Proof by Pointing*

Yves Bertot[1] Gilles Kahn[1] Laurent Théry[2]

[1] INRIA Sophia-Antipolis
B.P. 93, F-06902 Sophia Antipolis Cedex, France
{Yves.Bertot,Gilles.Kahn}@sophia.inria.fr
[2] University of Cambridge, Computer Laboratory
New Museums Site, Pembroke Street,
Cambridge CB2 3QG, England
Laurent.Thery@cl.cam.ac.uk

Abstract. This paper presents a principle for using locations in logical expressions to guide the process of building proofs. Using a sequent-style presentation of theorem provers, we annotate the inference rules to specify an *algorithm* that associates the construction of a proof tree to a location within a goal sequent. This principle provides a natural and effective use of the mouse in the user-interface of computer proof assistants. The implementation of the algorithm in a variety of theorem provers is discussed.

1 Introduction

A number of very powerful and elegant computer programs to assist in making formal proofs have been developed in the last decade. These systems include ever more sophisticated automatic proof tactics. Nevertheless, proofs that can be carried out without any user directions are the exception rather than the rule. In this paper, we present a general principle called *proof by pointing* that allows the user to guide precisely the proof process with the mouse of his workstation. This idea is widely applicable and has been implemented by the authors in user-interfaces for several proof development systems: Amy Felty's Theorem Prover [Felty89], Coq [Coq91], HOL [HOL88], and Isabelle [Isa90].

The paper is organized as follows: first, we give an example of the kind of interaction that results from adopting our principle. Then the second section describes rigorously the principle, its logical foundations and simple consequences. The third section examines potential difficulties encountered when implementing the idea in a variety of proof assistants. Finally, the last section discusses possible extensions.

An Example

In contrast with symbolic algebra systems such as MAPLE [Maple] or Mathematica [Matica] that operate as symbolic desk calculators, proof development

* This work was supported in part by the "Types for Proofs and Programs" Esprit Basic Research Action, by SERC grant GR/G 33837 and a grant from DSTO Australia.

systems usually work in a mode where the user sets a goal and attacks it in a backward-chaining (or goal-directed) manner: the user tries to eliminate one of several pending subgoals by applying a theorem that matches it. In case of success, the goal is removed from the list of pending subgoals and new subgoals, that correspond to verifying that the theorem's hypotheses hold, are added to the list of subgoals. Of course, the user may follow a strategy that does not lead to a proof, and need to backtrack and attempt an alternative proof.

With standard user-interfaces, the user issues commands to perform such actions. These commands may be typed by hand or constructed using a structured editor as in [TBK92] or [Nuprl86]. The idea in *proof by pointing* is that the mere gesture of pointing at a subexpression in a subgoal is enough to synthesize appropriate commands for the system.

Consider for example the following formula in first order logic, where a and b are individuals and p and q are predicate symbols, and assume that it is entered as goal G_0:

$$(G_0) \qquad (p(a) \vee q(b)) \wedge (\forall x\ p(x) \supset q(x)) \supset (\exists x\ q(x))$$

Formula G_0 can be paraphrased in english: *if we know that either p is verified for a or q is verified for b and that p implies q, then there exists an x for which property q is verified.*

The proof of this fact examines the two cases involved in the formula $p(a) \vee q(b)$. In the case where $p(a)$ holds, we use the fact $\forall x\ p(x) \supset q(x)$ to deduce $q(a)$. Then a is a witness to prove $\exists x\ q(x)$ in that case. In the second case $q(b)$ holds, so the witness b is directly available. We will tell the computer exactly this *using only the mouse.*

Starting from G_0, to steer the computer toward the proof, the user points to subformula $p(a)$ with the mouse. As it occurs within expression $p(a) \vee q(b)$, this indicates interest in a case analysis. The proof state changes to include two new subgoals G_1 and G_2:

$$(G_1) \qquad p(a),\ p(a) \vee q(b),\ (p(a) \vee q(b)) \wedge (\forall x\ p(x) \supset q(x)) \vdash \exists x\ q(x)$$

$$(G_2) \qquad q(b),\ p(a) \vee q(b),\ (p(a) \vee q(b)) \wedge (\forall x\ p(x) \supset q(x)) \vdash \exists x\ q(x)$$

In our notation, the turnstile symbol $\vdash$ separates the local assumptions from the conclusion in a subgoal, and assumptions are separated by commas. Naturally, assumptions that are local to a subgoal can only be used to prove this subgoal's conclusion.

The user is free to carry on working with subgoal G_1 or G_2, although G_1 should be emphasized since $p(a)$ rather than $q(b)$ was pointed at initially. In G_1, since $p(a)$ and $\forall x\ p(x) \supset q(x)$ hold one can deduce $q(a)$. This inference step is requested by pointing at subexpression $p(x)$ in G_1, meaning *prove an instance of $p(x)$ and deduce the corresponding instance of $q(x)$.* In the proof state, subgoal G_1 is replaced by G_3:

$$(G_3) \qquad q(a),\ p(a),\ p(a) \vee q(b),\ (p(a) \vee q(b)) \wedge (\forall x\ p(x) \supset q(x)) \vdash \exists x\ q(x)$$

Now subgoal G_3 can easily be dealt with. The fact $q(a)$ appears in the assumptions and we need to prove $\exists x\ q(x)$. The user simply selects $q(x)$ behind the existential quantifier in G_3 with the intended meaning *there is a witness for x in the assumptions of this goal that allows one to prove $q(x)$*. Subgoal G_3 vanishes and only G_2 remains. Subgoal G_2 is handled in an identical fashion and vanishes as well. As no subgoals remain to be proved, the result is established. The entire interaction took place without any need for typing commands. This is not to say that all commands may be eliminated when using a proof assistant, but rather that they are unnecessary for many boring logical tasks.

The meaning of mouse designation is not ad hoc. It is entirely determined by the precise shape of the formulas we are trying to prove. The intuition is as follows. In goal G_0, expression $p(a)$ is designated by a first mouse click. Precisely, this expression occurs, starting from the top of G_0 :

i) to the left of an implication symbol (denoted by $\supset$),
ii) to the left of a conjunction symbol (denoted by $\wedge$),
iii) to the left of a disjunction symbol (denoted by $\vee$).

When pointing at $p(a)$, each one of these facts is exploited in turn:

i) the antecedent of the implication is added as an assumption,
ii) the left part of the conjunction is extracted and added as an assumption,
iii) two subgoals corresponding to the two cases in the disjunction are created, with either disjunct as additional assumption; the goal created by the left disjunct is emphasized.

The second mouse click is simpler to explain. In goal G_1 , the user points at expression $p(x)$. This expression occurs in an assumption and:

i) to the right of a conjunction symbol,
ii) within a universally quantified expression,
iii) to the left of an implication symbol.

As a consequence, pointing at $p(x)$ directs the computer to:

i) extract the right conjunct,
ii) find a proof of $p(x)$ for some x,
iii) add a new assumption $q(x)$ for the same x, creating G_3.

The last two mouse clicks are even simpler. In goals G_3 , and similarly in goal G_2, the user points at $q(x)$ in the conclusion of the goal, within the existentially quantified formula. In both cases, the system looks through the assumptions to see if an instance of $q(x)$ is directly provable. In both cases it is successful, so the goals are eliminated.

After this intuitive presentation, the meaning of mouse clicks will now be described formally, taking all usual logical connectives into account.

2 The proof by pointing algorithm

To specify rigorously the pointing algorithm, we use Gentzen's presentation of logical deduction [Szabo69] in Sequent Calculus. In this formalism, propositions are represented by *sequents*, composed of a list of *assumptions* (this list is also called a *context*) separated by a turnstile symbol, $\vdash$, from the *conclusion*. The context may be empty. Legitimate inferences are specified by *rules*. A rule has two parts separated by an horizontal bar: a list of sequents, the rule's *premises*, and a single sequent, the rule's *conclusion*. A partial proof is a tree-like composition of instances of the inference rules. When a sequent is of the form $\Gamma \vdash A$ where A occurs in Γ, it can be *closed* and this fact is represented by drawing an horizontal bar above the sequent. A proof is a partial proof where all leaves are *closed* sequents. To start with, we limit ourselves to the standard logical connectives $\wedge$, $\vee$, $\supset$, $\forall$, and $\exists$ and take the familiar inference rules given on Figure 1. The process of building a proof in a goal-directed fashion is formalized as follows: starting from the proposition to prove, either it is possible to close it, or one picks an inference rule whose conclusion matches the proposition, and then reiterates the process non-deterministically on the premises of the rule after appropriate instantiation. At any time during this process, the unclosed leaves of the partial proof tree are the *pending goals* of the proof. Hence the proof is finished when there are no pending goals left.

$$\wedge\,left : \frac{A, B, A \wedge B, \Gamma \vdash C}{A \wedge B, \Gamma \vdash C} \qquad \wedge\,right : \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$\vee\,left : \frac{A, A \vee B, \Gamma \vdash C \qquad B, A \vee B, \Gamma \vdash C}{A \vee B, \Gamma \vdash C} \qquad \vee\,right : \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$\supset left : \frac{A \supset B, \Gamma \vdash A \qquad B, A \supset B, \Gamma \vdash C}{A \supset B, \Gamma \vdash C} \qquad \supset right : \frac{A, \Gamma \vdash B}{\Gamma \vdash A \supset B}$$

$$\forall\,left : \frac{A[x \backslash e], \forall x\, A, \Gamma \vdash C}{\forall x\, A, \Gamma \vdash C} \qquad \forall\,right : \frac{\Gamma \vdash A[x \backslash c]}{\Gamma \vdash \forall x\, A}$$

$$\exists\,left : \frac{A[x \backslash c], \exists x\, A, \Gamma \vdash C}{\exists x\, A, \Gamma \vdash C} \qquad \exists\,right : \frac{\Gamma \vdash A[x \backslash e]}{\Gamma \vdash \exists x\, A}$$

Fig. 1. Rules for the basic logical connectives

To construct the proof incrementally, the user is allowed to select a *subexpression* occurring in an arbitrary pending goal. The result of this action is:

i) possibly nothing,
ii) possibly the growth of the proof tree above the goal containing the selected

subexpression; in that case, a *residual* subexpression is selected in a pending goal.

The proof by pointing algorithm defines how the partial proof tree grows as a result of a selection, and the position of the residual selection. Obviously when selecting an expression in a given goal, we mean to work on that goal. Intuitively, when selecting a subexpression σ deeply inside the goal, we want to somehow bring σ to the surface, and leave a residual selection showing where σ is now.

The definition of the proof by pointing algorithm is by induction on the depth at which the selected subexpression can be found within a goal.

2.1 Selecting a subterm at depth 1

In a sequent, we define the subterms at depth 1 as the conclusion and all assumptions. Selecting such terms is taken to mean that a closure involving the subterm should be attempted. For example, assume we have the following goal:

$$p(a), p(b), \ldots \vdash p(b)$$

Selecting the first instance of $p(b)$ (an assumption)

$$p(a), \underline{p(b)}, \ldots \vdash p(b)$$

or the last instance (the conclusion)

$$p(a), p(b), \ldots \vdash \underline{p(b)}$$

closes the sequent and leaves the residual selection undefined:

$$\overline{p(a), p(b), \ldots \vdash p(b)}$$

Selecting an assumption that doesn't match the conclusion such as $p(a)$ simply leaves the goal unchanged and the residual selection on the selected assumption.

2.2 Selecting a subterm at depth n, where $n > 1$

Subterms located at depth n, where $n > 1$, are the descendants of the top logical connective of either the conclusion or some assumption. The rules of Figure 1 are partitioned into *right* and *left* rules. We take selecting *in the conclusion* of a goal to mean that we begin to grow the proof tree using a right rule, and selecting *in an assumption* that we start with a left rule. Now if the top logical connective is κ we use the κ-right (resp. the κ-left) rule. In the case of a conclusion that is a disjunction ($\vee$ *right*) we pick the first rule if the selection is to the left of the connective, the second one if it is to the right.

We rewrite the rules of Figure 1 in a more appropriate manner on Figure 2. A box around a subexpression means that the selection is *inside* that box. The box in a rule's conclusion means that the rule is applicable only if the selection is inside. A box in a rule's premises tells where the residual selection is after one

$$\wedge\, left_1 : \frac{\boxed{A}, B, A \wedge B, \Gamma \vdash C}{\boxed{A} \wedge B, \Gamma \vdash C} \qquad \wedge\, right_1 : \frac{\Gamma \vdash \boxed{A} \qquad \Gamma \vdash B}{\Gamma \vdash \boxed{A} \wedge B}$$

$$\wedge\, left_2 : \frac{A, \boxed{B}, A \wedge B, \Gamma \vdash C}{A \wedge \boxed{B}, \Gamma \vdash C} \qquad \wedge\, right_2 : \frac{\Gamma \vdash A \qquad \Gamma \vdash \boxed{B}}{\Gamma \vdash A \wedge \boxed{B}}$$

$$\vee\, left_1 : \frac{\boxed{A}, A \vee B, \Gamma \vdash C \qquad B, A \vee B, \Gamma \vdash C}{\boxed{A} \vee B, \Gamma \vdash C} \qquad \vee\, right_1 : \frac{\Gamma \vdash \boxed{A}}{\Gamma \vdash \boxed{A} \vee B}$$

$$\vee\, left_2 : \frac{A, A \vee B, \Gamma \vdash C \qquad \boxed{B}, A \vee B, \Gamma \vdash C}{A \vee \boxed{B}, \Gamma \vdash C} \qquad \vee\, right_2 : \frac{\Gamma \vdash \boxed{B}}{\Gamma \vdash A \vee \boxed{B}}$$

$$\supset left_1 : \frac{A \supset B, \Gamma \vdash \boxed{A} \qquad B, A \supset B, \Gamma \vdash C}{\boxed{A} \supset B, \Gamma \vdash C} \qquad \supset right_1 : \frac{\boxed{A}, \Gamma \vdash B}{\Gamma \vdash \boxed{A} \supset B}$$

$$\supset left_2 : \frac{A \supset B, \Gamma \vdash A \qquad \boxed{B}, A \supset B, \Gamma \vdash C}{A \supset \boxed{B}, \Gamma \vdash C} \qquad \supset right_2 : \frac{A, \Gamma \vdash \boxed{B}}{\Gamma \vdash A \supset \boxed{B}}$$

$$\forall\, left : \frac{\boxed{A[x\backslash e]}, \forall x\, A, \Gamma \vdash C}{\forall x \boxed{A}, \Gamma \vdash C} \qquad \forall\, right : \frac{\Gamma \vdash \boxed{A[x\backslash c]}}{\Gamma \vdash \forall x \boxed{A}}$$

$$\exists\, left : \frac{\boxed{A[x\backslash c]}, \exists x\, A, \Gamma \vdash C}{\exists x \boxed{A}, \Gamma \vdash C} \qquad \exists\, right : \frac{\Gamma \vdash \boxed{A[x\backslash e]}}{\Gamma \vdash \exists x \boxed{A}}$$

Fig. 2. Rules with selection propagation

elementary step. Notice that we have now systematically two left and two right rules for each binary connective.
We can make the following two remarks on the location of the selection after *one* inference rule on Figure 2 has been used to grow the proof tree:

1. the residual of the initial selection is unique,
2. if the selection was located at depth n, its residual is located at depth $n-1$.

From the first remark, we deduce that we can apply recursively the proof by pointing algorithm to the residual of the initial selection after using one inference rule. The second remark proves that this process terminates. More precisely, the process terminates on an attempt at closure. If this attempt succeeds, there will be no residual selection. If it fails, the selection will denote the residual of the initial selection after using $n-1$ inference rules and occur at depth 1. In other words, we have been successful in obtaining a goal where this expression is at the surface, that is at depth one.

2.3 Examples

We follow the algorithm on a number of small, but revealing, examples. The selection is underlined and we use an arrow $\rightarrow$ to indicate the next goal generated by the algorithm, when it is unique.

Goal: $(p(a) \wedge p(b)) \wedge p(c) \vdash p(b)$.

Proof: click on $p(b)$ in the assumption.

$$(p(a) \wedge \underline{p(b)}) \wedge p(c) \vdash p(b) \;\rightarrow\; p(a) \wedge \underline{p(b)}, p(c), (p(a) \wedge p(b)) \wedge p(c) \vdash p(b)$$

$$\rightarrow\; p(a), \underline{p(b)}, p(a) \wedge p(b), p(c), (p(a) \wedge p(b)) \wedge p(c) \vdash p(b)$$

$$\rightarrow\; \overline{p(a), p(b), p(a) \wedge p(b), p(c), (p(a) \wedge p(b)) \wedge p(c) \vdash p(b)}$$

Goal: $a \vee b \vdash b \vee a$.

Proof: clicking to the left on a, the goal $\underline{a} \vee b \vdash b \vee a$ generates two cases:

$$\underline{a}, a \vee b \vdash b \vee a$$

$$b, a \vee b \vdash b \vee a$$

The first goal contains the selection. It is proved by clicking on a in the conclusion. The second goal is resolved similarly by clicking on b in the conclusion. The proof has been done with three mouse clicks. Clicking initially in the conclusion is a dead-end.

Goal: $\vdash p(a) \supset p(a)$.

Proof: click on either instance of $p(a)$. For example, clicking on the left gives:

$$\vdash \underline{p(a)} \supset p(a) \;\rightarrow\; \underline{p(a)} \vdash p(a) \;\rightarrow\; \overline{p(a) \vdash p(a)}$$

Goal: $\vdash a \supset (b \supset (a \wedge b))$.

Proof: seeing the outermost a, one wants to click on the innermost one:

$$\vdash a \supset (b \supset (\underline{a} \wedge b)) \;\rightarrow\; a \vdash b \supset (\underline{a} \wedge b) \;\rightarrow\; b, a \vdash \underline{a} \wedge b$$

Here we get two goals (by $\wedge\, right_1$):

$$b, a \vdash \underline{a}$$

$$b, a \vdash b$$

The algorithm closes the first one and stops for lack of a residual selection, leaving the user with the second goal. Clicking on the conclusion b finishes the proof, which needed two clicks.

Goal: $\vdash a \supset ((a \supset b) \supset b)$

Proof: this proposition is proved with two clicks, either in a *forwards* style (building up new facts from the hypotheses) or in a *backwards* style (coming back from the goal to prove). Backwards, one points at the leftmost b.

$$\vdash a \supset ((a \supset \underline{b}) \supset b) \;\rightarrow\; a \vdash (a \supset \underline{b}) \supset b \;\rightarrow\; a \supset \underline{b}, a \vdash b$$

Here the algorithm produces two goals and solves the second one:

$$a \supset b, a \vdash a$$

$$\underline{b}, a \supset b, a \vdash b$$

The user closes the first goal trivially.

Noticing the outer a, the user could have used the forwards style and clicked on the inner a :

$$\vdash a \supset ((\underline{a} \supset b) \supset b) \;\rightarrow\; a \vdash (\underline{a} \supset b) \supset b \;\rightarrow\; \underline{a} \supset b, a \vdash b$$

Again the algorithm produces two goals and solves the first one:

$$a \supset b, a \vdash \underline{a}$$

$$b, a \supset b, a \vdash b$$

The user closes the second goal trivially.

To use rules $\forall$ *left* and $\exists$ *right*, we postulate the existence of an additional mechanism that supplies appropriate terms. This can be achieved either by querying the user or by providing logical variables and an extension of the closure rule. deciding which approach to use is orthogonal to the proof by pointing algorithm; we present both in the next section.

Goal: $p(a) \wedge (\forall x\, p(x) \supset q(x)) \vdash q(a)$.
Proof: here again we can do it forwards, clicking on $p(x)$ because we know $p(a)$ or backwards, clicking on $q(x)$ because we want to prove $q(a)$. Clicking on $p(x)$:

$$\begin{aligned}
&p(a) \wedge (\forall x\, \underline{p(x)} \supset q(x)) \vdash q(a) \\
&\rightarrow p(a), \forall x\, \underline{p(x)} \supset q(x), p(a) \wedge (\forall x\, p(x) \supset q(x)) \vdash q(a) \\
&\rightarrow \underline{p(a)} \supset q(a), p(a), \forall x\, p(x) \supset q(x), p(a) \wedge \forall x\, p(x) \supset q(x) \vdash q(a)
\end{aligned}$$

Here, the algorithm generates two goals;

$$p(a) \supset q(a), p(a), \forall x\, p(x) \supset q(x), p(a) \wedge \forall x\, p(x) \supset q(x) \vdash \underline{p(a)}$$

$$q(a), p(a) \supset q(a), p(a), \forall x\, p(x) \supset q(x), p(a) \wedge \forall x\, p(x) \supset q(x) \vdash q(a)$$

The first one is closed automatically, and the second one is closed with one additional click.

As a final example, let us come back to the proof given in the introduction:
Goal: $\vdash (p(a) \vee q(b)) \wedge (\forall x\, p(x) \supset q(x)) \supset \exists x\, q(x)$
Proof: The user's first selection is on $p(a)$, to produce the case analysis.

$$\vdash (\underline{p(a)} \vee q(b)) \wedge (\forall x\, p(x) \supset q(x)) \supset \exists x\, q(x)$$

$$\rightarrow\; (\underline{p(a)} \vee q(b)) \wedge (\forall x\, p(x) \supset q(x)) \vdash \exists x\, q(x)$$

$$\rightarrow\; \underline{p(a)} \vee q(b), (\forall x\, p(x) \supset q(x)), (p(a) \vee q(b)) \wedge (\forall x\, p(x) \supset q(x)) \vdash \exists x\, q(x)$$

There the proof splits in two subgoals, as intended:

$$\underline{p(a)}, (p(a) \vee q(b)), (\forall x\, p(x) \supset q(x)), (p(a) \vee q(b) \wedge (\forall x\, p(x) \supset q(x)) \vdash \exists x\, q(x)$$

$$q(b), (p(a) \vee q(b)), (\forall x\, p(x) \supset q(x)), (p(a) \vee q(b) \wedge (\forall x\, p(x) \supset q(x)) \vdash \exists x\, q(x)$$

The second subgoal is proved by a single click on $q(x)$ in the conclusion. To prove the first subgoal, one clicks on the first occurrence of $p(x)$ just as in the previous example, and then on $q(x)$ in the conclusion.

2.4 Intuitionistic negation

Extending proof by pointing to encompass intuitionistic negation is straightforward. The language of logical formulae is extended with the nullary symbol $\perp$ to mean *false* and the unary negation symbol $\neg$. Formula $\neg A$ is synonymous with $A \supset \perp$. The extension is done in two steps:

i) First, the closure rule is extended to sequents where $\perp$ occurs as an assumption:

$$\boxed{\perp}, \Gamma \vdash C \;\rightarrow\; \overline{\perp, \Gamma \vdash C}$$

ii) Two rules that are immediate consequences of the definition of $\neg$ are added:

$$\neg\ \mathit{left} : \frac{\neg A, \Gamma \vdash \boxed{A}}{\neg\boxed{A}, \Gamma \vdash C} \qquad\qquad \neg\ \mathit{right} : \frac{\boxed{A}, \Gamma \vdash \perp}{\Gamma \vdash \neg\boxed{A}}$$

These rules have the good properties of the rules of Figure 2 with respect to proof by pointing. The first rule is included for completeness because in practice, it is rarely used.

Let us play with a few examples:

Goal: $\vdash x \supset \neg\neg x$

Proof: Clicking on the rightmost x proves this proposition.

$$\vdash x \supset \neg\neg\underline{x} \;\rightarrow\; x \vdash \neg\neg\underline{x} \;\rightarrow\; \neg\underline{x}, x \vdash \perp \;\rightarrow\; \neg x, x \vdash \underline{x} \;\rightarrow\; \overline{\neg x, x \vdash x}$$

Goal: $\vdash (x \vee \neg y) \wedge y \supset x$

Proof: We click on the leftmost x to distinguish two cases

$$\vdash (\underline{x} \vee \neg y) \wedge y \supset x \;\rightarrow\; (\underline{x} \vee \neg y) \wedge y \vdash x \;\rightarrow\; \underline{x} \vee \neg y, y, (x \vee \neg y) \wedge y \vdash x$$

We are left with the two goals:

$$\underline{x}, x \vee \neg y, y, (x \vee \neg y) \wedge y \vdash x$$

$$\neg y, x \vee \neg y, y, (x \vee \neg y) \wedge y \vdash x$$

The first goal is closed automatically. In the second goal, we see both y and $\neg y$ in the assumptions, and express this with one click inside $\neg y$:

$$\neg\underline{y}, x \vee \neg y, y, (x \vee \neg y) \wedge y \vdash x \;\rightarrow\; \neg y, x \vee \neg y, y, (x \vee \neg y) \wedge y \vdash \underline{y}$$

$$\rightarrow \overline{\neg y, x \vee \neg y, y, (x \vee \neg y) \wedge y \vdash y}$$

Goal: $\vdash \neg\neg(x \vee \neg x)$
Proof: surprisingly, this fact is proved with two judiciously placed clicks.

$$\vdash \neg\neg(x \vee \neg \underline{x}) \;\rightarrow\; \neg(x \vee \neg \underline{x}) \vdash \bot \;\rightarrow\; \neg(x \vee \neg x) \vdash x \vee \neg \underline{x} \;\rightarrow\; \neg(x \vee \neg x) \vdash \neg \underline{x}$$
$$\rightarrow\; \underline{x}, \neg(x \vee \neg x) \vdash \bot$$

The process stops, and we click now on the second occurrence of x to reveal a contradiction:

$$x, \neg(\underline{x} \vee \neg x) \vdash \bot \rightarrow x, \neg(x \vee \neg x) \vdash \underline{x} \vee \neg x \rightarrow x, \neg(x \vee \neg x) \vdash \underline{x} \rightarrow \overline{x, \neg(x \vee \neg x) \vdash x}$$

2.5 Classical negation

Following Gentzen [Szabo69], we introduce classical negation with a meta mechanism, (implemented by a button in the interface for example) that asks the user for a formula A and adds the formula $A \vee \neg A$ to the hypotheses of the current goal. Alternatively, but in the same spirit, we can keep the following higher order theorem in the environment:

$$\textit{excluded middle} \;:\; \forall P.\, P \vee \neg P$$

and invoke it using the general mechanism described in the next section.

2.6 Theorems

In any serious proof, one reuses one or more theorems, i.e. results that have been proved earlier. One can use the mouse to invoke theorems in a manner that is consistent with proof by pointing. Assume there is currently a goal where the selection resides, and somewhere a list of theorem statements[3] containing the desired theorem $\vdash T$. The idea is to click within T directly. If the goal was

$$\Gamma \vdash B$$

it becomes

$$\Gamma, \boxed{T} \vdash B$$

The validity of this inference is an immediate consequence of the *cut* rule:

$$\textit{cut} : \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}$$

Consider the example:
Goal: $\vdash \neg\neg x \supset x$
Proof: invoke $\textit{excluded middle} : \;\forall P.\, \underline{P} \vee \neg P$ for x. We get immediately two cases:

$$\underline{x} \vdash \neg\neg x \supset x$$
$$\neg x \vdash \neg\neg x \supset x$$

The first case is solved by clicking on the rightmost x, and in the second case clicking on the inner $\neg x$ makes a contradiction apparent.

[3] How this list is created is interesting *per se*, but immaterial here.

2.7 Correctness and completeness

Proof by pointing is correct in a given logic as soon as the rules of Figure 1 are provable in that logic. The rules are a close variant of familiar systems, such as system IS used in [Sundholm83]. Thus, proof by pointing is correct in any logic that is an extension of intuitionistic logic. This category encompasses most practical logics.

Proof by pointing is complete if it makes it possible to prove all formulas that are provable with other interfaces. Here again, completeness will depend on the logic implemented by the proof system. A sufficient condition to obtain this property is to have completeness when clicking only at depth 1 and 2. This is a cramped style of proof, but it makes all proofs possible.

To keep the spirit of proof by pointing, we suggest the following principle. Let e be an expression occurrence at depth n and call *thread* of e the list of occurrences at depth $1, 2, \ldots, n$ containing e. There should be no difference for the user between clicking *directly* on e or stepping leisurely through the thread of e.

The issue of completeness will arise again in the paper in section 4. To make proofs less cumbersome, we will want to dispose of certain assumptions. In doing so, one must be careful not to sacrifice completeness.

3 Implementation

Proof by pointing has been implemented for several proof assistants: Felty's theorem prover [Felty89], Coq [Coq91], HOL[HOL88] and Isabelle [Isa90]. We used the Centaur system [Centaur93] as a toolkit for building interfaces, mostly for the ability to point at a subexpression with the mouse. As a matter of principle, we tried not to modify the proof engines at all. This is easier if the user is allowed to program his own proof tactics. This section discusses a number of little difficulties and the solutions we propose.

3.1 Basic tactics

The implementation attempts to follow exactly the description of the previous section. The first question then is to identify the commands that implement the inference rules of Figure 1. Most proof systems provide the ingredients to do that, but sometimes a tactic may be eager to perform several steps at once. For example, the Coq system provides an `Apply` command for eliminating implications. Given an assumption `H` of the form $A \supset (B \supset C)$, the tactic `Apply H` succeeds only if the goal's conclusion matches C. Then it performs a closure and two $\supset$ $left_2$, creating two goals with conclusion A and B respectively. In most cases, this tactic is very effective for a human user with a conventional interface. But it is partial to backwards reasoning and it does not allow us to implement the $\supset$ *left* rules correctly. A more atomic `Use` tactic was added to Coq to solve the difficulty.

The tactics for the $\exists$ *right* and $\forall$ *left* rules raise another problem. When implementing these rules the expression e that appears in the new subgoal has to be provided. Some proof engines require this term as an argument of the tactic and we then have to provide interface tools to supply this argument. Other systems introduce an unknown, basically a logical variable, that can be instantiated later.

This mechanism is particularly useful in the context of an extension of the closure rule. When the selection points to an assumption, we might ask for it to be *unified* with the goal's conclusion, i.e. to compute the instantiation of the unknowns that will allow one to use the closure rule. Similarly, if the selection points at the goal's conclusion, we may search for an assumption that unifies with it. The problem with applying this method systematically is that there may be several assumptions that match differently the goal. An arbitrary choice may preclude proving another goal.

3.2 Management of the selection

The next issue has to do with managing the selection. It is not immediately obvious how to do this because proof assistants do not have this notion and we certainly don't want to modify their private data structures.

All systems have means to perform logical operations in a goal's conclusion. They vary much more in their ways of acting on assumptions. In Coq, assumptions are named and very easy to handle. In other systems, one may refer to assumptions by rank or by content. Referring to an assumption by content means that the whole text of the assumption will occur as an argument in a command. This is safe, but commands may become huge, slowing down execution. Referring to assumptions by rank makes for shorter commands but it is less robust. For example, in Isabelle we had to use predictable but undocumented features of the system to implement properly the selection management.

After applying the basic tactics, we must indicate where the selection has propagated. If we have control over naming assumptions as in Coq, this is very easy to implement. In the cases where we use rank or content to refer to assumptions, there is an alternative solution. The idea is to single out, in the rules of Figure 2, the parameter that contains the residual selection. This could be done with a three-place sequent $\Gamma \vdash A : C$ where A is the distinguished assumption. In fact, this three-place sequent can be coded with the existing two-place sequent as $\Gamma \vdash A \supset C$. The rules of Figure 2 have been rewritten on Figure 3 using that encoding.

The proof by pointing algorithm works now as follows:

Step 1. If the initial selection is inside an assumption, then move this assumption to the distinguished position in the conclusion with the rule $\supset$ *right*$^{-1}$:

$$\supset right^{-1} : \frac{\Gamma \vdash \boxed{A} \supset B}{\boxed{A}, \Gamma \vdash B}$$

$$\wedge\, \mathit{left}_1 : \frac{B, A \wedge B, \Gamma \vdash \boxed{A} \supset C}{\Gamma \vdash \boxed{A} \wedge B \supset C} \qquad \wedge\, \mathit{right}_1 : \frac{\Gamma \vdash \boxed{A} \qquad \Gamma \vdash B}{\Gamma \vdash \boxed{A} \wedge B}$$

$$\wedge\, \mathit{left}_2 : \frac{A, A \wedge B, \Gamma \vdash \boxed{B} \supset C}{\Gamma \vdash A \wedge \boxed{B} \supset C} \qquad \wedge\, \mathit{right}_2 : \frac{\Gamma \vdash A \qquad \Gamma \vdash \boxed{B}}{\Gamma \vdash A \wedge \boxed{B}}$$

$$\vee\, \mathit{left}_1 : \frac{A \vee B, \Gamma \vdash \boxed{A} \supset C \qquad B, A \vee B, \Gamma \vdash C}{\Gamma \vdash \boxed{A} \vee B \supset C} \qquad \vee\, \mathit{right}_1 : \frac{\Gamma \vdash \boxed{A}}{\Gamma \vdash \boxed{A} \vee B}$$

$$\vee\, \mathit{left}_2 : \frac{A, A \vee B, \Gamma \vdash C \qquad A \vee B, \Gamma \vdash \boxed{B} \supset C}{\Gamma \vdash A \vee \boxed{B} \supset C} \qquad \vee\, \mathit{right}_2 : \frac{\Gamma \vdash \boxed{B}}{\Gamma \vdash A \vee \boxed{B}}$$

$$\supset \mathit{left}_1 : \frac{A \supset B, \Gamma \vdash \boxed{A} \qquad B, A \supset B, \Gamma \vdash C}{\Gamma \vdash (\boxed{A} \supset B) \supset C}$$

$$\supset \mathit{left}_2 : \frac{A \supset B, \Gamma \vdash A \qquad A \supset B, \Gamma \vdash \boxed{B} \supset C}{\Gamma \vdash (A \supset \boxed{B}) \supset C} \qquad \supset \mathit{right}_2 : \frac{A, \Gamma \vdash \boxed{B}}{\Gamma \vdash A \supset \boxed{B}}$$

$$\forall\, \mathit{left} : \frac{\forall x\, A, \Gamma \vdash \boxed{A[x \backslash e]} \supset C}{\Gamma \vdash \forall x \boxed{A} \supset C} \qquad \forall\, \mathit{right} : \frac{\Gamma \vdash \boxed{A[x \backslash c]}}{\Gamma \vdash \forall x \boxed{A}}$$

$$\exists\, \mathit{left} : \frac{\exists x\, A, \Gamma \vdash \boxed{A[x \backslash c]} \supset C}{\Gamma \vdash \exists x \boxed{A} \supset C} \qquad \exists\, \mathit{right} : \frac{\Gamma \vdash \boxed{A[x \backslash e]}}{\Gamma \vdash \exists x \boxed{A}}$$

$$\neg\, \mathit{left} : \frac{\neg A, \Gamma \vdash \boxed{A}}{\Gamma \vdash \neg \boxed{A} \supset C} \qquad \neg\, \mathit{right} : \frac{\Gamma \vdash \boxed{A} \supset \bot}{\Gamma \vdash \neg \boxed{A}}$$

Fig. 3. Modified rules for selection propagation

Step 2. Use the rules of Figure 3. The rules work on a goal's conclusion and keep the selection in the conclusion. When they are not applicable any longer, either (i) the selection is on the whole conclusion, or (ii) the conclusion is of the form $A \supset B$ and the selection is on A exactly.

Step 3. In case (i) try closure as usual, in case (ii) apply $\supset \mathit{right}_1$ and then attempt closure.

The rules of Figure 3 are unambiguous because $\supset \mathit{right}_1$, which would conflict with all the *left* rules, has been removed. Rule $\supset \mathit{right}_1$ is only applied at Step 3 in the algorithm.

The correctness of the rules of Figure 3 is immediate because they have been obtained from the rules of Figure 2 using $\supset \mathit{right}^{-1}$, which is a derived rule as soon as the cut rule is in the system. For intuitionistic negation, one simply uses the definition of $\neg A$ as $A \supset \bot$ to obtain two rules that fit very well with the rest of the system.

3.3 Compound tactics

When selecting an expression at a depth greater than 1, we have to combine basic tactics into a compound one. For this, we use the proof assistant's *tacticals* [Paulson87]. The principal problem is to direct the choice of the goal to attack next, and it is solved differently if different tacticals are available. In the ideal situation, there exists a **THENL** tactical [Paulson87]. In systems like HOL, Coq or Felty's theorem prover, tactics are functions that take a subgoal as argument and return a list of subgoals. The **THENL** tactical enables composing tactics in the following manner: assume that we attack subgoal σ with tactic `t`, which gives rise to new sub-goals $\sigma_1, \ldots, \sigma_n$ and that for every i, we attack sub-goal σ_i with tactic $\mathtt{t}_i$, which gives rise to new sub-goals $\sigma_i^1, \ldots, \sigma_i^{p_i}$. The tactic expression $\mathtt{t\ THENL\ [t_1;\ldots;t_n]}$ denotes the compound tactic that attacks subgoal σ and produces the subgoals $\sigma_1^1, \ldots, \sigma_1^{p_1}, \ldots, \sigma_n^1, \ldots \sigma_n^{p_n}$.

For example, suppose that the user selects expression B in the following sub-goal:

$$\Gamma \vdash A \wedge (\underline{B} \vee C)$$

This results in performing two basic tactics: first applying rule $\wedge\, right_2$, then applying $\vee\, right_1$ on the second of the resulting subgoals. The tactic to generate is simply the following one:

$$\wedge\, right_2 \ \mathtt{THENL}\ [\textit{do-nothing}\ ;\ \vee\, right_1]$$

The two goals that would be generated by the first basic tactic alone are $\Gamma \vdash A$ and $\Gamma \vdash B \vee C$ and the goals generated by the complete tactic are $\Gamma \vdash A$ and $\Gamma \vdash B$.

In most proof development systems, the **THENL** tactical is already present or is easily implemented. A notable exception is the Isabelle system, where the current state of an incomplete proof is represented as a theorem whose hypotheses are the remaining subgoals. Tactics basically are functions that map the state (a theorem) to a new theorem. In this setting, there is no clear notion of what new subgoals have been generated from old ones, and the **THENL** tactical loses a lot of its meaning. However, some tactics take a rank n as argument, specifying that these tactics work on the n^{th} sub-goal. The system provides a simple **THEN** tactical, that permits chaining tactics, independently of the subgoal they are working on: if $\mathtt{t}_1$ and $\mathtt{t}_2$ are two tactics that take rank arguments, the expression $(\mathtt{t}_1\ \mathtt{n})\ \mathtt{THEN}\ (\mathtt{t}_2\ \mathtt{p})$ denotes performing $\mathtt{t}_2$ after $\mathtt{t}_1$ but the programmer must take care of the relation between ranks `n` and `p` to make sure that $\mathtt{t}_2$ works on a sub-goal generated by $\mathtt{t}_1$.

For example, suppose again that subgoal n is the formula $\Gamma \vdash A \wedge (B \vee C)$ and that the user selects expression B. The compound tactic to use is the following one:

$$(\wedge\, right_2\ \ n)\ \mathtt{THEN}\ (\vee right_1\ \ n+1)$$

Note that the correct index had to be computed for the second basic step of the tactic.

3.4 Synthesizing the command from the graphic selection

The last point to discuss is the method for generating a compound tactic from a graphic selection. Indeed, proof by pointing is of little use if the operation of selecting a subterm is cumbersome. The ideal solution is to use a finger or a mouse to select a subterm. This implies that the interface component keeps track of the underlying term structure of the formulas being displayed. In our experiments, we have built user-interfaces following the methodology advocated in [TBK92]. The interface is a separate process that knows about the syntactic structure of logical expressions and is able to get at a subexpression with a single click and to construct an abstract selection describing the location of this subexpression relative to the complete goal where it occurs.

There are now two situations: either (i) the proof assistant has a fixed set of tactics or (ii) it has a provision for defining new tactics. In case (i), the interface generates the appropriate command line. This has the advantage that the command is readable. On the negative side, this command is possibly very large.

If the proof assistant provides for user-defined tactics, then it is better to define a new tactic `finger_tac` that takes directly an abstract selection as argument and performs the relevant combination of basic tactics. The script of the proof will now contain a single call to `finger_tac` with a somewhat opaque looking argument. On the positive side, `finger_tac` is entirely implemented in the proof assistant.

To represent an abstract selection, the standard solution is to use a list of integers that describes the path from the root of the goal to the selected subterm. This solution has the drawback that it depends on the internal representation of terms and that the generated commands contain absolutely obscure sequences of integers. An alternative is to have a specialized, still more abstract notion of path for our application. First, we consider that all connectives $\wedge$, $\vee$, $\supset$, $\forall$ and $\exists$ are binary, independently of the actual representation of these terms in the proof assistant. So for the term $(B \vee C) \supset A$, the path to C is denoted `[1;2]` Then we include additional information as in [Boudol85] by prefixing the integers with the name of the appropriate connective. In our previous example, the path to C becomes `[(⊃ 1);(∨ 2)]`. In this way, the notion of path is independent and of the exact abstract syntax used by the interface, and of the exact abstract syntax used by the proof assistant.

4 Extensions

Using extensively proof by pointing with various proof systems suggests many extensions to this paradigm. While these extensions do not necessarily fall in line with the formal foundations of the principle and the requirement of completeness, they help in making a more user-friendly interface to those systems. The extensions are only sketched here, because clearly more experience is needed.

4.1 Reducing the number of assumptions

In Figure 1, all *left* rules tend to add a new assumption in the context for at least one of the new subgoals, while no *right* rule removes any assumption from it. In long proofs, relevant assumptions tend to get hidden among many useless assumptions.

An alternative set of rules, given in Figure 4, attacks this problem by consuming assumptions as they are being used. Practically, this ensures that the number of assumptions will not grow, but some formulas are not provable any longer. One solution is to use both behaviors, associating each one to a different button of the mouse.

$$\wedge\, \mathit{left} : \frac{A, B, \Gamma \vdash C}{A \wedge B, \Gamma \vdash C} \qquad \wedge\, \mathit{right} : \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$\vee\, \mathit{left} : \frac{A, \Gamma \vdash C \qquad B, \Gamma \vdash C}{A \vee B, \Gamma \vdash C} \qquad \vee\, \mathit{right} : \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$\supset \mathit{left} : \frac{\Gamma \vdash A \qquad B, \Gamma \vdash C}{A \supset B, \Gamma \vdash C} \qquad \supset \mathit{right} : \frac{A, \Gamma \vdash B}{\Gamma \vdash A \supset B}$$

$$\forall\, \mathit{left} : \frac{A[x \backslash e], \Gamma \vdash C}{\forall x\, A, \Gamma \vdash C} \qquad \forall\, \mathit{right} : \frac{\Gamma \vdash A[x \backslash c]}{\Gamma \vdash \forall x\, A}$$

$$\exists\, \mathit{left} : \frac{A[x \backslash c], \Gamma \vdash C}{\exists x\, A, \Gamma \vdash C} \qquad \exists\, \mathit{right} : \frac{\Gamma \vdash A[x \backslash e]}{\Gamma \vdash \exists x\, A}$$

Fig. 4. Linear rules for the logical connectives

A better solution consists in tracking more carefully where the consumed assumptions come from, so as to consume them only when they can be regenerated from the context if necessary.

4.2 Induction

In many theories, such as Peano arithmetic, proving universally quantified formulas is not done exclusively with the $\forall$ *right* rule, but also with induction rules like the following one:

$$\frac{\Gamma \vdash P(0) \qquad P(n), \Gamma \vdash P(n+1)}{\Gamma \vdash \forall \boxed{n : \mathit{int}}\, P(n)}$$

When selecting inside a universally quantified formula, we mean to use the $\forall$ *right* rule. To obtain the induction rule, one idea is to select the typed bound variable, as shown. Since this expression has no subterms, the induction rule is terminal, i.e. it is the last one in a thread. There is no obvious choice for the propagation of the selection at this moment. Grouping together a number of universally quantified variables under the umbrella of a single $\forall$ might give a simple way to ask for several simultaneous recursions.

4.3 Equality

Equality has a special status in mathematics. Many operator definitions and properties are given using equality and replacing equals for equals is a pervasive form of reasoning. Many theorems have the form of a universally quantified implication, whose conclusion is an equality. When using such a theorem, one would like to have more than one selection: one to tell where in the current goal it is wished to apply it, one to tell whether to use the equality from left to right or from right to left. Several computer algebra ([Bonadio89, Paracomp88]) systems have begun experimenting with this type of ideas whose validity must be assessed.

4.4 Point and Shoot

The idea of proof by pointing is to bring a selected subexpression to the surface and then try finishing the proof of the resulting goal by a closure. This may fail, but most proof assistants provide a collection of tactics that work on subexpressions at the surface. As a result, users eventually end up alternating between proof by pointing to bring subexpressions of interest to the surface and applying regular tactics that manipulate these subexpressions.

We propose to integrate seamlessly both actions by providing users with the possibility to combine in a single command the selection of a subexpression and the choice of a tactic to manipulate this subexpression. With this new paradigm, nicknamed *point and shoot*, we preserve the full power of both approaches and really reduce the number of user interactions. Note that proof by pointing as presented in this paper is now a special case of *point and shoot* where the tactic to apply only performs closures.

Revisiting the problem of invoking induction rules, it seems more elegant to handle this using the point and shoot paradigm together with tactics that are specialized in inductive reasoning, rather than invent an *ad hoc* meaning for selecting the bound variable in a universally quantified formula. We have implemented point and shoot for Coq.

4.5 Finding other Domains of Application

This paper shows that the idea of using the mouse to guide computer activity is well understood in the realm of proofs and simple logics. It is reasonable to look for possible extensions. There are two obvious extension directions:

1. Extending to other logics, either more elaborate ones like temporal or modal logic, or more restrictive ones like linear logic. Obviously, the notion of focus used in this paper does not depend too closely on the exact set of rules used in the logic.
2. Extending to other behaviors than strict goal directed proof. There are obvious similarities with tools like *window inference* [Grundy91] that aim at supporting deduction through formula or program transformation.

5 Conclusion

In our experiments with proof assistants, we have noticed that similar proofs done in different systems seem quite different due to specific features of these systems: syntax of the logical language, nomenclature of the theorems, structure of the command language. This diversity increases, for no valid reason, the difficulty of learning how to use a new proof assistant. Proof by pointing provides a uniform approach for the basic and frequent logical manipulations. Proving simple propositions tends to become identical in HOL, Isabelle and Coq. For example, the fact proved in the introduction requires similar sequences of selections in all systems. Differences between proof assistants are then concentrated on more important issues.

The rules on Figure 1 represent very elementary manipulations in comparison to what we use in every day mathematics. For example, representing $\wedge$ and $\vee$ as binary connectives makes accessing an element in a disjunction or a conjunction *non-atomic* and dependent on the exact structure of terms. In an assumption such as $A \wedge B \wedge C$, to access A (or C depending on the associativity of $\wedge$) requires two eliminations of an $\wedge$ operator, i.e., two applications of the $\wedge$ *left* rules. Similarly, if we have two assumptions $p(a)$ and $\forall x\, p(x) \supset q(x)$, deducing $q(a)$ which is intuitively an atomic operation requires an explicit application of several rules. The proof by pointing mechanism restores this natural idea of atomicity.

Another advantage of proof by pointing is that it accomodates without need for any particular mental process both a forwards and a backwards style. Working in assumptions or in the conclusion leads to two different styles of proof usually called forwards and backwards. The forwards style consists in starting from what we know, to reach what we want to prove. In backwards style, we start from what we want to prove, determine what we need to infer it and so on recursively. The backwards style tends to be privileged by proof development systems, but humans, and computer algebra systems use the forwards style constantly as well. With proof by pointing, the user makes no permanent commitment to one style or the other. Every mouse selection gives an opportunity for a change in direction.

Acknowledgments The phrase "proof by pointing" comes from [Ritchie88], where it is limited to pointing at expressions at depth 1 and 2. The authors want to thank A. Felty for making initial experiments feasible with her proof assistant, and P. Anderson for help in improving this paper.

References

[Bonadio89] A. Bonadio, E. Warren. *Theorist Reference Manual*, Prescience Corp. 814 Castro St. San Francisco, 1989

[Boudol85] G. Boudol "Computational semantics of term rewriting systems", in *Algebraic Methods in Semantics*, M. Nivat, J. C. Reynolds *eds.*, Cambridge University Press, 1985.

[Centaur93] "The Centaur 1.3 Manual", I. Jacobs, *ed.*, available from INRIA-Sophia-Antipolis, January 1993.

[Coq91] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin-Mohring, B. Werner, *The Coq Proof Assistant User's Guide*, INRIA Technical Report no. 134, December 1991.

[Felty89] A. Felty, *Specifying and Implementing Theorem Provers in a Higher-Order Logic Programming Language*, PhD Thesis, University of Pennsylvania, August 1989.

[Grundy91] J. Grundy, "Window Inference in the HOL System", in *Proceeding of the 1991 International Workshop on the HOL Theorem Proving System and its Applications*, M. Archer, J. J. Joyce, K. N. Levitt, P. J. Windley, *eds.*, IEEE Computer Society Press, 1991.

[HOL88] M.J.C. Gordon, "HOL: A Proof Generating System for Higher-Order Logic", in *VLSI Specification, Verification and Synthesis*, G. Birtwistle, P. A. Subrahmanyam, *eds.*, Kluwer Academic Publishers, 1988.

[Isa90] L.C. Paulson, "Isabelle: The next 700 theorem provers", in *Logic and Computer Science*, P. Odifreddi, *ed.*, pp. 361–386, Academic Press, 1990.

[Maple] B. W. Char *et al.*, *MAPLE : reference manual : 5th edition*, Springer-Verlag, 1992.

[Nuprl86] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, J.T.Smith, *Implementing Mathematics with the Nuprl Proof Development System* Prentice-Hall, 1986.

[Paracomp88] Paracomp Inc. *Milo User's Guide*, 123 Townsend St., Suite 310, San Francisco, 1988.

[Paulson87] L. Paulson, *Logic and computation : interactive proof with Cambridge LCF*, Cambridge University Press, 1987.

[Sundholm83] G. Sundholm, "Systems of Deduction", in *Handbook of Philosophical Logic, Vol. I*, D. Gabbay and F. Guenthner, *eds.*, pp. 133–188, D. Reidel Publishing Company, 1983

[Ritchie88] B. Ritchie, *The design and implementation of an interactive proof editor*, PhD Thesis, University of Edinburgh, Nov. 1988. G. Sundholm, "Systems of Deduction", in *Handbook of Philosophical Logic*, Vol. I, D. Gabbay, F. Guenthner, *eds.*, D. Reidel Publishing Company, 1983.

[Szabo69] M.E. Szabo, G. Gentzen, *The Collected papers of Gerhard Gentzen*, North-Holland, 1969.

[TBK92] L. Théry, Y. Bertot, G. Kahn, "Real Theorem Provers Deserve Real User-Interfaces", in *Proceedings of the Fifth ACM SIGSOFT Symposium on Soft-*

ware Development Environments, Tyson's Corner, Va, USA, Software Engineering Notes, Vol. 17, no. 5, ACM Press, 1992.
[Matica] S. Wolfram, *Mathematica : a system for doing mathematics by computer*, Addison-Wesley, 1988.