absmaths. However it is often useful to build the abstract space only partially (*e.g.* to save time or to make the abstract search space smaller). ABSFOL provides commands which generate the components of the abstract space one by one, or by subsets.

```
maths:: ABSTRACT ALPHABET BY goedel;
maths:: ABSTRACT GOAL
  valid(all(x,diag(x))) and not provable(all(x,diag(x))) BY goedel;
  valid(~d) and not provable(~d) is the abstract goal.
maths:: ABSTRACT AXIOM diag BY goedel;
  adiag: provable(~d IFF ~d)
maths: ABSTRACT AXIOM m4 BY goedel;
  am4: forall F. provable(F) imp valid(F)
```

(The lines which follow ";" and are before the prompt are `ABSFOL`'s output). The first command completely generates the language of `absmaths`. The second abstracts the goal. The third and the fourth abstract two axioms. Notice that the argument of `provable` in `diag` collapses into a tautology (`diag` can therefore be forgotten), while `m4` translates unmodified.

3.3 Abstract reasoning

Step (iii) consists of "standard" theorem proving and can therefore be performed using `GETFOL`'s logic [Giu92]. Finding a proof in `absmaths` is much easier than finding a proof in `maths`. In fact, the abstract proof requires just 7 of the 22 abstract axioms and it does not require any of the complicated axioms like the diagonalization lemma. Indeed, experiments have shown that it is well within the reach of current resolution theorem provers. Otter, for example, is able to find it in 0.67 seconds on a Sun/4 generating just 183 clauses (of which 125 are kept).

The most important steps of the abstract proof are reported in Figure 2. The line numbers on the left are automatically associated by `GETFOL` to each new theorem. Each context has its own distinct numbering sequence. The numbers are used for future reference to the indexed theorems. The abstract proof divides naturally into two halves. In the first half we show that `~d` is unprovable (line `15`); in the second half, we show that `~d` is, however, valid (line `29`). Note that the abstract space is inconsistent; indeed at line `15` we demonstrate `not provable(~d)` whilst later on, at line `27`, we prove `provable(~d)`. In [GW93] we demonstrate that the fact that the abstract space might be inconsistent cannot be avoided with TI abstractions like `goedel`. However, provided we are careful not to exploit this inconsistency in our proofs, this does not cause problems [GW93]. For example, although it is able to determine the inconsistency of the abstract space with relative ease, Otter does not use this fact when proving the abstract goal.

Abstract Proof	**Ground Proof**
`1  provable(d)`	`1  provable(diag(x))`
`15 not provable(~d)`	`32 not provable(all(x,diag(x))`
`20 not provable(d)`	`38 not provable( prf(formno(diag(x),el))`
`27 provable(~d)`	`53 provable(diag(el))`
`29 valid(~d)`	`58 valid(all(x,diag(x))`
`30 valid(~d) and not provable(~d)`	`59 valid(all(x,diag(x))) and not provable(all(x,diag(x)))`

Figure 2: Outlines of the abstract and the ground proofs.

3.4 Mapping back and refinement

We want to use the abstract proof to help guide the construction of the ground proof. The idea is that the abstract proof should be shorter and easier to construct than the ground. It should also be "similar". These intuitions are formalized by a monotonicity relation between proof trees, called "*tree subsumption*" [GW92c]. Informally, we say that a tree Π_1 *subsumes* a tree Π_2, written "$\Pi_1 \subseteq \Pi_2$", iff the same wffs occur in Π_2 as in Π_1 with the same global ordering. Π_2 can therefore be obtained from Π_1 simply by adding nodes. The intuition is that the unabstraction of the abstract proof should subsume the ground proof. In this case we say that it is an *outline*. Outlines can be refined simply by adding nodes. In general we have to perform more complex operations, *e.g.* deleting nodes. However this is not the case for a very large class of abstractions including atomic abstractions [GW92c].

An outline of the ground proof which results from the mapping back and the refinement of the abstract proof is reported in Figure 2. The line numbers show that the ground proof is much longer than the abstract proof. They also show that the two proofs have the same global structure (if a node is above/ below/ adjacent another in the abstract proof, the same holds for their unabstractions in the ground proof).

The implementation of the calculi for the mapping back and the refinement rises some non trivial problems. First, from the abstract proof we can only build a parametric outline of the ground proof. This outline is not a proof in the ground space since it contains parameters which represent choices in unabstracting the (many-to-one) mapping function. In the example, the user is called upon in to pick unabstractions for `d` and `~d`. Thus, the abstract formula `~d` which is valid but not provable (lines `15`, `29` and `30` of the abstract proof) corresponds to the ground formula `all(x,diag(x))`. By comparison, the abstract formula `~d` at line `27`, which is provable and from which we derive the validity of `~d` at line `29`, corresponds to the ground formula `diag(el)` (where `el` is a random element of the standard model of PA). Second, the unabstraction of the abstract proof is not a ground proof since the abstract proof only contains deductions corresponding to the *key* steps in the ground proof. For instance some of the ground inferences between steps `53` (unabstraction of line `27`) and `58` (unabstraction of line `29`) translate `diag(el)` into `all(x,diag(x))`. Finally, we need to add a new datatype to represent the steps in an outline since they do not have the same status as steps in a proof. The steps in an outline may not be derivable as mapping back is not guaranteed. They are merely conjectures. To represent such steps we therefore introduce a new datatype called *try*. A try may contain parameters (which we will instantiate) and may not follow immediately from earlier tries.

`ABSFOL` allows for mapping back the whole of an abstract proof into a complete outline.

```
absmaths:: MAPBACK ALL BY goedel;
```

However, let us consider, as an example, the pointwise mapping back and refinement of the last two lines of the abstract proof.

```
absmaths:: SHOW PROOF;
...
  30   valid(~ d) and not provable(~ d)
  31   exists F. (valid(F) and not provable(F))
```

We first map lines `30` and `31` back to ground conjectures.

```
absmaths:: MAPBACK 31 BY goedel;
  31.0 exists F. (valid(F) and not provable(F))
absmaths:: MAPBACK 30 BY goedel;
  30.0 valid(~ d) and not provable(~ d)
```

A try, like a proof line, is numbered. In this case, the tries are numbered `30.0` and `31.0`. To add extra steps between these tries, `ABSFOL` would use line numbers of the form `30.n`. Note that `31.0` is the unique unabstraction of line `31` in the abstract proof. It represents the conjecture that we can prove the existence of a valid but unprovable formula. Line `30`, by comparison, has several possible unabstractions. The formula displayed for try `30.0` is meant to represent one of a set of ground formulae which abstract onto line `30`. In refining the outline, we must chose an instantiation for this parameter. In this case, we decide to instantiate it to the term, `all(x,diag(x))`.

```
maths:: INSTT 30.0 ~ d:all(x,diag(x)) BY goedel;
  30.0 valid(all(x,diag(x))) and not provable(all(x,diag(x)))
```

Try 31.0 now follows immediately from 30.0. To show this we perform an existential introduction on 30.0, and match the result of this existential introduction with 31.0.

```
maths:: TRYEXISTI 30.0 all(x,diag(x)):F;
  30.1  exists F. (valid(F) and not provable(F))
maths:: MATCHTRY 30.1 31.0;
  30.1 has been bound to 31.0
```

When one try does not follow immediately from previous tries, refinement also consists of adding in extra tries.

4 Conclusions

There are many different abstractions and uses of abstractions. However, we have shown that it is possible to envisage a general, abstraction independent, reasoner with abstraction called ABSFOL. The definition of this reasoner is based on the distinction between logic and control and on the interactive use of abstraction. Using ABSFOL avoids implementing abstraction from scratch. It also suggests a new use of abstraction where interactivity can be exploited to overcome some of the traditional problems with using abstraction.

References

[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.

[Bun88] A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In R. Luck and R. Overbeek, editors, *Proc. of the 9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available as DAI Research Paper No. 349, Dept. of Artificial Intelligence, Edinburgh.

[CAB$^+$86] R.L. Constable, S.F. Allen, H.M. Bromley, et al. *Implementing Mathematics with the NuPRL Proof Development System.* Prentice Hall, 1986.

[GG88] F. Giunchiglia and E. Giunchiglia. Building complex derived inference rules: a decider for the class of prenex universal-existential formulas. In *Proc. 7th European Conference on Artificial Intelligence*, pages 607–609, 1988. Extended version available as DAI Research Paper 359, Dept. of Artificial Intelligence, Edinburgh.

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

[GMW79] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanized logic of computation*, volume 78 of *Lecture Notes in Computer Science.* Springer Verlag, 1979.

[GW89] F. Giunchiglia and T. Walsh. Theorem Proving with Definitions. In *Proc. of the 7th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, pages 175–183, 1989. Also IRST-Technical Report 8901-03 and DAI Research Paper No 429, University of Edinburgh.

[GW91] F. Giunchiglia and T. Walsh. Using abstraction. In *Proc. of the 8th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, Leeds, UK, 1991. Also IRST-Technical Report 9010-08 and DAI Research Paper 515, University of Edinburgh.

[GW92a] F. Giunchiglia and T. Walsh. Theories of Abstraction: a Historical Perspective. In *Proc. AAAI Workshop on Approximation and Abstraction of Computational Theories*, San Jose', CA, 1992. Also IRST-Technical Report 9206-23, IRST, Trento, Italy.

[GW92b] 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.

[GW92c] 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.

[GW93] F. Giunchiglia and T. Walsh. The inevitability of inconsistent abstract spaces. *Journal of Automated Reasoning*, 11:23–41, 1993. Also IRST-Technical Report 9006-16, IRST, Trento, Italy.

[HMM86] R. Harper, D. McQueen, and Robin Milner. Standard ML. LFCS report series ECS-LFCS-86-2, Laboratory for Foundations of Computer Science, Dept. of Computer Science, University of Edinburgh, 1986.

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

[Kno90] C. A. Knoblock. Abstracting the Tower of Hanoi. In *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, pages 13–23. AAAI, 1990.

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

[Kor87] R.E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[Kow79] R. Kowalski. Algorithm = logic + control. *Communications of the ACM*, 22(7):424–436, 1979.

[KS92] H. Kautz and B. Selman. Forming Concepts for Fast Inference. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 786–793, 1992.

[McC90] W. W. McCune. Otter 2.0 users guide. Technical Report ANL-90/9, Maths and CS. Division, Argonne National Laboratory, Argonne, Illinois, 1990.

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

[Pau89] L. Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5:363–396, 1989.

[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.

[Plu87] D. Plummer. *Gazing: Controlling the Use of Rewrite Rules.* PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1987.

[Sac73] E.D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. In *Proceedings of the 3rd International Joint conference on Artificial Intelligence*, 1973.

[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.

[UR89] A. Unruh and P. Rosenbloom. Abstraction in problem solving and learning. In *Proc. of the 11th International Joint Conference on Artificial Intelligence*, pages 681–687, 1989.

[Wel91] D.S. Weld. Reasoning about model accuracy. Technical Report 91-05-02, University of Washington, Dept. Computer Science and Engineering, 1991.

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