When is *x* distinct from *y*?
Substitution in Jape – the not-so-obvious bits

Richard Bornat, Department of Computer Science,
Queen Mary and Westfield College, University of London
Bernard Sufrin, Programming Research Group, University of Oxford

April 1996

0 Abstract

Jape is a proof editor/calculator dedicated to animating logical proof at the surface level. Its glory is its user interface[1,2], but this paper concentrates on the the internal mechanisms which support surface-level proof – the mechanisms of its proof engine. The work of the proof engine is unification, and the definition of rules and theorems can include explicit substitution formulæ. The mechanisms discussed here are those used to support proof of theorem schemata – effectively, derived rules – in the presence of explicit substitution formulæ. We include some speculation on future steps we intend to take which should improve Jape's interface still further.

1 Background

It's normal to present rules of inference as rule schemata, to write

$$\frac{\Gamma, P, Q \vdash \Delta}{\Gamma, P \wedge Q \vdash \Delta}\ \wedge\vdash$$

and to understand that the rule applies to any collection of formulæ Γ and Δ, any formula P and any formula Q. Jape's internal mechanism of unification supports this interpretation: when this rule is applied an instantiated version is created in which the names Γ, Δ, P and Q are replaced by freshly-minted unknowns, and then the consequent $_\Gamma, _P \wedge _Q \vdash _\Delta$ is unified with a problem sequent from the proof[1].

It is natural to wish to use theorems of the logic in just the same way: if there is a proof of the theorem schema $P, P \to Q \vdash Q$ then that schema should be applicable, as a closing rule, to any problem sequent which can unify with $_P, _P \to _Q \vdash _Q$[2]: that is justified because, using the rules of the single-conclusion sequent calculus, the proof tree schema

$$\frac{\dfrac{}{P, P \to Q \vdash P}\ hyp \qquad \dfrac{}{P, Q \vdash Q}\ hyp}{P, P \to Q \vdash Q}\ \to\vdash$$

remains valid no matter what formulæ are substituted for P and Q[3].

Neither of the proof rules used in the tree above involve a substitution formula. Proof trees which use of rules which do involve substitution formulæ are more difficult to use in schematic proofs. At its heart this

---

1 At the time of writing Jape does not treat the collections Γ and Δ in this way, but its treatment has the same effect. In the future we intend that Jape should work in just the way described here.

2 Jape implicitly uses weakening rules, if they are present in a logic, to permit a theorem schema to match problem sequents with additional, unmatched, hypotheses and/or conclusions, but that doesn't affect the argument here.

3 The rules used in this proof are summarised in appendix A.

is a matter of the way that we deal with bindings: computer scientists will recognise the problem immediately if we say that we must prevent Jape from accepting $\lambda z.\lambda z.z : int \to real \to int$ as an instance of the theorem $\lambda x.\lambda y.x : T1 \to T2 \to T1$, but the problem doesn't stop there, as we shall see.

The difficulty is illustrated in the following proof tree, which again uses the rules of the single-conclusion sequent calculus:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(c,d)\vdash P(c,d)}\,hyp}{P(c,d)\vdash \exists x.P(x,d)}\,\vdash\exists}{\forall y.P(c,y)\vdash \exists x.P(x,d)}\,\forall\vdash}{\forall y.P(c,y)\vdash \forall y.\exists x.P(x,y)}\,\vdash\forall}{\exists x.\forall y.P(x,y)\vdash \forall y.\exists x.P(x,y)}\,\exists\vdash$$

The last step of the proof uses the $\exists\vdash$ rule

$$\dfrac{\Gamma, Q[z\backslash e]\vdash C}{\Gamma, \exists z.Q\vdash C}\ (\text{FRESH } e)\ \exists\vdash$$

and in the tree as displayed the simplification of $(\forall y.P(x,y))[x\backslash c]$ to $\forall y.P(c,y)$ is implicitly included as part of that step, because a substitution formula in a rule is interpreted as standing for the formula to which it can be simplified. It hardly seems necessary to draw attention to that part of the step.

But n order to be able correctly to make that particular simplification in this particular *schematic* proof – one in which the names x, y and P are schematic – it is necessary to be certain that in any instance of the theorem, and hence any instance of the proof tree, the name substituted for x is distinct from the name substituted for y, and also that the formula substituted for P[1] doesn't include any free occurrences of the name substituted for x. At present Jape provides that assurance via explicit provisos: the proof above would necessarily be annotated with the provisos x NOTIN y, x NOTIN P and y NOTIN P.

The first of the three provisos does no more than ensure that the binding structure evident in the statement of the theorem schema is preserved in any valid instance. To put it another way: it ensures that $\exists z.\forall z.A(z)(z,z)\vdash \forall z.\exists z.A(z)(z,z)$ isn't accepted as an instance of the theorem schema $\exists x.\forall y.P(x,y)\vdash \forall y.\exists x.P(x,y)$ if we rely on the schematic proof given above. The other two provisos have to do with the treatment, or rather non-treatment, of predicate notation in the current version of Jape: $P(x,y)$ is an unterpreted formula, so that $(P(x,y))[x\backslash c]$ is equivalent to $P[x\backslash c](x[x\backslash c], y[x\backslash c])$.

What isn't clear from this example is that the problem isn't just a matter of careful choice of schematic variables, of confusion about the interaction between alpha-conversion and instantiation, nor even simply a matter of a proper treatment of predicate notation. The rest of this paper explains what the problem is, how Jape solves it at present, and how it might solve it more elegantly in the future.

2 Argument formulæ in rule and theorem application

It is helpful when applying a rule or a theorem schema to allow the user to provide argument formulæ to describe the particular instance which is to be used and so to override the automatic instantiation mechanism. Most obviously this is helpful when a rule doesn't have the subformula property, as for example in the case of single-conclusion sequent calculus *cut*:

$$\dfrac{\Gamma\vdash B \quad \Gamma, B\vdash C}{\Gamma\vdash C}\ cut$$

Giving an argument formula to use in place of B avoids the introduction of an unknown to the proof, and that can sometimes make the progress of the proof much easier to understand.

---

[1] Jape doesn't have any treatment of predicate notation at the time of writing. Later in this paper we outline a proposal to remedy this defect.

Where a rule or a theorem includes an explicit substitution in its consequent as, for example, in the ∀ elimination rule of a sequent presentation of natural deduction:

$$\frac{\Gamma \vdash \forall x.P}{\Gamma \vdash P[x\backslash E]}\ \forall\,\text{elim}$$

it is very helpful to give an argument formula to use in place of the substituted formula E, so that Jape can construct a substitution which will simplify into the relevant component of the problem sequent. The process is made more straightforward still if the name x is not schematic and Jape can replace it with a fresh name x' in every instance.

3 α-conversion in rule and theorem application

Jape is capable of α-conversion: $\forall x.\, formula_1$ unifies with $\forall y.\, formula_2$ if $formula_1[x \setminus w]$ unifies with $formula_2[y \setminus w]$ for freshly-minted variable name w in the presence of the provisos w NOTIN $formula_1$ and w NOTIN $formula_2$ which guard against any possible bad effects of future unifications., but it isn't always used in the matching of instances of rules or theorems to problem sequents. Given appropriate syntactic declaration of binding structures, and assurances or evidence that x and y don't occur free in a formula P, Jape will unify $\forall x.P(x)$ with $\forall y.P(y)$. That would seem to solve the problem illustrated above: make every variable that occurs in a binding position in a theorem schema non-schematic, and unify binding structures always by α-conversion.

There are two problems with this proposal. First, it isn't always convenient. Consider, for example, the ⊢∀ rule:

$$\frac{\Gamma \vdash P[x\backslash c]}{\Gamma \vdash \forall x.P}\ (\text{FRESH}\, c)\ \vdash\forall$$

If we forget about α-conversion and make the rule schematic in x, then matching it to a problem sequent $\Delta \vdash \forall y.Q$ by instantiation and unification is extremely simple. In order to complete the proof step the only problem then is to simplify, if possible, the substitution formula $Q[y\backslash c]$ in the antecedent.

If we make x non-schematic then various approaches are possible. The most straightforward uses the same name x in every instance of the rule: then to unify $\forall x._P$ with $\forall y.Q$ means unifying $_P[x\backslash x']$ with $Q[y\backslash x']$ where x' is a fresh variable. That's by no means a simple task, and though Jape will have a stab at it, it often misses. It makes more sense to use a fresh name x' each time the rule is used: then we only have to unify $_P$ with $Q[y\backslash x']$, which is perfectly straightforward. To complete the proof step, however, Jape has to simplify the formula $(Q[y\backslash x'])[x'\backslash c]$;that will simplify at least as far as $Q[y, x'\backslash c, c]$ but in very many circumstances it won't be possible to eliminate x' from the result, and an additional name will have been unnecessarily introduced to the proof.

The second problem with the proposal to use α-conversion is that it doesn't always work. If a name occurs in a consequent of a rule only in binding and bound occurrences – as it does in the rule above, for example – then α-conversion is a reasonable way of matching the rule to a problem sequent. But we may be confronted, in some bizarre logic, with a rule or a theorem in which the same name occurs both binding, bound and free. Jape can't be committed to any particular view of what constitutes a 'reasonable' use of names.

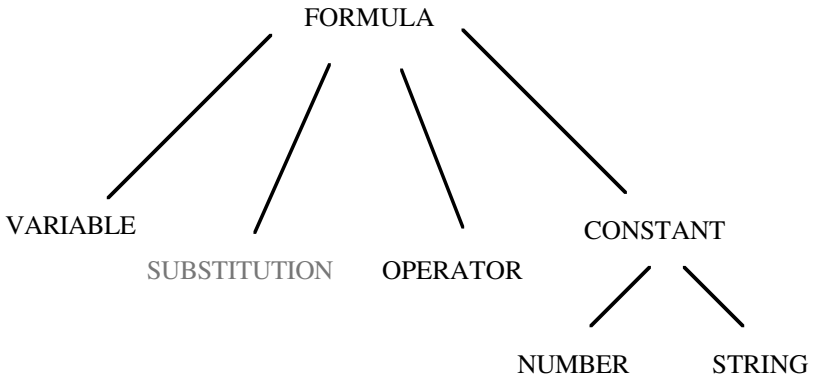
If we sometimes don't want to use α-conversion, and we sometimes can't use it, how should we proceed? We commence by describing how Jape handles the problem at present. We note first that to if we mix α-conversion and pattern unification of binding structures then Jape's unification algorithm becomes incomplete, but we observe that in practice this hasn't proved a problem.

4 First approximation: syntactic classes are disjoint, internal names are distinct.

In proving a conjectured theorem schema it would seem that we cannot suppose that distinct schematic names stand for distinct formulae – that x is distinct from y, or that x doesn't occur free in P – unless there is an explicit proviso to say so. So although $x[x\backslash E]$ must always simplify to E, we can't always

simplify $y[x\backslash E]$. These problems are compounded in the case of formulas which include binding instances: $(\forall x.P)[y\backslash Q]$ will simplify to $\forall x.(P[y\backslash Q])$ only if we know from provisos that the schematic names x and y always stand for distinct variables, and also that x can never occur free in Q.

But in most logics we make a distinction between syntactic classes of names: we might reserve x and y for schematic variables, P and Q for schematic formulæ, a and b for schematic constants. Then certainly $a[x\backslash E]$ simplifies to E, because in no instance of a proof schema can a and x be replaced by the same name. The same applies to particular constants, of course: if in a particular logic we encode $F+G$ as $(+)FG$ – that is, as an application of the constant function $(+)$ to arguments F and G – then $(F+G)[x\backslash E]$ simplifies at least to $F[x\backslash E]+G[x\backslash E]$. At present Jape has a single fixed hierarchy of syntactic classes:[1]:



The unification process respects the hierarchy so that, for example, an unknown which is declared to be a NUMBER won't unify with a name declared to be a VARIABLE. Logic encodings can say that a name is schematic (CLASS VARIABLE ...) or simply a particular name in a class (VARIABLE ...), this does away with the need to specify parameters to rules or theorems. The schematic names in a rule are now always evident. This syntactic hierarchy is merely an instance of what we intend eventually to become a user-definable structure of syntact classes, but even so the mechanism reduces the need for provisos considerably.

We make many more provisos unnecessary by assuming that names introduced during a proof, and which aren't introduced to the base sequent of the proof by unification – *internal* names of the proof – are distinct from all schematic names. That is so by virtue of a meta-α-conversion argument: since an internal name is chosen arbitrarily then no matter what arguments might be chosen to correspond to the schematic names, we could always construct a proof tree in which the internal name was chosen to be distinct from any names in the arguments. So if w is an internal name then $w[x\backslash E]$ always simplifies to w, $x[w\backslash E]$ to x, and $(\forall x.P)[w\backslash E]$ to $\forall x.(P[w\backslash E])$ provided that x is certain not to occur free in E, or to $\forall v.(P[x,w\backslash v,E])$ for freshly-minted v, otherwise[2].

Lastly, Jape can invent provisos in some particular circumstances: for example, if it is asked to unify $F[x\backslash c]$ with F it will deduce that this is only possible provided that x NOTIN F.

5 Second approximation: 'independent' names are distinct

Since Jape is capable of α-conversion, and a meta-α-conversion argument is used to justify treating internal names of a proof as distinct from each other and from all schematic names, the next step is to reason about the distinctions between names in the base sequent of a proof tree in the same sort of way.

---

1 SUBSTITUTION is an internal category which allows the unification process to proceed when a substitution can't immediately be reduced: it isn't a category in which a name can be placed.

2 We don't make simplifications which require α-conversion unless they are necessary to complete a unification. To do so outside unification would tend to introduce lots of new names and perhaps even new provisos into a proof It is one thing to do that sort of thing during unification, as the result of a definite user action: it is another to do it arbitrarily elsewhere in a proof tree just to reduce a formula which may reduce more straightforwardly later on. Whatever the truth of the matter, we proceed cautiously.

Some of the schematic names of a proof can be treated as if they were distinct. Suppose, for example, that the statement of a theorem contains formulas $\forall x.P$ and $\forall y.Q$, and suppose that it contains no other occurrences of x, y, P or Q. We can reason that the parameter name x can be treated, during the proof of the theorem, as distinct from y and furthermore we can assume that the argument variable corresponding to x cannot occur free in any argument formula other than the one which corresponds to P; symmetrically y can be treated as distinct from x and as not occurring free in any argument formula other than the one which corresponds to Q. The argument goes as follows:

> Suppose that there is a concrete instance of the theorem produced by replacing x, y, P and Q with u, v, R and S, where u is not distinct from v and/or u occurs other than in $\forall u.R$ and/or v occurs other than in $\forall v.S$. Then that instance is α-convertible into one which contains u', v', R' and S' which do obey the condition and where $\forall u.R$ is α-convertible with $\forall u'.R'$, and $\forall v.S$ with $\forall v'.S'$. The new instance of the theorem will be α-convertible with the old, and it will have a proof tree which is an instance of the one constructed by assuming that x doesn't occur other than in $\forall x.P$ and y doesn't occur other than in $\forall y.Q$.

This argument applies to all kinds of bindings, of course – Jape isn't restricted to the structures of predicate calculus. In particular we use it to handle occurrences of names in the variable position of an explicit substitution formulae as binding occurrences: in $P[x \setminus c]$ P occurs in the scope of a binding occurrence of x. The argument eliminates the need for many otherwise necessary provisos and makes Jape correspond more nearly to the way that people would like to construct formal proofs on paper. If x and y are distinct variable names, then the possibility that they might both be replaced by z in some pathological instance of the theorem schema hardly occurs to the average paper logician and, therefore, Jape shouldn't make a fuss about it unless it has to.

The argument allows us to deduce not only that x and y in the statement of the theorem can be treated as distinct during any proof of the theorem, but also that they can also be treated as independent of any other name z or T which occurs free in the statement of the theorem. Notice that the argument doesn't have anything to do with the rules of the logic in which a proof is made: it only has to do with formula structure and occurrences of names.

Unfortunately the argument doesn't extend as far as you might expect or hope it to. If the statement of a theorem contains $\forall x.P$ as well as a free occurrence of x then we've partially lost control of x: we can't use a meta-α-conversion argument to say that it must be distinct from a name z which occurs free in the statement of the theorem. If z and x are both replaced by the same variable in some instance of the theorem, we can't α-convert them apart. For the same reason, if P occurs also outside the scope of a binding occurrence of x, then the argument variable corresponding to x may occur free as well as in a binding position, and we have lost control of x in just the same way. Similarly, we have lost control of both x and y if the theorem statement contains both $\forall x.P$ and $\forall y.P$. And we've lost control of both x and y if it contains both $\forall x.\forall y.P$ and $\forall y.\forall x.P$ – or more or less any pair of formulæ in which the bindings of x and y occur in opposite orders.

It is certainly a pity that we can't automatically treat all variables that occur in binding positions as distinct. One possible remedy might be to make the argument work by demanding that variables can only occur in binding and bound positions. That could work in some logics – the $\forall$ introduction rule in a sequent presentation of natural deduction, for example, can be written as

$$\frac{\Gamma \vdash P[x\backslash c]}{\Gamma \vdash \forall x.P}\ (\text{FRESH } c)\ \forall \text{ intro}$$

But to force such a choice on every logic designer would be against our policy of non-commitment to any particular form of logic: it would make it impossible, for example, to write the rule

$$\frac{C \oplus x : T1 \vdash E : T2}{C \vdash \lambda x.E : T1 \rightarrow T2}\ \text{fun}$$

and impossible to use it if the problem is to type $\lambda x.x : \text{int} \rightarrow \text{int}$.

The only other remedy seems to be to embrace the problem. It may not occur very often in 'sensible' logics and proofs in any case. We can, and at present Jape does, analyse the structure of the base sequent of a proof to construct:

- groups of *interdependent* names which cannot be judged distinct unless a proviso makes them so;
- groups of names each *dependent* on a particular binder.

Then a variable name x is distinct from a name N if they don't appear in the same group of interdependent names and, if x is a variable name, N isn't one of the names which depend on x. At the same time Jape computes 'automatic' provisos which ensure that simultaneous bindings are correctly constructed: if a theorem or a rule contains, for example, $P[x,y \setminus m,n]$ then in any instance of that theorem or rule the arguments provided in place of x and y must be distinct.

There are still two complications. First, if the base sequent of the theorem contains an unknown we can't be sure of its structure, and we can't make any judgments of independence – all names have to be treated as interdependent until all unknowns have disappeared from the base sequent. That makes it relatively difficult to use Jape as a calculator, proving 'theorems' like $\vdash(\lambda x.\lambda y.y) : _T$, whereas in $\vdash(\lambda x.\lambda y.y) : T1 \to T2 \to T2$ it is easy to see that x and y are independent. In these particular examples, drawn from the polymorphically-typed lambda calculus, it is clear that program variable names can't appear in type formulae at all, so that a more refined syntactic hierarchy would solve the problem in this case. But in general the problem will remain.

6 A helpful proof, an obstructive proof and an offensive proof

It is necessary, at this point, to attempt to dispel a doubt that may be forming in the reader's mind. Jape's rule instantiation/unification mechanism takes almost no account of binding structures, but its substitution-simplification mechanism does, and so does unification when dealing with non-identical binding structures. Isn't it possible that such a collection of mechanisms can go wrong?

The answer we believe is No: Jape is sound, we are confident, in the sense that every theorem instance that can be constructed corresponds to a valid proof tree instance. The assurance that binding structures are treated properly during instantiation/unification comes from the NOTIN proviso, which has to be included as part of the statement of a theorem schema or may, in some circumstances, be inserted during the proof[1]. An example of those circumstances is shown in a proof of the theorem of the polymorphically-typed λ-calculus, that in any context $\lambda x.\lambda y.x$ has type $T1 \to T2 \to T1$. The first two steps of the proof use the rule of λ-abstraction to form a context containing two monotype type schemes:

$$\dfrac{\dfrac{C \oplus x \Rrightarrow \hat{}T \oplus y \Rrightarrow \hat{}T1 \vdash x:T}{(\lambda x.E) : T1 \to T2}}{C \oplus x \Rrightarrow \hat{}T \vdash (\lambda y.x):T1 \to T}$$
$$\dfrac{}{(\lambda x.E) : T1 \to T2}$$
$$C \vdash (\lambda x.\lambda y.x):T \to T1 \to T$$

[1] In most cases a theorem schema will be proved before it is ever used, but Jape does allow premature use. Since the proviso structure, and in some case the statement of the theorem, may be revised as a result of the proof it is obviously necessary to include a mechanism which reviews prior uses ofa theorem schema once it has been proved. Jape doesn't, at the time of writing, have such a mechanism, and in that sense it is unsound. We intend to rectify the matter in the very near future.

In the final step it is necessary to pick up the scheme binding $x \Rightarrow\hat{}\, T$ and to ignore $y \Rightarrow\hat{}\, T1$; that step automatically inserts a proviso to ensure that in any instance of the tree, the step can validly go through:

$$\dfrac{\dfrac{\dfrac{}{C(x) \Rightarrow S;\ S \gg T}}{C \oplus x \Rightarrow\hat{}\, T \oplus y \Rightarrow\hat{}\, T1 \vdash x : T}}{(\lambda x.E) : T1 \to T2}$$
$$\dfrac{C \oplus x \Rightarrow\hat{}\, T \vdash (\lambda y.x) : T1 \to T}{(\lambda x.E) : T1 \to T2}$$
$$C \vdash (\lambda x.\lambda y.x) : T \to T1 \to T$$

x NOTIN y

Such a proof is helpful to the user who doesn't understand or remember the necessity to include a proviso enforcing the constraints which would be a matter of course if theorem instances were matched using α-conversion. Clearly it would be more helpful still if we can find a way in which Jape can do without the proviso crutches!

Jape isn't so helpful in every circumstance. If we attempt to make a proof that $\exists x.\forall y.P(x,y) \vdash \forall y.\exists x.P(x,y)$ without stating any provisos and if we do so without providing any arguments to guide the instantiation/unification process, we get into a nice mess. The display after the first two steps of the proof shows the immediate difficulty:

$$\dfrac{\dfrac{\dfrac{(\forall y.P(x,y))[x \backslash m] \vdash (\exists x.P(x,y))[y \backslash m1]}{\vdash \forall}}{(\forall y.P(x,y))[x \backslash m] \vdash \forall y.\exists x.P(x,y)}}{\exists \vdash}$$
$$\exists x.\forall y.P(x,y) \vdash \forall y.\exists x.P(x,y)$$

Neither substitution formula can be instantly reduced

Sometimes it is possible to make proofs without requiring the NOTIN proviso to enforce a particular binding structure. Consider, for example, the problem of proving $\exists x.\forall y.P \vdash \forall y.\exists x.P$ – note that the rule is not written in predicate notation – in a version of the single-conclusion sequent calculus where the generalisation rules are

$$\dfrac{\Gamma \vdash A[x \backslash y]}{\Gamma \vdash \forall x.A}\ (\text{FRESH } y)\ \vdash \forall \qquad \dfrac{\Gamma, A[x \backslash y] \vdash C}{\Gamma, \exists x.A \vdash C}\ (\text{FRESH } y)\ \exists \vdash$$

– that is, where those rules use fresh *variables* rather than fresh *constants*. The proof is straightforward, if the right choice is made of variables at each step (x, y, y, x in the version displayed here):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash P}\ hyp}{P \vdash \exists x.P}\ \vdash \exists}{\forall y.P \vdash \exists x.P}\ \forall \vdash}{\forall y.P \vdash \forall y.\exists x.P}\ \vdash \forall}{\exists x.\forall y.P \vdash \forall y.\exists x.P}\ \exists \vdash$$

Jape will make this proof, and it won't need any NOTIN provisos to do so. It follows that $\exists z.\forall z.Q \vdash \forall z.\exists z.Q$ is an instance of this theorem schema. That, at first, is shocking: the instance doesn't seem to have the same binding structure as the theorem. It's a bit less worrying when you realise that the instance is valid in any case just because it reduces to $\forall z.Q \vdash \exists z.Q$: we haven't revealed an unsoundness in our mechanism. But it is worrying still, because you shouldn't be able to be so cavalier in treating binding structure.

You might suppose that the problem with this theorem is that it is stated in such a way that the binding structure is irrelevant: there are no explicit bound occurrences of x or y to fix it. But not so: in Jape the theorem $\exists z.\forall z.P(z,z) \vdash \forall z.\exists z.P(z,z)$ *can* be an instance of the schema $\exists x.\forall y.P(x,y) \vdash \forall y.\exists x.P(x,y)$ – the proof has just the same shape as the one above, and the particular theorem instance fits it. It's a valid theorem just because, in this logic, $\forall z.P(z,z) \vdash \exists z.P(z,z)$.

Metalogically the problem seems to us to lie in the particular rules that are being used: they allow these peculiar results. Nevertheless the result, especially in the case when the theorem schema is stated in predicate notation, is so shocking as to be offensive. We intend to eliminate the possibility of that proof.

7 Future improvements

Jape's treatment of schematic variables in theorem schemas has been described as a 'carbuncle' by one academic user. We rather agree, and it is possible to state what we ought to do about the problem.

1. We should make it possible to state generalisation rules which employ constants rather than variables. This requires an INTERNAL proviso, to prevent an unreasonable deduction of generality from a particular constant instance. Then the rules above would become, perhaps,

$$\frac{\Gamma \vdash A[x\backslash m]}{\Gamma \vdash \forall x.A}\ (\text{FRESH } m,\ \text{INTERNAL } m)\ \vdash\forall \qquad \frac{\Gamma, A[x\backslash m] \vdash C}{\Gamma, \exists x.A \vdash C}\ (\text{FRESH } m,\ \text{INTERNAL } m)\ \exists\vdash$$

2 We should deduce information about binding structure from bound occurrences of variables. For example, $\lambda z.\lambda z.z : num \to string \to num$ *can't* be an instance of $\lambda x.\lambda y.x : T1 \to T2 \to T1$ because in the first the inner binding captures the body of the function, and in the second the outer binding. Jape can use that information in two ways: first, it can insert additional 'automatic' provisos when the theorem schema $\lambda x.\lambda y.x : T1 \to T2 \to T1$ is applied, thus preventing $\lambda z.\lambda z.z : num \to string \to num$ from being recognised as an instance; second, because those provisos will always be included when the theorem is used, x and y can be assumed independent during a proof of the theorem schema $\lambda x.\lambda y.x : T1 \to T2 \to T1$. As a consequence Jape will be much 'quieter' in operation and the carbuncle largely removed.

3 Provisos such as x NOTIN P in a proof of a theorem schema which includes $P(x)$ in the scope of a binding occurrence of x are necessary only to make Jape treat P in a particular way: in effect, as a kind of abstraction formula. It should be possible, given a declaration that P is such a name, to 'compile' rule and theorem schemas which include it into forms which use explicit substitution formulæ such as $P[v\backslash x]$ and to include an automatic proviso x NOTIN P. Those compiled forms would be used in place of the original whenever the rule or theorem schema was applied. When a proof is attempted, as in the previous paragraph, that proviso information could be implicitly included into a proof by assuming that x and P were independent.

8 Summary

Jape's mechanism of rule and theorem instantiation is based on unification, tempered a little by α-conversion. As a result it is necessary to rely on explicit provisos to ensure that particular substitution formulæ can be reliably reduced during a proof. Jape's treatment of schematic and internal names is intricate, in order to reduce the necessity to state such provisos, but not yet intricate enough. It is necessary to make finer judgements of independence of schematic names, and to include a treatment of abstractions – of which predicates are an example – before it can be considered adequate.