Approximation and Abstraction Techniques for Generating Concise Answers to Database Queries

Thomas Ellman
Department of Computer Science
Rutgers University
ellman@cs.rutgers.edu

Abstract

Database queries can return voluminous answers that are time consuming to read and that provide information with too much detail. A solution to this problem is presented in this paper. The solution utilizes *approximate domain abstraction* to construct a hierarchy of databases. Each successive database utilizes a more abstract domain language than the previous one. Each successive database is also an approximation of the previous one. Users may interact with the system to gradually refine concise, abstract, approximate answers into detailed, lengthy, exact ones. Users may also utilize the system to construct concise hierarchically structured answers to database queries. Implementation of this method for datalog databases is in progress. Expected results include empirical tests of the method on real world databases.

Motivation

The problem of voluminous answers is illustrated in Figure 1. A user seeks information about students taking math courses. The system returns a long list of all pairs (S, C) such that S is taking C and C is a math course. The answer takes a long time to read. It also may provide more detail than the user desires. The user may not have wanted to know the individual course and student pairs. He may actually have wanted only a general characterization of types of students and the types of math courses they take.

A more useful interaction is shown in Figure 2. The system initially responds with an answer stating that science majors take introductory math courses. The user is expected to take this answer to mean that all science majors take all introductory math courses, and no other students take any math courses. The answer is "abstract" because it is stated in terms of categories like "science-major" or "introductory-math", rather than individual students our courses. The answer is also approximate, since not all science majors take all introductory math courses, and in addition, other kinds

```
?- take(S,C),math(C).

S=''Tom Jones'', C=''Calculus'';
S=''Susan Green'', C=''Linear Algebra'';
S=''John Black'', C=''Dynamical Systems'';
S=''Jane Davis'', C=''Abstract Algebra'';
S=''Tony Gray'', C=''Precalculus'';
...
S=''Karl Gauss'', C=''Math Anxiety Clinic'';
No.
```

Figure 1: Voluminous Detailed Answer

of students take other kinds of courses. If the user desires a less abstract or less approximate answer, he may request the system to refine the answer. The system then utilizes the student and course hierarchies shown in Figures 3 and 4 to gradually refine the approximate, abstract initial answer into an exact, detailed answer.

The user may find a "hierarchically structured" answer to be even more useful. An answer of this type is shown in Figure 5. The answer has been refined down to the third level of the hierarchy. The answer is expressed in a nested format. Each level of nesting corresponds to one level of refinement. Each level of refinement is expressed as a modification of the previous level, rather than as simple list of pairs. The hierarchically structured answer in Figure 5 thus provides exactly the same information as the third refinement in Figure 2; however, the hierarchically structured answer is more concise.

Approach

We are developing a technique called "approximate domain abstraction" combining ideas on domain abstraction in databases from (Imielinski 1987) with ideas on approximate symmetry in constraint satisfaction from (Ellman 1993a) and (Ellman 1993b). Approximate domain abstraction operates by collecting the constant

```
?- take(S,C),math(C).

S=science-major, C=introductory-math;

No More. Refine? Yes

S=math-science-major, C=calculus;
S=math-science-major, C=linear-algebra;
S=math-science-major, C=upper-math;
S=non-math-science-major, C=calculus;
S=humanities-major, C=remedial-math;

No More. Refine? Yes

S=math-major, C=calculus;
S=math-major, C=linear-algebra;
S=math-major, C=upper-applied-math;
S=math-major, C=upper-pure-math;
S=physics-major, C=calculus;
S=physics-major, C=linear-algebra;
S=physics-major, C=upper-applied-math;
S=biology-major, C=calculus;
S=chemistry-major, C=calculus;
S=economics-major, C=calculus;
S=history-major, C=precalculus;
S=english-major, C=math-anxiety-clinic;

No More. Refine? No
```

Figure 2: Hierarchy of Answers

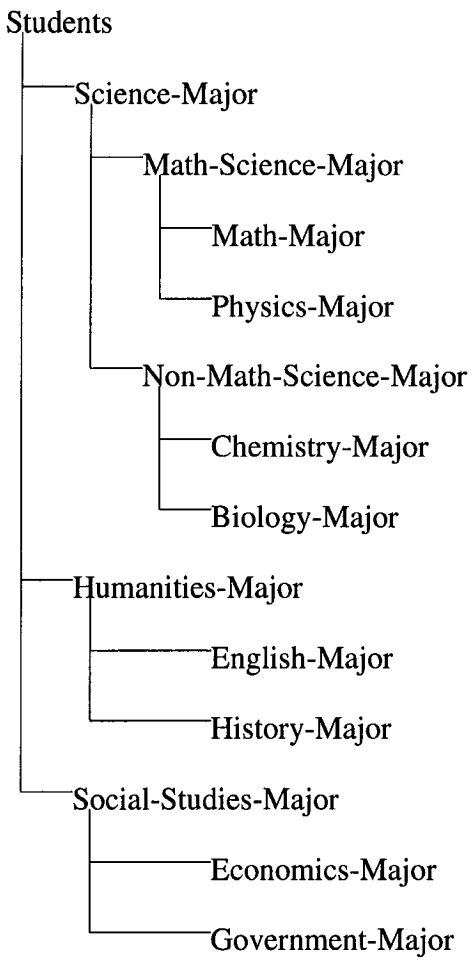

Figure 3: Hierarchy of Students by Major

symbols of a database into "equivalence" classes. Objects are grouped in common classes if they have identical, or nearly identical properties, according to the information in the database. The initial database is then transformed into an abstract database. The abstract database describes properties and relations in terms of the equivalence classes rather than the original constant symbols. Queries to the system are first processed using the abstract database generating an abstract answer. If the user so desires, the answer is refined into a concrete answer using initial database.

Our approach will work best when the facts in database have a particular kind of regularity, i.e., when the database contains lots of individuals that are equivalent, or nearly equivalent in terms of the information in the database. Queries to such a database will return answers that are approximately equal to cartesian products, or unions of small numbers of cartesian products. A cartesian product $A \times B$ can be expressed concisely, provided the factors A and B can be identified. Our approach may be seen as a method of identifying regularities in the database that allow approximate answers to be constructed in terms of factored cartesian products.

Defining Approximate Domain Abstraction

Approximate domain abstraction is defined formally in Figure 6. The input is a datalog knowledge base K consisting of a set L of positive ground literals and a set R of horn rules with no function symbols. The input also includes a description of the domains $\{D_1, \ldots, D_n\}$ on which the predicates appearing in L are defined. The output is a generate series of databases $K^1 \ldots K^H$, each of which differs from the previous one in the following ways: The domains $\{D_1^l, \ldots D_n^l\}$ for the database at level l are abstractions of the domains $\{D_1^{l-1}, \ldots D_n^{l-1}\}$ for the database at level $l-1$. In particular, each member of D_i^l is a union of members of D_i^{l-1}. In addition, the ground literals L^l for level l refer only to constants

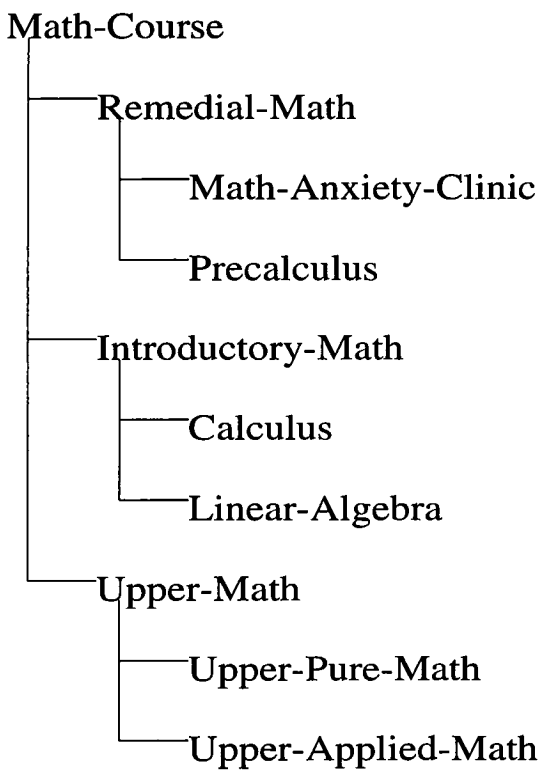

Figure 4: Hierarchy of Courses by Topic

in the domains for level l.

Approximate domain abstraction is defined in Figure 6 as a relation between whole knowledge bases, K and K^l, rather than a mapping from one, concrete language into another, abstract language. as is done in (Giunchiglia and Walsh 1992). For this reason, some additional mechanism is needed to define the semantics the abstract knowledge base. Two different interpretations of an abstract sentence $P(\bar{a})$ are proposed in (Imielinski 1987): The "universal" interpretation says that P is true of *every* concrete tuple $\bar{r}$ of which $\bar{a}$ is an abstraction. The "existential" interpretation says that P is true of *some* concrete tuple $\bar{r}$ of which $\bar{a}$ is an abstraction. In this work, we are adopting the universal interpretation of abstract knowledge bases.

A domain abstraction will be TI "Theorem Increasing", TD "Theorem Decreasing" or TC "Theorem Constant", or TX "Theorem Mixed" ((Giunchiglia and Walsh 1992)) at a given level l of the abstraction hierarchy depending on how the abstract ground facts at level l are related to the ground facts in the initial database.

- TI if $(\forall P \epsilon Predicates(L))(\forall \bar{a} \epsilon Tuples(P,l))$: $(P(\bar{a}) \epsilon L^l) \Leftarrow (\forall \bar{r} \epsilon Refinements(\bar{a},P,0))P(\bar{r}) \epsilon L$
- TD if $(\forall P \epsilon Predicates(L))(\forall \bar{a} \epsilon Tuples(P,l))$: $(P(\bar{a}) \epsilon L^l) \Rightarrow (\forall \bar{r} \epsilon Refinements(\bar{a},P,0))P(\bar{r}) \epsilon L$
- TC if $(\forall P \epsilon Predicates(L))(\forall \bar{a} \epsilon Tuples(P,l))$: $(P(\bar{a}) \epsilon L^l) \Leftrightarrow (\forall \bar{r} \epsilon Refinements(\bar{a},P,0))P(\bar{r}) \epsilon L$
- TX if None of the above.

(Definitions of terms are shown in Figure 7.) The proof of this observation is straightforward. The TI case requires showing that the abstraction process transforms valid concrete resolution proofs over K into valid abstract resolution proofs over K^l. The TD case requires showing that any valid abstract proof over K^l can be refined into a valid concrete proof over K. The TC case follows from the other two. These proofs are similar, but not identical, to proofs presented in (Plaisted 1981).

```
?- take(S,C),math(C).

S=science-major, C=introductory-math;

 Minus:
  S=non-math-science-major, C=linear-algebra;
 Plus:
  S=math-science-major, C=upper-math;
  S=humanities-major, C=remedial-math;

  Minus:
   S=physics-major, C=upper-pure-math;
   S=english-major, C=precalculus;
   S=history-major, C=math-anxiety-clinic;
  Plus:
   S=economics-major, C=calculus;
```

Figure 5: Hierarchically Structured Answer

Constructing an Approximate Domain Abstraction Hierarchy

An algorithm for constructing an approximate domain abstraction hierarchy is shown in Figure 8. (Definitions of terms appearing in the algorithm are shown in Figure 7.) The algorithm takes as input the ground literals L and the rules R of the initial database. The input also includes a flag $STRATEGY$ which may be TC, TI, TD or TX indicating the type of abstraction the system should build. The algorithm operates by repeatedly selecting a domain to be abstracted and then merging two elements of that domain. At each level of the hierarchy, the algorithm maintains an array $N_P^l(\bar{a})$. This array indicates for each predicate P and each abstract tuple $\bar{a}$ in the abstract domain of P, the number of refinements of $\bar{a}$ that satisfy P in the initial

Given: A datalog knowledge base $K = L \cup R$:

- A set L of positive ground literals.
- A set R of horn rules with no constant or function symbols.
- The set $Predicates(L) = \{P_1, \ldots P_m\}$ of predicates appearing in L.
- The set $Domains(L) = \{D_1, \ldots D_n\}$ on which the predicates in L are defined.
- For each predicate $P \epsilon Predicates(L)$, the arity $A(P)$ of P, and for each $i \epsilon [1 \ldots A(P)]$ the index $I(P,i)$ of the domain $D_{I(P,i)}$ of the i-th argument of P.

Find: A series of abstract knowledge bases, $K^1, \ldots, K^H$, each defined by:

- A set $Domains(L^l) = \{D_1^l, \ldots D_n^l\}$ of abstract domains, such that:

  For all $(i \epsilon [1 \ldots n])$, D_i^l is a partition of D_i

  For all $(i \epsilon [1 \ldots n])$, each member of D_i^l is a union of members of D_i^{l-1}.

- A set L^l of positive ground literals such that for each $P(\bar{a}) \epsilon L^l$:

  $P \epsilon Predicates(L)$.

  $\bar{a} = (a_1, \ldots, a_{A(P)})$.

  For each $i \epsilon [1 \ldots A(P)], a_i \epsilon D_{I(P,i)}^l$

- A set R^l of rules such that $R^l = R$.

Figure 6: Approximate Domain Abstraction Definition

database. This quantity is used to decide whether $P(\bar{a})$ will be placed in the abstract database at level l. For example, when constructing TX abstractions, $P(\bar{a})$ will be placed in the abstract database whenever $N_P^l(\bar{a})$ is greater than half the total number of refinements of $\bar{a}$.

An additional mechanism is needed to construct abstractions of databases containing references to interpreted predicates, e.g., predicates implementing arithmetic or relational operations. These primitives can be treated as if they were represented extensionally by sets of ground literals. To properly handle interpreted predicate P, it suffices to implement an abstract interpreted predicate P^l at each level of the hierarchy. The code for P^l must arrange that P^l will succeed or fail according to the rules in the case statement of the algorithm in Figure 8.

- $Tuples(P) = D_{I(P,1)} \times \ldots \times D_{I(P,A(P))}$: The set of tuples over which the predicate P is defined.
- $Tuples(P,l) = D_{I(P,1)}^l \times \ldots \times D_{I(P,A(P))}^l$: The l-th level abstractions of $Tuples(P)$.
- $Refinements(\bar{a}, P, l)$: The set of all $\bar{r}$ in $Tuples(P,l)$ of which $\bar{a}$ is an abstraction.
- $B_P^l : Tuples(P,l) \rightarrow \{T, F\}$: $B_P^l(\bar{a})$ is a boolean value indicating whether tuple $\bar{a}$ will appear in the extension of ground predicate P in the l-th level database.
- $N_P^l : Tuples(P,l) \rightarrow Integers$: $N_P^l(\bar{a})$ is the number of tuples in $Refinements(\bar{a}, P, 0)$ that appear in the extension of the ground predicate P in the initial database.
- $C_P^l : Tuples(P,l) \rightarrow Integers$: $C_P^l(\bar{a})$ is the smallest number of additions or deletions to the extension of P in the initial database needed to create a new database in which either all of $Refinements(\bar{a}, P, 0)$ are included in the extension of the ground predicate P, or none of $Refinements(\bar{a}, P, 0)$ are included in the extension of the ground predicate P. (A positive number indicates additions. A negative number indicates deletions).

Figure 7: Definitions

The algorithm in Figure 8 is non-deterministic. It includes the non-deterministic steps of choosing a domain to be abstracted and choosing elements from that domain to merge. The algorithm thus requires a control strategy. We plan to use a heuristic that attempts to minimize the amount by which the abstract databases deviate from TC abstractions. We define the error of the abstract knowledge base $K^l = L^l \cup R^l$ with respect to the initial, concrete knowledge base $K = L \cup R$ to be the minimum number of additions or deletions of positive ground literals needed to convert K into a new knowledge based K' such that K^l is a TC abstraction of K'. The error can be determined entirely from the abstract ground literals L^l and the initial, concrete ground literals L:

$$E(L^l, L) = \Sigma_{P \epsilon Predicates(L)} \{ \Sigma_{\bar{a} \epsilon Tuples(P,l)} \ |C_P^l(\bar{a})| \}$$

(The quantity $|C_P^l(\bar{a})|$ is the number of changes associated with predicate P and tuple $\bar{a}$. This quantity is computed by the algorithm in Figure8.) When constructing the database at level l the algorithm must choose some domain D_i^{l-1} to be abstracted and some pair e_a and e_b of elements of D_i^{l-1} to be merged. These

Abstract(L,R,Strategy,H):

1. Initialize each D_i^0 to be a collection of singleton sets.
2. Initialize each $B_P^0(\bar{x})$ to be T if $P(\bar{x})$ else F.
3. Initialize each $N_P^0(\bar{x})$ to be 1 if $P(\bar{x})$ else 0.
4. Initialize each $C_P^0(\bar{x})$ to be 0.
5. For $(l = 1 \ldots H)$ do:
   (a) Define domains $D_1^l, \ldots, D_n^l$ for level l:
      i. Choose some domain D_i^{l-1} to be abstracted.
      ii. Choose two elements e_a and e_b of D_i^{l-1}.
      iii. Define $e_c \equiv e_a \cup e_b$.
      iv. $D_i^l \leftarrow (D_i^{l-1} - \{e_a, e_b\}) \cup \{e_c\}$.
      v. For $(j = 1 \ldots n, j \neq i)$ do $D_j^l \leftarrow D_j^{l-1}$.
   (b) For each $P \epsilon Predicates(L)$ and $\bar{a} \epsilon Tuples(P, l)$ do:
      i. Let $R = Refinements(\bar{a}, P, l-1)$.
      ii. $N_P^l(\bar{a}) \leftarrow \Sigma_{\bar{r} \epsilon R} N_P^{l-1}(\bar{r})$.
      iii. Let $S = |Refinements(\bar{a}, P, 0)|$.
      iv. Case(Strategy):
      TC: If $N_P^l(\bar{a}) = 0$:$B_P^l(\bar{a}) \leftarrow F; C_P^l(\bar{a}) \leftarrow 0$
      If $N_P^l(\bar{a}) = S$:$B_P^l(\bar{a}) \leftarrow T; C_P^l(\bar{a}) \leftarrow 0$
      If $0 < N_P^l(\bar{a}) < S$:$Fail$
      TI: If $N_P^l(\bar{a}) = 0$:$B_P^l(\bar{a}) \leftarrow F; C_P^l(\bar{a}) \leftarrow 0$
      If $N_P^l(\bar{a}) > 0$:$B_P^l(\bar{a}) \leftarrow T; C_P^l(\bar{a}) \leftarrow S - N_P^l(\bar{a})$
      TD: If $N_P^l(\bar{a}) = S$:$B_P^l(\bar{a}) \leftarrow T; C_P^l(\bar{a}) \leftarrow 0$
      If $N_P^l(\bar{a}) < S$:$B_P^l(\bar{a}) \leftarrow F; C_P^l(\bar{a}) \leftarrow -N_P^l(\bar{a})$
      TX: If $N_P^l(\bar{a}) > S/2$:$B_P^l(\bar{a}) \leftarrow T; C_P^l(\bar{a}) \leftarrow S - N_P^l(\bar{a})$
      If $N_P^l(\bar{a}) \leq S/2$:$B_P^l(\bar{a}) \leftarrow F; C_P^l(\bar{a}) \leftarrow -N_P^l(\bar{a})$
   (c) Define the abstract database $K^l = L^l \cup R^l$:
      i. $L^l \leftarrow \{P(\bar{a}) \mid B_P^l(\bar{a})\}$
      ii. $R^l \leftarrow R$.

Figure 8: Constructing a Hierarchy of Databases

choices must be made in a manner that makes the error $E(L^l, L)$ as small as possible.

The algorithm for constructing a hierarchy of databases can be implemented in $O(NM[D^{A+1}])$ where N is the number of domains, M is the number of relations D is the size of the largest domain, and A is the highest arity of any relation. (This efficiency depends on noticing that most quantities don't change when moving from level to level.) The algorithm uses space $O(H|L|)$, where H is the height of the hierarchy, and $|L|$ is the number of ground facts in the initial database. (This efficiency depends on using sparse array implementations of arrays indexed over cartesian products of domains.)

$$
\begin{array}{llll}
A_n & & & \\
- & (A_n - A_{n-1}) & \cup & (A_{n-1} - A_n) \\
- & (A_{n-1} - A_{n-2}) & \cup & (A_{n-2} - A_{n-1}) \\
 & \ldots & & \\
- & (A_{n-i} - A_{n-(i+1)}) & \cup & (A_{n-(i+1)} - A_{n-i}) \\
 & \ldots & & \\
- & (A_1 - A_0) & \cup & (A_0 - A_1)
\end{array}
$$

Figure 9: Hierarchically Structured Answer Schema

Generation of Answers

Hierarchically structured answers can be constructed in the following fashion. A general schema for such answers is shown in Figure 9. Given a query $Q(\bar{x})$, the system begins by processing the query at each level of the abstraction hierarchy, obtaining an answer A^l for each level. The system then fills out the schema by computing the set differences between answers at successive pairs of levels. Proceeding downward from the highest level to the lowest, the system deletes tuples erroneously included at the previous level, and then adds adds tuples erroneously left out at the previous level. The set differences can be computed efficiently because the answer A^l at each level is represented in terms of abstract tuples, each of which stands for a whole set of concrete tuples.

The abstract answers in shown in Figures 2 and 5 are potentially misleading to the reader. In particular these answers refer to semantically laden symbols such as "science-major" or "introductory-math" to describe classes of constants in the initial database. Unfortunately, when equivalence classes are constructed automatically, no such meaningful symbols will be available. To begin with, notice that the system will still be useful for generating concise answers. In particular, consider the answer shown in Figure 10. This answer is more concise than a direct listing of all pairs in the cartesian product. Nevertheless, a better solution requires a user friendly language for describing equivalence classes generated by the system in the course of constructing abstract databases. One approach would modify the hierarchy construction algorithm to construct only such equivalence classes as can be described by queries in a query language known to the user. Each equivalence class could then be described to the user as the set of objects that satisfy a query in a language he understands, as shown in Figure 11.

Speedup of Query Processing

Abstraction hierarchies generated by using the *TI* (Theorem Increasing) strategy in the algorithm in Fig-

```
?- take(S,C),math(C).

{(S,C) |
  S in {``Tom Jones'',``Susan Green'',...}
  C in {``Calculus'',``Linear Algebra'',...}}
```

Figure 10: Cartesian Product of Extensionally Described Classes

```
?- take(S,C),math(C).

{(S,C) | Q1(S)} and Q2(C)}

Q1(X) = major(X,Y),science(Y)

Q2(Y) = topic(Y,math),level(Y,introduction)
```

Figure 11: Cartesian Product of Intensionally Described Classes

ure 8 can be used as a method of speeding up query processing, quite aside from any objectives of providing concise answers. Speedup is achieved by processing queries at each level of abstraction starting from the highest, and moving downward. The answer A^l generated at each level l will be a superset of the answer A^{l-1} generated at the next level. Query processing at each level $l-1$ can thus be required to explore only those paths of the search space that are guaranteed to lead to solutions that are refinements of A^l. Such constraints between levels of the hierarchy will enable the system to prune away large regions of the search space while working at upper levels of the hierarchy, without the larger cost of exploring those regions at the lowest level of the hierarchy.

Expected Results

We are using a synthetic domain of the sort show in Figures 1 though 5 as testbeds for developing the system. Once the system is implemented and debugged, we plan to use real world databases for experimental tests. Our approach will be evaluated according to the conciseness of the answers it generates, and their comprehensibility to human users.

Future Work

In the future, we plan to enhance our system in several ways. We plan constrain the algorithm to generate equivalence classes that can be expressed in terms of particular query languages, e.g., CLASSIC. We plan to use relevance reasoning (Levy and Sagiv 1993) to construct abstraction hierarchies that are suited to particular classes of queries. We plan to extend the system to handle general first order logic databases. We plan to approximate versions abstraction techniques other than domain abstraction.

Acknowledgements

The research reported in this paper has benefited from discussions with Alon Levy and Tomasz Imielinski, and has been supported in part, by the National Science Foundation (Grants IRI-9017121 and IRI-9021607).

References

T. Ellman. Abstraction via approximate symmetry. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, August 1993.

T. Ellman. Synthesis of abstraction hierarchies for constraint satisfaction by clustering approximately equivalent objects. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, 1993.

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

T. Imielinski. Domain abstraction and limited reasoning. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987.

A. Levy and Y. Sagiv. Exploiting irrelevance reasoning to guide problem solving. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

D. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16:47 – 108, 1981.