Selectively Instantiating Definitions

Matthew Bishop and Peter B. Andrews

Department of Mathematical Sciences, Carnegie Mellon University,
Pittsburgh, PA 15213, U.S.A.*
mbishop+@cs.cmu.edu, andrews+@cs.cmu.edu
http://www.cs.cmu.edu/~andrews/tps.html

Abstract. When searching for proofs of theorems which contain definitions, it is a significant problem to decide which instances of the definitions to instantiate. We describe a method called *dual instantiation*, which is a partial solution to the problem in the context of the connection method; the same solution may also be adaptable to other search procedures. Dual instantiation has been implemented in TPS, a theorem prover for classical type theory, and we provide some examples of theorems that have been proven using this method. Dual instantiation has the desirable properties that the search for a proof cannot possibly fail due to insufficient instantiation of definitions, and that the natural deduction proof which results from a successful search will contain no unnecessary instantiations of definitions. Furthermore, the time taken by a proof search using dual instantiation is in general comparable to the time taken by a search in which exactly the required instances of each definition have been instantiated. We also describe how this technique can be applied to the problem of instantiating set variables.

1 Introduction

When searching for proofs of theorems which contain definitions, it is a significant problem to decide which instances of the definitions to instantiate. Often, one needs to instantiate some, but not all, of them, and if one does instantiate all of them, one can cause the search space to expand in a very undesirable way. This problem has been noted in [4] and [23], and treatments of it may be found in [6], [8] and [12].

We have found a partial solution to this problem; it involves making each instance of a definition accessible to the search procedure in both its instantiated and its uninstantiated form, and letting the search procedure decide which to use, with a bias in favor of the uninstantiated form. This is very effective in some cases.

This solution to the problem seems particularly well suited to use with the connection method (also known as the matings procedure). We will discuss it

* This material is based upon work supported by the National Science Foundation under grant CCR-9624683. This work was supported in part by Intel Corporation through an equipment grant.

as it is implemented in TPS, an automated theorem prover for classical type theory developed at Carnegie Mellon University, based on the ideas in [3] and most recently described in [4].

2 Dual Instantiation

Well-formed formulae (wffs) in TPS are the formulae of Church's type theory[10]. A sentence in TPS is a wff of type o, where o is the type of truth values. Proofs in TPS are presented in the style of [2], and in wffs a dot stands for a left bracket whose mate is as far to the right as is consistent with the pairing of brackets already present. Quantifiers and negations have the smallest possible scope. Omitted brackets associated with the connectives $\wedge$, $\vee$, $\supset$, and $\equiv$ should be restored in that order, giving the connective the smallest possible scope in each case. Type symbols are omitted wherever it is possible to do so without ambiguity.

Any sentence in TPS may contain *abbreviations*, which are typed symbols such as $\subseteq_{o(o\alpha\alpha)(o\alpha\alpha)}$ or $\mathrm{TRANSITIVE}_{o(o\alpha\alpha)}$. Each abbreviation is associated with a *definition*[1] — in the case of $\subseteq$ (inclusion between binary relations), the definition is the wff $\lambda P_{o\alpha\alpha}\lambda R_{o\alpha\alpha}\forall x_\alpha \forall y_\alpha . Pxy \supset Rxy$. The process of replacing an abbreviation by its definition, and λ-normalizing the resulting expression if necessary, is referred to as *instantiating* the abbreviation.

Given such a sentence to be proven, we negate it, skolemize, and recursively construct an *expansion tree* [16]. So, for example, if the topmost symbol of the parse tree of the sentence is a conjunction, the expansion tree will consist of a conjunction node whose children are the expansion trees of the conjuncts.

In previous versions of TPS, the user had two options for instantiating abbreviations: either instantiate them all (*full instantiation*), or instantiate none of them. Uninstantiated abbreviations are regarded as constant symbols by the mating-search procedures. A similar choice was available for equalities, which could be instantiated by the Leibniz definition of equality ($x_\alpha = y_\alpha$ becomes $\forall q_{o\alpha} . q x_\alpha \supset q y_\alpha$), or left uninstantiated and treated as constant symbols. (Equalities between sets could also be instantiated using the extensional definition of equality, and those between truth values as equivalences, but for our purposes the essential distinction is between instantiated — by whichever definition — and uninstantiated equalities.)

In the current version of TPS, there is another choice, called dual instantiation. In this case, an atom A which contains an abbreviation is replaced by $A \wedge A'$ if A occurs positively, and by $A \vee A'$ if it occurs negatively, where A' is the result of instantiating the outermost abbreviation in A. A is left as a leaf of the expansion tree, and TPS continues to construct the expansion tree of A' exactly as it would for an ordinary instantiation.

When a vertical path diagram is constructed, as described in [1] and [3], the effect of this is to produce a conjunct, the uppermost literal of which is

[1] The classical terms for *abbreviation* and *definition* are *definiendum* and *definiens*, respectively.

simply A, and the rest of which is an instantiated version of A. Clearly, since $X \Leftrightarrow (X \wedge X) \Leftrightarrow (X \vee X)$, the truth or falsity of the sentence is unaffected by this.

As an example of the use of dual instantiation, consider the simple theorem, which we call THM136, asserting that the transitive closure of any relation is transitive:

$$\forall r_{o\alpha\alpha} \text{TRANSITIVE .TRANSITIVE-CLOSURE } r$$

where TRANSITIVE is defined as

$$\lambda r_{o\alpha\alpha} \forall a_\alpha \forall b_\alpha \forall c_\alpha . rab \wedge rbc \supset rac$$

and the transitive closure of r is defined to be the relation which relates x to y whenever x is related to y by every transitive relation p which includes r. Thus, the definition of TRANSITIVE-CLOSURE is:

$$\lambda r_{o\alpha\alpha} \lambda x_\alpha \lambda y_\alpha \forall p_{o\alpha\alpha} \, . \; r \subseteq p \;\wedge \text{TRANSITIVE } p \supset pxy$$

After instantiating TRANSITIVE in THM136, it becomes

$$\begin{aligned} \forall r_{o\alpha\alpha} \forall a_\alpha \forall b_\alpha \forall c_\alpha . \quad & \text{TRANSITIVE-CLOSURE } rab \\ & \wedge \text{TRANSITIVE-CLOSURE } rbc \\ & \supset \text{TRANSITIVE-CLOSURE } rac \end{aligned}$$

Instantiating TRANSITIVE-CLOSURE produces

$$\begin{aligned} \forall r_{o\alpha\alpha} \forall a_\alpha \forall b_\alpha \forall c_\alpha . \quad & \forall p^1_{o\alpha\alpha} [r \subseteq p^1 \wedge \text{TRANSITIVE } p^1 \supset p^1 ab] \\ & \wedge \forall p^2_{o\alpha\alpha} [r \subseteq p^2 \wedge \text{TRANSITIVE } p^2 \supset p^2 bc] \\ & \supset \forall p_{o\alpha\alpha} . r \subseteq p \wedge \text{TRANSITIVE } p \supset pac \end{aligned}$$

Negating and skolemizing the above, and putting the result into negation normal form, produces

$$\begin{aligned} & \forall p^1_{o\alpha\alpha} [\sim [R_{o\alpha\alpha} \subseteq p^1] \vee \; \sim \text{TRANSITIVE } p^1 \vee p^1 A_\alpha B_\alpha] \\ \wedge \; & \forall p^2_{o\alpha\alpha} [\sim [R_{o\alpha\alpha} \subseteq p^2] \vee \; \sim \text{TRANSITIVE } p^2 \vee p^2 B_\alpha C_\alpha] \\ & \wedge R_{o\alpha\alpha} \subseteq P_{o\alpha\alpha} \\ & \wedge \text{TRANSITIVE } P_{o\alpha\alpha} \\ & \wedge \sim P_{o\alpha\alpha} A_\alpha C_\alpha \end{aligned}$$

Instantiating the original theorem using full instantiation produces the vertical path diagram in Fig. 1, which clearly corresponds to the wff above. (Terms of the form $X^i p^j$ and $Y^i p^j$ are skolem terms, as are A, B and C, and the labels $\boxed{N}$ have been added for convenience in describing the mating below.)

To find a complete mating for this diagram, it is necessary to duplicate the x^2_α quantifier once and the x^1_α quantifier twice, producing the diagram in Fig. 2. The new copies of a literal $\boxed{N}$ created by duplication will be referred to as $\boxed{N'}$, $\boxed{N''}$ and so on; we will write connected pairs of literals as $\boxed{M - N}$. The complete mating is as follows: $\boxed{16-5}$, $\boxed{17-4}$, $\boxed{13-3}$, $\boxed{14-2}$, $\boxed{15-1}$, $\boxed{10-5'}$, $\boxed{11-4'}$, $\boxed{7-3'}$, $\boxed{8-2'}$, $\boxed{9-1'}$, $\boxed{12-3''}$, $\boxed{6-2''}$, $\boxed{18-1''}$.

$$
\left[\begin{array}{l}
\forall p^1_{o\alpha\alpha} \\
\quad \left[\left[\begin{array}{l} \boxed{16}\ R_{o\alpha\alpha}[X^0p^1].Y^0p^1 \\ \boxed{17}\ \sim p^1_{o\alpha\alpha}[X^0p^1].Y^0p^1 \end{array}\right] \vee \left[\begin{array}{l} \boxed{13}\ p^1_{o\alpha\alpha}[X^1p^1].Y^1p^1 \\ \boxed{14}\ p^1_{o\alpha\alpha}[Y^1p^1].Z^0p^1 \\ \boxed{15}\ \sim p^1_{o\alpha\alpha}[X^1p^1].Z^0p^1 \end{array}\right] \vee \boxed{12}\ p^1_{o\alpha\alpha}A_\alpha B_\alpha\right] \\
\quad \forall p^2_{o\alpha\alpha} \\
\quad \left[\left[\begin{array}{l} \boxed{10}\ R_{o\alpha\alpha}[X^2p^2].Y^2p^2 \\ \boxed{11}\ \sim p^2_{o\alpha\alpha}[X^2p^2].Y^2p^2 \end{array}\right] \vee \left[\begin{array}{l} \boxed{7}\ p^2_{o\alpha\alpha}[X^3p^2].Y^3p^2 \\ \boxed{8}\ p^2_{o\alpha\alpha}[Y^3p^2].Z^1p^2 \\ \boxed{9}\ \sim p^2_{o\alpha\alpha}[X^3p^2].Z^1p^2 \end{array}\right] \vee \boxed{6}\ p^2_{o\alpha\alpha}B_\alpha C_\alpha\right] \\
\qquad \forall x^2_\alpha \forall y^2_\alpha \\
\qquad \left[\boxed{5}\ \sim R_{o\alpha\alpha}x^2_\alpha y^2_\alpha \vee \boxed{4}\ P_{o\alpha\alpha}x^2_\alpha y^2_\alpha\right] \\
\qquad \forall x^1_\alpha \forall y^1_\alpha \forall z^1_\alpha \\
\qquad \left[\boxed{3}\ \sim P_{o\alpha\alpha}x^1_\alpha y^1_\alpha \vee \boxed{2}\ \sim P_{o\alpha\alpha}y^1_\alpha z^1_\alpha \vee \boxed{1}\ P_{o\alpha\alpha}x^1_\alpha z^1_\alpha\right] \\
\qquad \boxed{18}\ \sim P_{o\alpha\alpha}A_\alpha C_\alpha
\end{array}\right]
$$

Fig. 1. THM136 after full instantiation

$$
\left[\begin{array}{l}
\forall p^1_{o\alpha\alpha} \\
\quad \left[\left[\begin{array}{l} \boxed{16}\ R_{o\alpha\alpha}[X^0p^1].Y^0p^1 \\ \boxed{17}\ \sim p^1_{o\alpha\alpha}[X^0p^1].Y^0p^1 \end{array}\right] \vee \left[\begin{array}{l} \boxed{13}\ p^1_{o\alpha\alpha}[X^1p^1].Y^1p^1 \\ \boxed{14}\ p^1_{o\alpha\alpha}[Y^1p^1].Z^0p^1 \\ \boxed{15}\ \sim p^1_{o\alpha\alpha}[X^1p^1].Z^0p^1 \end{array}\right] \vee \boxed{12}\ p^1_{o\alpha\alpha}A_\alpha B_\alpha\right] \\
\quad \forall p^2_{o\alpha\alpha} \\
\quad \left[\left[\begin{array}{l} \boxed{10}\ R_{o\alpha\alpha}[X^2p^2].Y^2p^2 \\ \boxed{11}\ \sim p^2_{o\alpha\alpha}[X^2p^2].Y^2p^2 \end{array}\right] \vee \left[\begin{array}{l} \boxed{7}\ p^2_{o\alpha\alpha}[X^3p^2].Y^3p^2 \\ \boxed{8}\ p^2_{o\alpha\alpha}[Y^3p^2].Z^1p^2 \\ \boxed{9}\ \sim p^2_{o\alpha\alpha}[X^3p^2].Z^1p^2 \end{array}\right] \vee \boxed{6}\ p^2_{o\alpha\alpha}B_\alpha C_\alpha\right] \\
\qquad \forall x^2_\alpha \forall y^2_\alpha \\
\qquad \left[\boxed{5}\ \sim R_{o\alpha\alpha}x^2_\alpha y^2_\alpha \vee \boxed{4}\ P_{o\alpha\alpha}x^2_\alpha y^2_\alpha\right] \\
\qquad \forall x^{2}_\alpha{}' \forall y^{2}_\alpha{}' \\
\qquad \left[\boxed{5'}\ \sim R_{o\alpha\alpha}x^{2}_\alpha{}' y^{2}_\alpha{}' \vee \boxed{4'}\ P_{o\alpha\alpha}x^{2}_\alpha{}' y^{2}_\alpha{}'\right] \\
\qquad \forall x^1_\alpha \forall y^1_\alpha \forall z^1_\alpha \\
\qquad \left[\boxed{3}\ \sim P_{o\alpha\alpha}x^1_\alpha y^1_\alpha \vee \boxed{2}\ \sim P_{o\alpha\alpha}y^1_\alpha z^1_\alpha \vee \boxed{1}\ P_{o\alpha\alpha}x^1_\alpha z^1_\alpha\right] \\
\qquad \forall x^{1}_\alpha{}' \forall y^{1}_\alpha{}' \forall z^{1}_\alpha{}' \\
\qquad \left[\boxed{3'}\ \sim P_{o\alpha\alpha}x^{1}_\alpha{}' y^{1}_\alpha{}' \vee \boxed{2'}\ \sim P_{o\alpha\alpha}y^{1}_\alpha{}' z^{1}_\alpha{}' \vee \boxed{1'}\ P_{o\alpha\alpha}x^{1}_\alpha{}' z^{1}_\alpha{}'\right] \\
\qquad \forall x^{1}_\alpha{}'' \forall y^{1}_\alpha{}'' \forall z^{1}_\alpha{}'' \\
\qquad \left[\boxed{3''}\ \sim P_{o\alpha\alpha}x^{1}_\alpha{}'' y^{1}_\alpha{}'' \vee \boxed{2''}\ \sim P_{o\alpha\alpha}y^{1}_\alpha{}'' z^{1}_\alpha{}'' \vee \boxed{1''}\ P_{o\alpha\alpha}x^{1}_\alpha{}'' z^{1}_\alpha{}''\right] \\
\qquad \boxed{18}\ \sim P_{o\alpha\alpha}A_\alpha C_\alpha
\end{array}\right]
$$

Fig. 2. THM136 after full instantiation and duplication

Although the required unification depth is not large, there are many literals with flexible heads, and this makes it difficult to rule out connections by unification failure. This, coupled with the increase in the size of the search space caused by duplicating the quantifiers, prevents TPS from finding a proof.

Notice that, in the mating, we have often mated all of the literals in some disjunction with all of the literals in some conjunction (e.g. $\boxed{16-5}$, $\boxed{17-4}$). This happens because we have instantiated some abbreviation that need not have been instantiated: for example, the disjunction containing $\boxed{5}$ and $\boxed{4}$ is an instantiation of $R_{o\alpha\alpha} \subseteq P_{o\alpha\alpha}$, and the conjunctions to which the copies of this are mated ($\boxed{16}$ and $\boxed{17}$, $\boxed{10}$ and $\boxed{11}$) are instantiations of $\sim R_{o\alpha\alpha} \subseteq p^1_{o\alpha\alpha}$ and $\sim R_{o\alpha\alpha} \subseteq p^2_{o\alpha\alpha}$.

Dual instantiation, at first sight, appears to make the problem even worse, since it produces even more literals. The same theorem, instantiated with dual instantiation, gives the vertical path diagram of Fig. 3.

$$\left[\begin{array}{c}
\boxed{19}\ \sim \text{TRANSITIVE .TRANSITIVE-CLOSURE } R_{o\alpha\alpha} \\
\boxed{20}\ \text{TRANSITIVE-CLOSURE } R_{o\alpha\alpha}A_\alpha B_\alpha \\
\forall p^1_{o\alpha\alpha} \\
\left[\left[\begin{array}{c} \boxed{21}\ \sim .R_{o\alpha\alpha} \subseteq p^1_{o\alpha\alpha} \\ \boxed{16}\ R_{o\alpha\alpha}[X^0p^1].Y^0p^1 \\ \boxed{17}\ \sim p^1_{o\alpha\alpha}[X^0p^1].Y^0p^1 \end{array}\right] \vee \left[\begin{array}{c} \boxed{22}\ \sim \text{TRANSITIVE } p^1_{o\alpha\alpha} \\ \boxed{13}\ p^1_{o\alpha\alpha}[X^1p^1].Y^1p^1 \\ \boxed{14}\ p^1_{o\alpha\alpha}[Y^1p^1].Z^0p^1 \\ \boxed{15}\ \sim p^1_{o\alpha\alpha}[X^1p^1].Z^0p^1 \end{array}\right] \vee \boxed{12}\ p^1_{o\alpha\alpha}A_\alpha B_\alpha\right] \\
\boxed{23}\ \text{TRANSITIVE-CLOSURE } R_{o\alpha\alpha}B_\alpha C_\alpha \\
\forall p^2_{o\alpha\alpha} \\
\left[\left[\begin{array}{c} \boxed{24}\ \sim .R_{o\alpha\alpha} \subseteq p^2_{o\alpha\alpha} \\ \boxed{10}\ R_{o\alpha\alpha}[X^2p^2].Y^2p^2 \\ \boxed{11}\ \sim p^2_{o\alpha\alpha}[X^2p^2].Y^2p^2 \end{array}\right] \vee \left[\begin{array}{c} \boxed{25}\ \sim \text{TRANSITIVE } p^2_{o\alpha\alpha} \\ \boxed{7}\ p^2_{o\alpha\alpha}[X^3p^2].Y^3p^2 \\ \boxed{8}\ p^2_{o\alpha\alpha}[Y^3p^2].Z^1p^2 \\ \boxed{9}\ \sim p^2_{o\alpha\alpha}[X^3p^2].Z^1p^2 \end{array}\right] \vee \boxed{6}\ p^2_{o\alpha\alpha}B_\alpha C_\alpha\right] \\
\boxed{26}\ \sim \text{TRANSITIVE-CLOSURE } R_{o\alpha\alpha}A_\alpha C_\alpha \\
\boxed{27}\ R_{o\alpha\alpha} \subseteq P_{o\alpha\alpha} \\
\forall x^2_\alpha \forall y^2_\alpha \\
\left[\boxed{5}\ \sim R_{o\alpha\alpha}x^2_\alpha y^2_\alpha \vee \boxed{4}\ P_{o\alpha\alpha}x^2_\alpha y^2_\alpha\right] \\
\boxed{28}\ \text{TRANSITIVE } P_{o\alpha\alpha} \\
\forall x^1_\alpha \forall y^1_\alpha \forall z^1_\alpha \\
\left[\boxed{3}\ \sim P_{o\alpha\alpha}x^1_\alpha y^1_\alpha \vee \boxed{2}\ \sim P_{o\alpha\alpha}y^1_\alpha z^1_\alpha \vee \boxed{1}\ P_{o\alpha\alpha}x^1_\alpha z^1_\alpha\right] \\
\boxed{18}\ \sim P_{o\alpha\alpha}A_\alpha C_\alpha
\end{array}\right]$$

Fig. 3. THM136 after dual instantiation

However, this diagram has a much simpler mating than the previous one: $\boxed{21-27}$, $\boxed{24-27}$, $\boxed{22-28}$, $\boxed{25-28}$, $\boxed{12-3}$, $\boxed{6-2}$, $\boxed{1-18}$. Note that it is no longer necessary to duplicate any quantifiers.

It is also obvious from this mating that two of the instances of TRANSITIVE, and all of the instances of $\subseteq$, need never have been instantiated at all, since we have not made any connections involving the literals in their instantiations. We can use this information to construct a proof in which no unnecessary instantiations have been performed. To do this, we begin by deleting the unused parts of the vertical path diagram, reducing it to the diagram in Fig. 4. More details about the translation process are given in Section 2.2.

$$\left[\begin{array}{l}
\forall p^1_{o\alpha\alpha} \\
\quad \left[\boxed{21} \sim .R_{o\alpha\alpha} \subseteq p^1_{o\alpha\alpha} \lor \boxed{22} \sim \text{TRANSITIVE } p^1_{o\alpha\alpha} \lor \boxed{12}\ p^1_{o\alpha\alpha} A_\alpha B_\alpha\right] \\
\forall p^2_{o\alpha\alpha} \\
\quad \left[\boxed{24} \sim .R_{o\alpha\alpha} \subseteq p^2_{o\alpha\alpha} \lor \boxed{25} \sim \text{TRANSITIVE } p^2_{o\alpha\alpha} \lor \boxed{6}\ p^2_{o\alpha\alpha} B_\alpha C_\alpha\right] \\
\qquad\qquad\qquad\qquad\qquad\qquad \boxed{27}\ R_{o\alpha\alpha} \subseteq P_{o\alpha\alpha} \\
\qquad\qquad\qquad\qquad\qquad \boxed{28}\ \text{TRANSITIVE } P_{o\alpha\alpha} \\
\qquad\qquad \forall x^1_\alpha \forall y^1_\alpha \forall z^1_\alpha \\
\qquad\qquad\quad \left[\boxed{3} \sim P_{o\alpha\alpha} x^1_\alpha y^1_\alpha \lor \boxed{2} \sim P_{o\alpha\alpha} y^1_\alpha z^1_\alpha \lor \boxed{1}\ P_{o\alpha\alpha} x^1_\alpha z^1_\alpha\right] \\
\qquad\qquad\qquad\qquad\qquad\qquad \boxed{18} \sim P_{o\alpha\alpha} A_\alpha C_\alpha
\end{array}\right]$$

Fig. 4. The diagram in Fig. 2 after deleting unnecessary instantiations

2.1 Automatic Search

The automatic search procedures of TPS work well with dual instantiation; all that is required is to ensure that when an open path is being searched for two complementary literals, each literal containing an uninstantiated abbreviation is considered before any of the literals in the corresponding instantiation. In TPS, the construction above will suffice to do this, since on each vertical path, the search for a connection (potentially complementary pair of literals) proceeds from the top of the vertical path diagram downwards.

Although dual instantiation introduces new literals, it does not introduce any new paths into the vertical path diagram. It does, however, introduce a number of new potential connections, which might be expected to cause a combinatorial explosion of the required search time. We now explain why this is not usually the case, and why dual instantiation shortens the search time for most of the proofs which contain some abbreviation that need not be instantiated.

Firstly, we consider the reasons why dual instantiation often reduces the search time. Dual instantiation may reduce the number of quantifier duplications required, by removing unnecessary unification steps. One can see this effect by comparing Fig. 2 with Fig. 3. In the former, TPS would first attempt to unify x^2_α with both $X^0 p^1_{o\alpha\alpha}$ and $X^2 p^2_{o\alpha\alpha}$; since this is impossible, it would then have to duplicate the quantifier containing x^2 in order to complete the mating. In the latter, this is unnecessary. Removing unnecessary unification steps also reduces the depth of the unification tree, and hence improves the search time by allowing for stronger constraints on unification. (Since higher-order unification is undecidable, unification in TPS is always given a bound, roughly corresponding to the depth of the tree generated by the algorithm in [14].)

More importantly, an instantiated definition usually consists of more than one literal, and these literals may occur on paths that are not considered consecutively by the search procedure. Consider Fig. 2: the first correct connection that TPS will find is $\boxed{16-5}$. The leftmost open path will then be $\boxed{16}$-$\boxed{17}$-$\boxed{10}$-$\boxed{11}$-$\boxed{4}$-..., and before it can find and add the connection $\boxed{17-4}$, it must consider all the other potential connections that can be made from $\boxed{16}$, $\boxed{17}$, $\boxed{10}$ and $\boxed{11}$. Not only does this waste time, but if any of these connections is acceptable it will be added to the mating and the rest of the search space will be searched until unification failure forces us to backtrack and remove it. Compare Fig. 3: the first correct connection that TPS will find is $\boxed{21-27}$, and this blocks all paths through $\boxed{21}$,$\boxed{16}$ and $\boxed{17}$ at one stroke.

Secondly, we consider the reasons why dual instantiation does not cause a combinatorial explosion in the search space. In the presence of dual instantiation, TPS never considers connections between an uninstantiated literal A and any of the literals of its instantiation A', since such connections cannot possibly be required in the mating. Also, the new uninstantiated literals are simply unable to unify with many of the other literals in the diagram (and when they do unify, the unification constraints that this generates are usually so strong that TPS is seldom sent off down a blind alley; if the connection is wrong, it will become obvious very quickly). This allows us to rule out most possible connections involving the uninstantiated literals almost immediately.

2.2 Proof Translation

Once a complete mating is found, TPS uses it to construct a natural deduction proof of the original theorem. The translation from expansion trees to natural deduction proofs was described in [18]; however, a little more work is required when dual instantiation is used, since the modified expansion tree does not correspond to the original sentence in the usual manner. These changes occur in two steps of the procedure: during merging and after translation.

Merging is the procedure which deletes any unnecessary parts of the tree (in particular, it deletes any subtrees which do not occur in the mating), merges together any redundant branches, applies unification substitutions, and generally cleans up the results prior to translation into a natural deduction proof. During merging, TPS finds each of the points at which dual instantiation was used. These parts of the expansion tree will each be a subtree of the same form as the "dual instantiation" tree in Fig. 5, where A is a literal containing an uninstantiated abbreviation, and A' is a subtree containing the instantiation of A.

The diagrams in Fig. 5 show the relevant parts of the expansion trees for the three possibilities for instantiation. Note that in the construction of the expansion tree, dual instantiation occurs in several steps: first A is replaced by $A \wedge A$ (by $A \vee A$ if this is a negative occurrence of A), then the conjunction

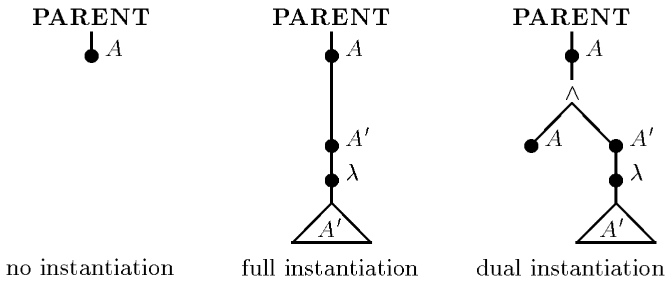

Fig. 5. Internal representations of instantiations

(resp. disjunction) is deepened, and then the rightmost A is instantiated to A' and deepened into an expansion tree. The λ-nodes, which are obtained by λ-normalization of their parent nodes, will not occur if A' is already in λ-normal form.

We now proceed by a case analysis of the leaves of the expansion tree which occur in the complete mating. We assume that unnecessary connections in the mating have already been removed, so that the mating is minimal.

1. If no leaves of the instantiated expansion subtree A' occur in the mating, then regardless of whether the uninstantiated literal A occurs or not, replace the dual expansion subtree with the "no instantiation" expansion subtree.
2. If the uninstantiated literal A does not occur in the mating, but literals from the instantiated expansion subtree A' do occur, replace the dual expansion subtree with the "full instantiation" expansion subtree.
3. Otherwise, if both A and leaves of A' occur in the mating, leave the dual expansion subtree as it is.

The translation from an expansion tree to a natural deduction proof is then performed using the methods of [18], with additional special cases which allow $A \wedge A'$ and $A \vee A'$ to be seen by TPS as equivalent to A under the "Elimination of Definitions" rule. The remaining dual instantiations will occur as either support or planned lines in the proof; for example, the following is a proof fragment corresponding to an introduction of $A \wedge A'$ as a support line:

(l_1)	(hyps)	$\vdash A$	(justification)
(l_2)	(hyps)	$\vdash A \wedge A'$	Edef: l_1
(l_3)	(hyps)	$\vdash A$	Econj: l_2
(l_4)	(hyps)	$\vdash A'$	Econj: l_2

where "Edef" means "by elimination of definitions" and "Econj" means "by elimination of a conjunction". The step from line l_1 to line l_2 corresponds to a dual instantiation of the topmost abbreviation in A, and from this we deduce line l_3 (corresponding to the uninstantiated literal) and line l_4 (corresponding to the instantiated expansion tree). The hypotheses are the same for each line.

After the translation is complete, we produce a more natural proof by replacing these four lines with the following:

(l_1)	(hyps)	$\vdash A$	(justification)
(l_4)	(hyps)	$\vdash A'$	Edef: l_1

We also change any line in the proof using l_3 as a justification, by justifying it from l_1 instead. No lines in the proof, other than l_3 and l_4, will have been justified from l_2. It is trivial to see that these changes do not affect the validity of the proof.

We omit discussion of the case in which $A \vee A'$ occurs as a planned line; it occurs only rarely and is more technical, since there are several proof fragments which TPS might generate to justify such a planned line.

The other two cases are clearly redundant and cannot happen due to the minimality of the mating. If A is a planned line, it is clearly redundant to instantiate A to $A \wedge A'$ and then justify this new plan by proving both A and A' and introducing a conjunction. Similarly, if A is a support line for some goal G, it is redundant to instantiate A to $A \vee A'$ and then prove G twice, once from A and once from A'.

After these transformations of the natural deduction proof are complete, the resulting natural deduction proof will show no signs of dual instantiation.

2.3 Examples of Theorems Proven Using Dual Instantiation

All the timings quoted below represent the time taken for a complete proof, including translation to natural deduction style, on a Dell System Workstation 400MT, with dual 300 MHz Intel Pentium II processors, running TPS under Allegro Common Lisp.

THM48: $\forall F_{\alpha\beta} \forall G_{\beta\xi}.\text{INJECTIVE} F \wedge \text{INJECTIVE } G \supset \text{INJECTIVE } .F \circ G$ where INJECTIVE is defined by $\lambda f_{\alpha\beta} \forall x_\beta \forall y_\beta . fx = fy \supset x = y$, and $\circ$ is defined by $\lambda f_{\alpha\beta} \lambda g_{\beta\gamma} \lambda x_\gamma f.gx$. In this theorem, both INJECTIVE and $\circ$ must be instantiated; however, the proof time differs depending on the treatment of equality. If equality is instantiated using the Leibniz definition ($x = y$ becomes $\forall q.qx \supset qy$) then the proof takes 3.2 seconds; if equality is not instantiated at all, then it takes 0.1 seconds. Using dual instantiation, we get a proof in 0.2 seconds; note that this is almost as fast as the "cheating" proof in which TPS is told not to instantiate equalities at all.

THM70: $\forall S_{o(o\alpha\alpha)} \text{TRANSITIVE } . \bigcap . \lambda R_{o\alpha\alpha}.\text{TRANSITIVE } R \wedge SR$
This theorem states that the intersection of a set of transitive relations is transitive. Using dual instantiation, TPS can prove it in 1.2 seconds; without dual instantiation TPS has not found a proof.

THM136: $\forall r_{o\alpha\alpha} \text{TRANSITIVE } .\text{TRANSITIVE-CLOSURE } r$
This is the theorem which was discussed earlier in this paper. Using dual instantiation, TPS can prove this theorem in 0.6 seconds. Without dual instantiation, TPS has not been able to find a proof, even after searching for several weeks.

THM250: $\forall R_{o\alpha\alpha} \forall S_{o\alpha\alpha}.\text{TRCL } [R \cup S] = \text{TRCL } .\text{TRCL } R \cup \text{TRCL } S$
This theorem was suggested by Xiaorong Huang. TRCL is the transitive closure, as before, and $\cup$ is the union of a set of pairs, which is defined by $\lambda A_{o\alpha\alpha} \lambda B_{o\alpha\alpha} \lambda x_\alpha \lambda y_\alpha . Axy \vee Bxy$. Instantiating all of the many abbreviations in this theorem results in a very large vertical path diagram, since TRCL is itself defined in

terms of $\subseteq$ and TRANSITIVE, so it is not surprising that TPS cannot prove this using full instantiation. Using dual instantiation, a proof is found in 11.3 seconds.

THM252: $\forall P_{o(o\alpha\alpha)} \forall R_{o\alpha\alpha} \forall S_{o\alpha\alpha}.\text{CL } P[R \cup S] = \text{CL } P.\text{CL } PR \cup \text{CL } PS$
Inspection of the proof of THM250 revealed that the definition of TRANSITIVE was never used, and hence that the theorem is also true of the closure of relations R and S under any property P. We define this closure CL as $\lambda P_{o(o\alpha\alpha)} \lambda r_{o\alpha\alpha} \lambda x_\alpha \lambda y_\alpha \forall q_{o\alpha\alpha}.r \subseteq q \wedge Pq \supset qxy$. This theorem is easier to prove than THM250, although again it requires dual instantiation. The proof is found in 6.5 seconds (of which the actual mating search takes only 0.6 seconds).

THM260: $\forall R_{o\alpha\alpha}.\text{EQUIV-REL } R \supset \text{PARTITION .EQUIV-CLASSES } R$
This theorem states that the equivalence classes of an equivalence relation form a partition. The proof requires that the definitions of EQUIV-REL, PARTITION and EQUIV-CLASSES, as well as the definitions of REFLEXIVE, TRANSITIVE and SYMMETRIC that they contain, all be fully instantiated. Because of this, we would expect dual instantiation to take longer, since it increases the search space without providing any benefit. The proof with full instantiation takes 1.6 seconds; that with dual instantiation takes 3 seconds.

GAZING-THM43: $\forall S_{o\alpha} \forall T_{o\alpha} \forall U_{o\alpha}.[S \bigtriangleup T] \bigtriangleup U = S \bigtriangleup [T \bigtriangleup U]$
This is example 43 from [6]. $\bigtriangleup$ is the symmetric difference, defined as usual in terms of the abbreviations $\cup$ and $-$. In the proof which TPS finds, all abbreviations must be instantiated, and so dual instantiation will not help. The proof with full instantiation takes 15 seconds; that with dual instantiation takes 18 seconds.

3 Related Work on Instantiating Definitions

In the UT theorem prover described in [8], a technique called *peeking* was implemented. In this technique, each abbreviation was associated with a list of all the predicate symbols in its definition, and an abbreviation would be instantiated if one of these symbols occurred in the goal to be proven. This only allowed for looking ahead a single step at a time (the possibility of further rewriting the predicate symbols in the definition was not considered), and ignored function symbols completely. A different and more specialized approach was used in the theorem prover for set theory which is described in [17]. In both cases, it was possible for an attempted proof to fail due to insufficient instantiation, or to succeed but contain unnecessary instantiations. Dual instantiation is similar in spirit, but allows for multiple-step look-ahead by allowing the uninstantiated and instantiated forms of every abbreviation to occur in the diagram. Furthermore, an attempted proof cannot fail from insufficient instantiation, and if the theorem is eventually proven then the resulting natural deduction proof will contain no unnecessary instantiations.

Gazing, as defined in [21] and [6], is based on the idea of searching for a collection of concepts used in both the hypotheses and the conclusions. This is sometimes described as "building the *common currency*", and is done by constructing a *predicate abstraction space* around the database of rewrite rules which comprise the theory. Proofs in the abstraction space can be constructed in a decidable propositional theory and then generalized (sometimes) to proofs in the original undecidable first-order theory; unfortunately, this generalization is not always possible. The requirement to construct a different abstraction space for each domain, and then to discover by experimentation whether or not the abstraction space is useful in that domain, is a problem with gazing that does not occur with dual instantiation. Gazing also has difficulty with theories containing many function symbols and few predicate symbols, so that although it works well for naïve set theory, it is of little or no use for equational theories. Furthermore, there is still the possibility of either unnecessary or insufficient instantiation. However, gazing also provides a partial solution to the problem of using lemmas and rewrite rules within a theory, whereas dual instantiation does not.

The idea of using abstraction to simplify a search has a long history in artificial intelligence, and further references on this subject can be found in [13]. A theory of abstraction for resolution-based theorem-proving was developed in [19] and [20], and a more generalized theory of abstraction was developed in [13].

Gazing was further developed in [12] (in particular, the handling of function symbols was improved), and is the basis of the ABSFOL theorem prover described in [11]. The idea has also been implemented [22] for a theorem prover using the connection method.

All of the papers mentioned above are concerned mostly with first-order logic.

4 Instantiating Set Variables

In both first-order and higher-order logic, a key step in discovering a proof of a theorem is finding correct instantiations for the variables. In higher-order logic, however, variables with types of the form $(o\alpha)$ will correspond to sets; these variables may require instantiations which cannot be generated by unification alone. For example, it may be necessary to instantiate $p_{o\alpha}$ with the term $\lambda w_\alpha . A_{o\alpha} w \wedge B_{o\alpha} w$, meaning that p is the intersection of the two sets A and B. The problem of instantiating set variables in higher-order proof search has been addressed by papers such as [3], [4], [5], [7] and [9]; we have found that the techniques used for dual instantiation can also help with this problem.

TPS can generate instantiations for set variables incrementally, by first replacing a variable $p_{o\alpha}$ with one of the following:

conjunction $\lambda w_\alpha . p^1_{o\alpha} w \wedge p^2_{o\alpha} w$ **universal quantifier** $\lambda w_\alpha \forall x_\gamma p^1_{o\gamma\alpha} w x$
disjunction $\lambda w_\alpha . p^1_{o\alpha} w \vee p^2_{o\alpha} w$ **existential quantifier** $\lambda w_\alpha \exists x_\gamma p^1_{o\gamma\alpha} w x$

The new set variables p^i in these formulae may then be replaced with similar conjunctions, disjunctions and quantifiers, to generate instantiations of arbitrary depth.

We have added to TPS the ability to generate set variable instantiations which contain some of the abbreviations that occur in the sentence being proven – so that, for example, a sentence containing equality and the definitions TRANSITIVE and SUBSET can generate set variable substitutions for $p_{o\alpha}$ of the following forms:

equality $\lambda w_\alpha . p^1_{\gamma\alpha} w = p^2_{\gamma\alpha} w$ **transitivity** $\lambda w_\alpha .\text{TRANSITIVE } . p^1_{o\gamma\gamma\alpha} w$

subset $\lambda w_\alpha . p^1_{o\gamma\alpha} w \subseteq p^2_{o\gamma\alpha} w$

We refer to these new set variable instantiations as *abbreviational set instantiations*, or ABR instantiations. To construct ABR instantiations, we first search the expansion tree, extracting the formulae stored at each leaf and rewrite node. In these formulae, which will all be of type o, we search for maximal subformulae which contain only variables and constants (but not, for example, quantifiers or abbreviations), and replace these formulae by *blanks* of appropriate types, denoted $\bullet_\tau$. We also generate the negations of any blanked-out formulae with rigid heads, and we add the single formula $\bullet_o$ (in the unlikely event that it isn't generated anyway).

For example, THM136 contained the subformula

$$[r_{o\alpha\alpha} \subseteq p^1_{o\alpha\alpha} \wedge \text{TRANSITIVE } p^1 \supset p^1 a_\alpha b_\alpha]$$

which, when deepened into a subtree of the expansion tree, will generate blanked formulae

$$\begin{array}{ccc} \bullet_{o\alpha\alpha} \subseteq \bullet_{o\alpha\alpha} & \text{TRANSITIVE } \bullet_{o\alpha\alpha} & \bullet_o \\ \sim [\bullet_{o\alpha\alpha} \subseteq \bullet_{o\alpha\alpha}] & \sim [\text{TRANSITIVE } \bullet_{o\alpha\alpha}] & \end{array}$$

These formulae are then combined into larger formulae, using conjunctions, disjunctions and quantifiers exactly as before. Quantified variables are also left blank at this stage. Because of the number of possible combinations, we usually only generate these larger formulae in prenex normal form. Using the formulae above, we can generate

$$\forall \bullet_\alpha . [\sim [\bullet_{o\alpha\alpha} \subseteq \bullet_{o\alpha\alpha}]] \wedge \bullet_o$$

To transform this into a substitution for a variable such as $p_{o\alpha\alpha}$, we add a prefix of lambda-bound variables (in this case, $\lambda w^1_\alpha \lambda w^2_\alpha$), replace the quantified blanks with variables of the appropriate type (in this case, w^3_α), and then replace each of the remaining blanks with a new set variable of appropriate type which takes all of the w-variables as arguments. The blanked-out formula above would thus generate a substitution of the form:

$$\lambda w^1_\alpha \lambda w^2_\alpha \forall w^3_\alpha . [\sim [p^1_{o\alpha\alpha\alpha\alpha\alpha} w^1 w^2 w^3 \subseteq p^2_{o\alpha\alpha\alpha\alpha\alpha} w^1 w^2 w^3]] \wedge p^3_{o\alpha\alpha\alpha} w^1 w^2 w^3$$

Because the formula $\bullet_o$ was definitely generated, all of the substitutions which were generated by the old procedure will also be generated here (since they amount to combining $\bullet_o$ with itself using conjunctions, disjunctions and quantifiers).

ABR instantiations are useful for much the same reasons as dual instantiation: they help to minimize the depth of unification required, and provide stronger unification constraints than the more general set instantiations. In addition, they are more compact than the equivalent substitutions in which the abbreviations have been instantiated, and hence can reduce the size of the vertical path diagram. They also interact well with dual instantiations; several of the examples listed below can only be proven by TPS using both dual instantiation and ABR instantiation.

4.1 Examples of Theorems Proven Using ABR Instantiation

THM15B: $\forall f_{\iota\iota}.\exists g_{\iota\iota}[\text{ITERATE+ } fg \wedge \exists x_\iota.gx = x \wedge \forall z_\iota.gz = z \supset z = x] \supset \exists y_\iota.fy = y$

This theorem was stated in [15], and was discussed in some detail in [4]. It states that if some iterate g of a function f has a unique fixed point, then f has a fixed point. ITERATE+ is defined by $\lambda f_{\alpha\alpha}\lambda g_{\alpha\alpha}\forall p_{o(\alpha\alpha)}.pf \wedge \forall j_{\alpha\alpha}[pj \supset p.f \circ j] \supset pg$, and the principal problem for TPS is to find the appropriate instantiation for p. This instantiation will be the set of all the functions that commute with f; TPS then proves that g is one such function, and hence that the unique fixed point of g is a fixed point of f. Because TPS now finds the ABR instantiation $\lambda F_{\iota\iota}.f_{\iota\iota}[Fx_\iota] = F.fx$ (instead of the old set instantiation $\lambda g_{\iota\iota}\forall q_{o\iota}.q[g.f_{\iota\iota}x_\iota] \supset q.f.gx$), the proof can now be completed in 4 seconds instead of 1 minute, and the resulting natural deduction proof is more readable and 20% shorter.

THM120I: $\exists R_{o(o\iota)(o\iota)}.\text{REFLEXIVE } R \wedge \text{TRANSITIVE } R \wedge \forall X_{o\iota}\forall Y_{o\iota}.RXY \wedge RYX \supset X \equiv^s Y$

$\equiv^s$ is set equivalence, defined by $\lambda P_{o\alpha}\lambda R_{o\alpha}\forall x_\alpha.Px = Rx$. This theorem says that there is a reflexive, transitive and antisymmetric relation between sets. Using dual instantiation and ABR instantiation, TPS discovers the relation $\equiv^s$ and completes the proof in 10 seconds.

X5211: $y_{o\alpha} = \bigcup.\lambda z_{o\alpha}\exists x_\alpha.yx \wedge z = [= x]$

This says that a set is the union of the unit-sets of each of its elements. (Note that $[= x]$ is the unit set which is normally written as $\{x\}$, since both are equal to $\lambda y.y = x$.) This theorem requires both dual instantiation (of equality) and ABR instantiation in order for TPS to find a proof. It can be proven in 1.3 seconds.

X6104: $\exists i_{o(\alpha\alpha)(\alpha\alpha)}.\forall g_{\alpha\alpha}[ig[\lambda x_\alpha x] \wedge ig.\lambda x.g.gx] \wedge \forall f_{\alpha\alpha}\forall y_\alpha.i[\lambda xy]f \supset fy = y$

This says that there is a relation i between functions such that every function g is related both to the identity function and to $g \circ g$, and that $fy = y$ if the constant function λxy is related to f. The theorem was formulated with the expectation that TPS would discover a proof in which i is the "ITERATE" relation: f is related to g iff g is an iterate of f. However, TPS discovers a short proof in which i is the "commutes with" relation, taking 2.5 seconds in total. Again, this requires both dual instantiation and ABR instantiation.

5 Conclusions and Further Work

We have demonstrated that dual instantiation can provide an effective way to decide which abbreviations to instantiate when searching for a proof using the connection method. Moreover, by using dual instantiation one is guaranteed that the search for a proof cannot possibly fail due to insufficient instantiation, and that if the search succeeds then the resulting natural deduction proof will contain no unnecessary instantiations. This method of instantiation has allowed TPS to prove many previously intractable theorems, and has improved the search time and the readability of the natural deduction proofs produced by TPS for a number of other theorems. Several other lines of enquiry that have been suggested by our work on dual instantiation are outlined below.

5.1 Multiple Instantiation

The method of dual instantiation could easily be generalized to *multiple instantiation*, in which a single abbreviation is allowed to have several different instantiations — so that, for example,

$$A_{o\alpha} = B_{o\alpha} \text{ becomes } [A_{o\alpha} = B_{o\alpha}] \wedge [\forall x_\alpha . Ax \equiv Bx] \wedge [\forall q_{o(o\alpha)} . qA \supset qB]$$

$$P_{o\alpha} \subseteq Q_{o\alpha} \text{ becomes } [P_{o\alpha} \subseteq Q_{o\alpha}] \wedge [P \cap Q = P] \wedge [P \cup Q = Q] \wedge [\forall x_\alpha . Px \supset Qx]$$

By preventing the search procedure from making connections between various instantiations of the same occurrence of an abbreviation, just as we currently forbid connections between the uninstantiated and instantiated abbreviations, it might then be possible to avoid a combinatorial explosion in the size of the search space and yet allow the search procedure to use whichever definition is most natural for a given proof.

We have not yet implemented multiple instantiation in TPS.

5.2 Discovering Abbreviations

It would be very useful to have a theorem prover capable of inventing abbreviations; this would correspond in some way to discovering a mathematical concept and putting it to use. As we remarked earlier in the context of THM136, when an entire conjunction generated from a formula involving an abbreviation is mated to a similar entire disjunction, it is a sign that the abbreviation need not have been instantiated. The converse is also true: whenever a complete mating contains a subset of connections which mate one entire subtree to a negated version of itself, this suggests that the subtree in question might usefully have been replaced by an abbreviation.

TPS is currently capable of discovering such subsets, and of suggesting abbreviations that might shorten the proof. At the moment, this routine is called after the tree is merged (and hence after a proof has already been discovered), and nothing is done with the suggested abbreviations.

References

1. Peter B. Andrews. Theorem Proving via General Matings. *Journal of the ACM*, 28:193–214, 1981.

2. Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, 1986.
3. Peter B. Andrews. On Connections and Higher-Order Logic. *Journal of Automated Reasoning*, 5:257–291, 1989.
4. Peter B. Andrews, Matthew Bishop, Sunil Issar, Dan Nesmith, Frank Pfenning, and Hongwei Xi. TPS: A Theorem Proving System for Classical Type Theory. *Journal of Automated Reasoning*, 16:321–353, 1996.
5. Sidney C. Bailin and Dave Barker-Plummer. Z-match: An Inference Rule for Incrementally Elaborating Set Instantiations. *Journal of Automated Reasoning*, 11:391–428, 1993. Errata: JAR 12:411–412, 1994.
6. Dave Barker-Plummer. Gazing: An Approach to the Problem of Definition and Lemma Use. *Journal of Automated Reasoning*, 8:311–344, 1992.
7. W. W. Bledsoe. Using Examples to Generate Instantiations of Set Variables. In *Proceedings of IJCAI-83, Karlsruhe, Germany*, pages 892–901, Aug 8-12, 1983.
8. W. W. Bledsoe and Peter Bruell. A Man-Machine Theorem-Proving System. *Artificial Intelligence*, 5(1):51–72, 1974.
9. W. W. Bledsoe and Gohui Feng. Set-Var. *Journal of Automated Reasoning*, 11:293–314, 1993.
10. Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.
11. Fausto Giunchiglia and Adolfo Villafiorita. ABSFOL: a Proof Checker with Abstraction. In M.A. McRobbie and J.K. Slaney, editors, *CADE-13: Proceedings of the 13th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 1104, pages 136–140. Springer-Verlag, 1996.
12. Fausto Giunchiglia and Toby Walsh. Theorem Proving with Definitions. In *Proceedings of AISB 89*, Society for the Study of Artificial Intelligence and Simulation of Behaviour, 1989.
13. Fausto Giunchiglia and Toby Walsh. A Theory of Abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.
14. Gerard P. Huet. A Unification Algorithm for Typed λ-Calculus. *Theoretical Computer Science*, 1:27–57, 1975.
15. Ignace I. Kolodner. Fixed Points. *American Mathematical Monthly*, 71:906, 1964.
16. Dale A. Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.
17. D. Pastre. Automatic Theorem Proving in Set Theory. *Artificial Intelligence*, 10:1–27, 1978.
18. Frank Pfenning. *Proof Transformations in Higher-Order Logic.* PhD thesis, Carnegie Mellon University, 1987. 156 pp.
19. D.A. Plaisted. Abstraction Mappings in Mechanical Theorem Proving. In *5th Conference on Automated Deduction*, Lecture Notes in Computer Science 87, pages 264–280. Springer-Verlag, 1980.
20. D.A. Plaisted. Theorem Proving with Abstraction. *Artificial Intelligence*, 16:47–108, 1981.
21. Dave Plummer. *Gazing: Controlling the Use of Rewrite Rules.* PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1987.
22. K. Warren. Implementation of a Definition Expansion Mechanism in a Connection Method Theorem Prover. Master's thesis, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1987.
23. Larry Wos. The Problem of Definition Expansion and Contraction. *Journal of Automated Reasoning*, 3:433–435, 1987.