SEARCHING AN APPROPRIATE LANGUAGE ABOUT PROOFS AND CONSTRUCTIONS: what a small semantical change may accomplish

Jean Paul VAN BENDEGEM

I

Everybody is familiar with the famous proof of the irrationality of the $\sqrt{2}$. It is easy to see – at least, for us, humans (and maybe mathematicians-logicians in particular) – that only a minor modification is required to obtain a proof of the irrationality of $\sqrt{p}$, where p is a prime number. Let us suppose that computer scientists succeed in writing a program capable of mathematical reasoning, and suppose further that the program produces a proof of the original theorem. It would save us a lot of time and energy if the program could somehow reflect on the proof itself. For, only if the program has this capability, is there any chance of it "discovering" the generalization of the theorem to arbitrary prime numbers without actually constructing a new proof altogether. (Note that I am not claiming that this would be sufficient). This requirement is obviously equivalent to the inclusion of a meta-level in the program.

This observation is hardly new: Bundy (1983) has a separate chapter on meta-inference and in Wos et al. (1984) meta-level reasoning has become an integrated part of the automated reasoner's tool kit. In order to develop such a higher level performance, a language is needed that must necessarily involve expressions about programs. This is exactly what the title of this paper refers to. I am convinced that most computer scientists and logicians involved in computer logic will have their answer ready: *dynamic logic* is the very language for that purpose. It has been and still is an object of intensive study, it is known to be of invaluable use and it is generally recognized within the field as the most plausible candidate. Nevertheless I will try to show in this paper, that there are some positively odd features about dynamic logic. Although this does not discredit dynamic logic itself, it certainly is the case that there are alternatives lacking all or some of these odd features.

In II, I present the propositional part of standard dynamical logic

(PDL). In III, the odd features are discussed and a network of alternatives is explored, but as it turns out, none of these is really satisfactory. An analysis of this failure leads to a change in the semantics of PDL. This change is such that whereas all cases considered in III are conservative extensions([1]) of PDL, the system presented in IV, is not. The paper ends by a brief comment on the use and value of this system for automated reasoning.

II

The essentials of PDL are the following:

syntax: – a list of propositional variables: p, q, r, ...
– a list of atomic programs: α, β, γ,...

- formation rules for programs:
  – an atomic program is a program
  – if α and β are programs then α; β (first α, then β) $\alpha \cup \beta$ (α or β) and α^* (α repeated an arbitrary number of times) are programs
  – if p is a program-free formula, then p? (is p the case?) is a program
- formation rules for formulas:
  – a propositional variable is a formula
  – if p and q are formulas and α is a program, then $p \vee q$, $\sim p$ and $[\alpha]p$ are formulas.

axiomatics:

- axioms: – PC-tautologies
  – $[\alpha](p \supset q) \supset ([\alpha]p \supset [\alpha]q)$
  – $[\alpha;\beta]p \equiv [\alpha][\beta]p$
  – $[\alpha \cup \beta]p \equiv [\alpha]p \& [\beta]p$
  – $[\alpha^*]p \equiv (p \& [\alpha][\alpha^*]p)$
  – $[p?]q \equiv (p \supset q)$
- rules: – if A and $A \supset B$, then B
  – if $\vdash A$, then $\vdash [\alpha]A$

([1]) I use "conservative extension" in a slightly different meaning. A theory T′ is a conservative extension of T if anything provable in T is also provable in T′ (plus some additional theorems).

semantics: a model is a triple $<W,R,v>$ such that:

- v: programs $\rightarrow$ $2^{W\times W}$
  α $\rightarrow$ $v(\alpha)$
  where $v(\alpha) = \{(s,t) \mid s,t \in W\}$

for composite programs the following definitions hold:

- $v(\alpha;\beta) = v(\alpha)^{\circ}v(\beta)$
  i.e. $= \{(s,t) \mid \exists u((s,u) \in v(\alpha) \text{ and } (u,t) \in v(\beta))\}$
- $v(\alpha \cup \beta) = v(\alpha) \cup v(\beta)$
- $v(\alpha^*) = \cup v(\alpha^n)$
  where $v(\alpha^0) = v((p \vee \sim p)?)$
  $v(\alpha^n) = v(\alpha)^{\circ}v(\alpha^{n-1})$
- $v(p?) = \{(s,s) \mid v(p,s) = 1\}$

- v: $F \times W \rightarrow \{0,1\}$

for program-free formulas $v(p,s)$ is treated according to classical propositional logic (relative to s). Of special interest is the semantical interpretation of $[\alpha]p$: $v([\alpha]p,s) = 1$ iff for all t, if $(s,t) \in v(\alpha)$ then $v(p,t) = 1$ The notion of validity and valid consequence is classically defined. In the sequel, I will write $s \vDash p$ instead of $v(p,s) = 1$ and $s\alpha t$ instead of $(s,t) \in v(\alpha)$.

As is well-known, PDL is complete and decidable, since it has the finite model property.

III

What is a proof or construction expressed in PDL? Consider any mathematical proof: you start out with some assumptions, p, *the initial data*, you perform a number of *operations* (basically applying the rules of the theory under consideration) and you end with a conclusion, q, *the final output*. In terms of PDL, this is neatly captured in the formula:

$$[p?;\alpha]q$$

i.e. if p is the case and α is executed, then q will result. (I will assume throughout that α itself does not contain tests, unless otherwise indicated). But, strangely enough, within PDL itself, the above formula behaves in a peculiar fashion. Direct semantical control shows that the following are invalid, and hence not provable:

(1) $p, [p?;\alpha]q \nvdash q$
(2) $\nvdash [p?;\alpha]p$
(3) $[p?;\alpha]q \nvdash [p?;\alpha](p \& q)$
(4) $\nvdash [p?;\alpha]q \vee [q?;\alpha]p$
(5) $[p?;\alpha]q, [q?;\beta]r \nvdash [p?;\alpha;\beta]r$
(6) $p \nvdash [q?;\alpha]p$ (provided q is not a falsehood)
(7) $\nvdash [p?;\alpha]q$ (provided q is not a theorem and p not a falsehood)

Some of these results are quite distressing, and some are not. (1) and (5) are troublesome: the former says that modus ponens for proofs or constructions does not hold, while the latter says that transitivity does not hold. (2) and (3) make clear that initial data cannot be preserved, they are confined to the input-world. On the other hand, the invalidity of statement (4) – the counterpart of the classical PC-theorem, $(p \supset q) \vee (q \supset p)$ – is a pleasing result. It would be very odd indeed if, given two statements p and q, we can either construct p out of q or vice versa. (6) is the classical counterpart of $p \vdash q \supset p$ and is generally recognized as one of the paradoxes of material implication.(2) So, its being invalid is a positive sign. (7), although unproblematic, informs us that there are no generally valid constructions.

One might wonder, if so little turns out to be valid or provable, if anything is provable at all. The following statements, though undesirable, do hold:

(8) $\vdash [p?;\alpha](q \vee \sim q)$
(9) $\vdash [(p \& \sim p)?;\alpha]q$

These two formulas are valid, because semantically speaking the worlds, i.e. the elements of W, are considered to be complete and consistent. As such they have nothing to do with α itself; in fact, the formulas make clear that α is arbitrary.
Even the weaker form of (9), viz.:

(10) $\sim p \vdash [p?;\alpha]q$

also holds.

A more general result, as to what is provable, can be stated:

(2) See Anderson and Belnap (1975) and Routley, Meyer, Plumwood and Brady (1982) for a detailed discussion on the problem of the paradoxes of material implication.

Theorem 1. $A \vdash B$ iff $[C?;\alpha]A \vdash [C?;\alpha]B$ iff $[B?;\alpha]C \vdash [A?;\alpha]C$

Proof: all proofs are semantically straightforward.

Although many formulas turn out to be provable, the result is still rather weak, because the correct inferences express no connection between input and output world. The conclusion is quite inevitable: the language should be made stronger in one way or another. But, rather surprisingly, it does not take much to trivialize $[p?;\alpha]q$. It is sufficient to add to the semantics the following two clauses:

(TF): if $s \vDash p$ and $s\alpha t$ then $t \vDash p$

This is the transfer rule, since it creates the possibility of transferring anything from the input world to the output world.

(RP): for all α, $s\alpha s$

The reflection principle states that the relation R is reflexive. Let PDL + (TF) + (RP) stand for PDL with the clauses added.
Then we have:

Theorem 2. in PDL + (TF) + (RP), $[p?;\alpha]q \equiv (p \supset q)$ is valid.

Proof: although the proof is rather trivial, I do present it, in order to show where (TF) and (RP) enter into the picture.

(i) $[p?;\alpha]q \supset (p \supset q)$

assume $s \vDash [p?;\alpha]q$ and $s \nvDash p \supset q$, i.e. $s \vDash p \& \sim q$. Since (RP) holds, we have $s\alpha s$. Together with $s \vDash p$, we have $s \vDash q$. Contradiction.

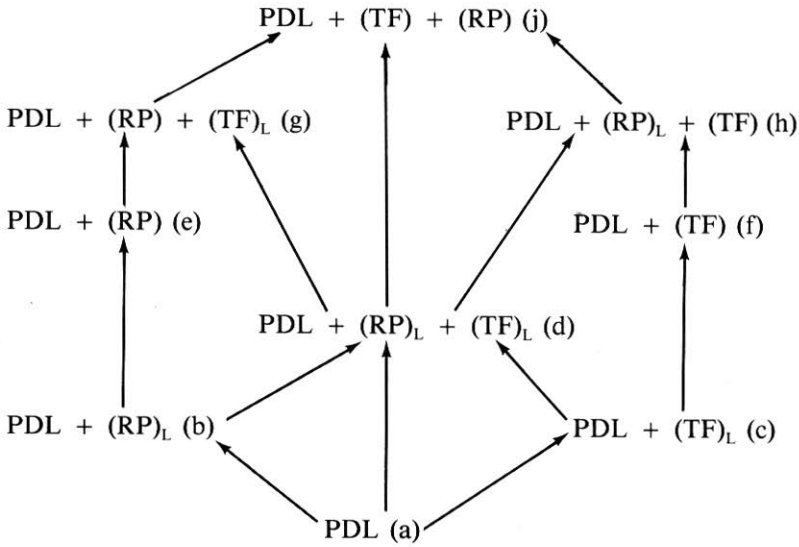
(ii) $(p \supset q) \supset [p?;\alpha]q$

assume $p \supset q$ and $<p?;\alpha> \sim q$. Hence there is a (s,t) such that $s\alpha t$ and $t \vDash \sim q$, and $s \vDash p$. But $s \vDash p \supset q$, so $s \vDash q$ and by (TF) $t \vDash q$. Contradiction. □

This result implies that all unhappy consequences of material implication([3]) are valid for $[p?;\alpha]q$, especially (4) which, as argued, is highly undesirable.

([3]) See 2.

The strategy that will be followed in the remainder of this chapter is this: cases in between PDL and PDL + (TF) + (RP) can be distinguished by restricting (TF) and (RP) (either drop (TF) or (RP) or restrict one or both of them to a particular set of programs L). This generates at least the following possibilities:



The cases involving (TF) are particularly uninteresting, viz. (f), (h) and (j). Theorem 2 showed that if (TF) holds, then $(p \supset q) \supset [p?;\alpha]q$ holds. But this implies that all irrelevant properties of $\supset$ are transferred to $[p?;\alpha]q$.

I first consider PDL + (RP). Axiomatically, the semantical addition corresponds to adding the following axiom:

(11) $[p?;\alpha]q \supset (p \supset q)$

(11) is equivalent to $[\alpha]q \supset q$, as the following theorem shows.

Theorem 3. $[p?;\alpha]q \supset (p \supset q)$ is equivalent with $[\alpha]q \supset q$.

Proof: (i) assume $[p?;\alpha]q$ and $[\alpha]q \supset q$. $[p?;\alpha]q$ is equivalent to $p \supset [\alpha]q$. Together with $[\alpha]q \supset q$, we find $p \supset q$.
(ii) assume $[p?;\alpha]q \supset (p \supset q)$ and $[\alpha]q$. If $[\alpha]q$ then $(p \vee \sim p) \supset [\alpha]q$ or $[(p \vee \sim p)?;\alpha]q$, hence $(p \vee \sim p) \supset q$. But then q. □

By adapting one of the well-known proofs of the completeness theorem of PDL,([4]) it is easy to show that:

Theorem 4. PDL + (11) is complete with respect to the PDL-semantics with (RP).

In this logic, modus ponens for programs does hold:

(12) $p, [p?;\alpha]q \vdash q$

and transitivity holds for identical programs:

(13) $[p?;\alpha]q, [q?;\alpha]r \vdash [p?;\alpha]r$

But transitivity for different programs still fails as might be expected. For anyone familiar with modal logic, it is obvious that PDL + (RP) is based on the modal logic T, whereas PDL itself is based on K (corresponding to the first two axioms and the necessitation rule). A rather particular feature of this logic is that no formula of the form $[p?;\alpha]q$ can be a theorem, if p is not a theorem. The argument is quite simple: it is easy to construct a countermodel, viz. a model in which $<p?;\alpha> \sim q$ is true. Take $W = \{s,t\}$, $v(\alpha) = \{(s,t), (s,s), (t,t)\}$ and $v(p,s) = 1$ and $v(q,t) = 0$. Therefore if we restrict ourselves to formulas that are program-free or of the form $[p?;\alpha]q$, then the language must be purely inferential as far as program statements is concerned. In other words, there is no program of which we can claim that it is always executable.

The variant PDL + $(RP)_L$ consists in restricting the universal quantifier in (RP) to a particular set L, thus

$(RP)_L$ for all $\alpha \in L$, $s\alpha s$.

Axiomatically, this corresponds to the addition of a list of axioms, one for each member of L stating that for that particular program $[\alpha]p \supset p$ holds. As hardly anything new can be expected from this variant that has not been said concerning PDL + (RP) I now turn to the remaining

([4]) See Kozen and Parikh (1981) for a completeness proof for PDL. Their definition of $A \rightarrow^a B$ should be extended to include the case where A&B is consistent, viz. if A&B then $A \rightarrow^a B$, for all a occurring in the formulas of FL(W).
It is also possible to transcribe the Kozen-Parikh proof into a typical modal logic completeness proof. In that case, the proof proceeds along the lines of the proof presented in Hughes and Cresswell (1968) for the system T.

cases all of which involve $(TF)_L$. Assume once again that a particular list of programs, L is given. Members of L will be introduced in the language by certain constants, say ℓ_1, ℓ_2, ..., ℓ_n (n finite). Semantically, a particular set will correspond to each of these constants and for these sets we stipulate that (TF) holds.
Axiomatically this corresponds to introducing a particular axiom of the form $[p?; \ell_i]q$, for each ℓ_i. A typical example would be:

(14) $[p?; \mathrm{LET}]p$

This program leaves p unchanged. It transfers p from the input to the output world. Obviously we now do have theorems expressing basic capabilities. But, modus ponens for programs remains invalid, unless (RP) is added. The systems (c), (d) and (g) do have one very nice property, that deserves special attention:

(15) $[p?;\alpha]q \nvdash [p?;\alpha](p\&q)$ if α is not a constant
(16) $[p?;\ell_i]q \vdash [p?;\ell_i](p\&q)$
(17) $[p?;\alpha]q, [p?;\alpha]p \vdash [p?;\alpha](p\&q)$ for all α

(15) and (16) express the difference between a constant and an arbitrary program. (17) shows what has to be added to an arbitrary program so that it behaves like a constant program. Why is this a nice property? Take the following quite simple program:

$$[(x = 0)?; x \to x+1](x = 1)$$

This program changes the value of x, if it is 0, into 1. For this particular program, if (TF) were to apply, we would obtain an inconsistency, viz. $(x = 0) \,\&\, (x = 1)$. In short, it is clearly not a necessary property of *every* program that it should satisfy (TF). Because of theorem 2, it is straightforward to see what happens: suppose you have $[p?;\alpha] \sim p$ of which the program above is a special instance (q is such that it implies $\sim p$). Since $[p?;\alpha] \sim p \equiv (p \supset \sim p)$ and $p \supset \sim p \equiv \sim p$, $[p?;\alpha] \sim p \equiv \sim p$, i.e. the program itself is annihilated. What we try to capture, viz. that x has changed its value, is precisely wiped out. Hence it is reasonable to restrict (TF) to a particular set so as to safeguard its expressive power. An important remark is that, given an arbitrary list of programs L, the members of L are not necessarily independent. Suppose that we add, besides LET, the special program SEL_L which does the following:

(18) $[(p\&q)?\ ;\ SEL_L]p$

However within PDL + $(TF)_{\{LET\}}$, SEL_L can be derived, i.e. there is a program that produces the very same behaviour:

LET assures us that $[(p\&q)?;\ LET](p\&q)$ holds

But $(p\&q) \supset p$. Because of theorem 1, if a program produces A, then it produces all consequences of it, thus:

$[(p\&q)?;\ LET]p$.

Thus LET does exactly the job of SEL_L. Actually, LET is a particularly strong program for it is easy to see that, if A and B are program-free formulas and $\vdash_{PC} A \supset B$, then $[A?;\ LET]B$. Thus the following is provable in the system under discussion:

(19) $\vdash (p \supset q) \supset [p?;\ LET]q$

Using a technique similar to that of theorem 4, we can prove that

Theorem 5. PDL + (19) is complete with respect to the PDL-semantics with $(TF)_{\{LET\}}$.

Since it is obvious that we want modus ponens for programs to hold for all programs, the system (g), i.e. PDL + (RP) + $(TF)_{\{LET\}}$ is the most interesting case. For this system, the following theorem holds (combining theorem 4 and 5):

Theorem 6. PDL + (11) + (19) is complete with respect to the PDL-semantics with (RP) and $(TF)_{\{LET\}}$.

This logic is particularly interesting because reasonings about programs are such that:
(i) nearly all classical logical rules are valid for formulas involving programs:

$[p?;\alpha]q,\ p \vdash q$
$[p?;\alpha]q,\ [q?;\alpha]r \vdash [p?;\alpha]r$
$[p?;\alpha]q,\ [p?;\alpha]r \vdash [p?;\alpha](q\&r)$
$[p?;\alpha]r,\ [q?;\alpha]r \vdash [(p\vee q)?;\alpha]r$

(ii) if $p \supset q$ is a logical truth, then the program LET translates this truth into the capability for transforming p into q, or

$$\text{if } \vdash_{PC} p \supset q, \text{ then } [p?;\text{LET}]q$$

However, although this language is definitely richer than PDL, and has the nice features listed above, it still lacks one property that is essential: transitivity for different programs, i.e. (5).
All systems discussed so far suffer from this shortcoming. Is there an extension of PDL that does make it valid? The answer is no, because of the specific semantics of PDL and its extensions. When we evaluate a PDL-formula, we choose a specific world s as a base world. All programs have their starting point in this world. Given a composite program $\alpha;\beta$, semantically its translation will be $s\alpha t$ and $t\beta u$. Thus β never starts in s, unless α is a test-program, but then (5) does hold.
Hence the presence of the formula $[q?;\beta]r$ does not help us much. (Actually what is required is $[p?;\alpha;q?;\beta]r$). But that implies that we have no other choice than to change the PDL-semantics itself. The system presented in the next chapter turns out to be a non-conservative extension of PDL. This is however not a necessary feature and it is an open problem whether a non-trivial conservative extension can be found. But, fortunately, the valid inferences that become invalid due to the change, were troublesome anyway (as far as irrelevance is concerned).

IV

At first sight, the change in the semantics is quite simple. It is sufficient to strenghten the clause for $[\alpha]p$. Instead of selecting a particular world as a starting point, we propose a semantical interpretation of $[\alpha]p$ that is independent of the pair of worlds chosen. Thus:

$$v([\alpha]p,s)=1 \text{ iff for all t, t' if } t\alpha t' \text{ then } t' \vDash p$$

Obviously, in this system – indicated by PDL + (∀) – transitivity for different programs does hold.
Although one might expect that anything valid in PDL should be valid in PDL + (∀) (because the new semantical clause is definitely stronger), this is not the case, as we already mentioned.
Examples of (PDL + (∀))-invalid, but PDL-valid formulas are:

(20) $\not\vdash (p \supset q) \supset [p?]q$

It is sufficient to evaluate $<p?> \sim q$ in a world different from the one in which $p \supset q$ is evaluated.

(21) $\sim p \not\vdash [p?;\alpha]q$

This is a "hopeful" statement, as the invalidity of (21) eliminates a typical, unwanted, irrelevant property of implication. However what does remain valid, is:

if $\vdash A$ then $\vdash [\sim A?;\alpha]B$

Thus PDL + (∀) will in the best of cases, prove to be a weakly relevant system.

(22) $\not\vdash [\alpha;\beta]p \supset [\alpha][\beta]p$

This is a rather important result, as it shows that in PDL + (∀) the elimination of ";" is no longer possible.
Although some formulas cease to be valid, there is an explosion of valid formulas. The following theorem provides a measure of this growth.

Theorem 7. Consider the K-fragment of PDL + (∀), i.e. PDL restricted to (i) PC tautologies
(ii) $[\alpha]\,(p \supset q) \supset ([\alpha]p \supset [\alpha]q)$
and (iii) the new semantical clause for $[\alpha]p$
This fragment is equivalent to the OS5-fragment of PDL (where OS5 is the deontic "partner" os S5).([5])

Proof: It is easy to show that the semantical properties of an OS5-model are equivalent to the new semantical clause.

However, it is still the case that we need (RP) to ensure modus ponens and $(TF)_{\{LET\}}$ to ensure the existence of basic capabilities. Hence the system that seems to be the most promising candidate is:

PDL + (∀) + (RP) + $(TF)_{\{LET\}}$

([5]) See Åqvist (1984) for an excellent overview of deontic logic.

Note that in this logic, (20) and (21) remain invalid, whereas (22) is now valid.
The following theorem presents a first result:

Theorem 8. PDL + (∀) + (RP) restricted to a single program α and to first-order statements involving programs (i.e. only expressions of the form $[p?;\alpha]q$ where p and q do not contain α are admitted), is *completely* characterized by the following axioms and rules:

(i) PC tautologies
(ii) $([p?;\alpha]r \,\&\, [q?;\alpha]r) \equiv [(p \vee q)?;\alpha]r$
(iii) $([p?;\alpha]q \,\&\, [q?;\alpha]r) \supset [p?;\alpha]r$
(iv) $([p?;\alpha]q \,\&\, [p?;\alpha]r) \equiv [p?;\alpha](q \& r)$
(v) $p,\ [p?;\alpha]q \ /\ q$
(vi) if $\vdash A$ then $\vdash [B?;\alpha]A$

Instead of presenting the proof that is quite lengthy, I will briefly discuss an example:

the formula $[(p \vee q)?;\alpha]r \supset [p?;\alpha]r$ is valid and equivalent to $<(p \vee q)?;\alpha> \sim r \vee [p?;\alpha]r$

As one can see the following holds:

$\vdash \sim p \vee (p \vee q)$ ("collecting" the inputs)
$\vdash r \vee \sim r$ ("collecting" the outputs)

By (vi) we have $\vdash [p?;\alpha](r \vee \sim r)$
or $\vdash [p?;\alpha]r \vee <p?;\alpha> \sim r$
We also have $\vdash p \supset (p \vee q)$ hence
$<p?;\alpha> \sim r \supset <(p \vee q)?;\alpha> \sim r$
thus obtaining
$\vdash <(p \vee q)?;\alpha> \sim r \vee [p?;\alpha]r$

It must be noted that the full system does have some strange properties. Among other things, the role of the programs is severely reduced, as the following is valid:

(23) $[\alpha;\beta]p \supset [\beta]p$

and even more distressing, that

(24) $[p?;\alpha\,;q?;\beta]r \supset [(p \& q)?;\beta]r$

is valid.

Apparently, it is possible to "erase" programs from the formulas. This raises the question what the "appropriateness" of the title of this paper amounts to. As it is, in the field of automated reasoning, this logic may prove to be quite interesting.

Consider this very simple argument as an example:

$$\begin{array}{l} p \\ p \supset (q \vee r) \\ q \supset s \\ r \supset s \\ \hline s \end{array}$$

An automated reasoning program transforms the premisses into (where the vertical bar indicates disjunction):

(a) p
(b) $\sim p \mid q \mid r$
(c) $\sim q \mid s$
(d) $\sim r \mid s$

and proceeds thus:

(e) $q \mid r$ (by resolution on (a) and (b))
(f) $r \mid s$ (by resolution on (c) and (e))
(g) s (by resolution on (d) and (f))

In terms of the language presented here, the transcription is somewhat different:

(a) p
(b) $[p?;\alpha](q \vee r)$
(c) $[q?;\alpha]s$
(d) $[r?;\alpha]s$

and we proceed thus:

(e) $q \vee r$ (MP for programs on (a) and (b))
(f) $[(q \vee r)?;\alpha]s$ (composition for identical outputs on (c) and (d))
(g) s (MP for programs on (e) and (f))

Rewriting the argument this way *is* different from the classical resolu-

tion method but one must agree that it is closer to the "natural" way of reasoning. Furthermore, it opens up the possibility of an *economical* automated reasoner. The following argument, classically valid, is invalid in the system presented here:

$$\frac{p}{q \supset p}$$

as its transcription

$$\frac{p}{[q?;\alpha]p}$$

is invalid

Of course, the central question remains unanswered: is it possible to find a resolution-type formulation for this logic.
If so, then we would have indeed an economical *effective* automated reasoner. ([6])

Research Associate NFWO Jean Paul VAN BENDEGEM
Rijksuniversiteit Gent
Vrije Universiteit Brussel

([6]) The very first issue of the Journal of Automated Reasoning contains an interesting short paper by Graham Wrightson stressing the importance of non-classical logical systems. It is equally interesting to note that one of the disadvantages mentioned by him concerning the resolution method is the fact that it introduces redundancy.

REFERENCES

Anderson, Alan Ross & Benap, Nuel D., Jr.: *Entailment, the logic of relevance and necessity*, volume 1, Princeton UP, Princeton, 1975.

Åqvist, Lennart: "Deontic Logic", in: Gabbay & Guenthner, pp. 605-714.

Bundy, Alan: *The computer modelling of mathematical reasoning*, Academic Press, New York, 1983.

Gabbay, D. & Guenthner, F. (eds): *Handbook of Philosophical Logic, volume II: extensions of classical logic*, Reidel, Dordrecht, 1984.

Harel, David: *First-order Dynamical Logic*, Springer, Heidelberg, 1979.

Harel, David: "Dynamic Logic", in: Gabbay & Guenthner, pp. 497-604.

Hughes, G.E. & Cresswell, M.J.: *An introduction to modal logic*, Methuen, London, 1968.

Kozen, Dexter & Parikh, Rohit: "An elementary proof of the completeness of PDL", *Theoretical Computer Science*, 14, 1981, pp. 113-118.

Routley, R.; Meyer, R.K.; Plumwood, V. & Brady, R.: *Relevant Logics and Their Rivals. Part I: The Basic Philosophical and Semantic Theory*, Ridgeview, Atascadero, California, 1982.

Wos, Larry; Overbeek, Ross; Lusk, Ewing & Boyle, Jim: *Automated Reasoning: introduction and applications*, Prentice-Hall, Englewood Cliffs, 1985.

Wrightson, Graham: "Nonclassical Logic Theorem Proving", *Journal of Automated Reasoning*, 1,1, 1985, pp. 35-37.