On the design of calculational proofs

§ 0. By their brevity and unexpected turns, calculational proofs often strike the reader as being very ingenious constructions. Far be it from us to belittle the ingenuity of their inventors, but that should not prevent us from studying a number of representative proofs in the hope of learning why those proofs have the structure they have. This paper reports on such a study.

Among the things that will emerge, I already mention a few.

Firstly, many steps that may seem surprising at first sight, are, in fact, (almost) dictated, as they are the only (or by far the simplest) transformation that will enable us to exploit one of the givens that has to be taken into account. In this connection I recommend that we maintain as fine-grained a bookkeeping as possible of what of the givens we have used: what has not been used yet often indicates the direction in which the proof should be completed

Secondly, when faced with an antisymmetric and reflexive relation, i.e. a relation $\leq$ (with transpose $\geq$) such that

$$x = y \;\equiv\; x \leq y \wedge x \geq y \quad \text{for all } x, y \;,$$

we must be prepared for a proof with a possibly subtle interplay between calculating with equalities and with inequalities: any hints we can extract as to when to stress which of the two approaches will obviously be most welcome.

§ 1. Let me show you, as a first indication of the kind of considerations I would like to highlight, a calculational proof of which each step is forced (as a result of which the proof is very well known, not to say canonical).

About predicate transformers f and g is given

(0) $\quad [f.x \Rightarrow y] \equiv [x \Rightarrow g.y] \quad$ for all x, y ;

we are asked to show that g is universally conjunctive, i.e.

(1) $\quad [g.\langle \forall y : y \in W : y \rangle \equiv \langle \forall y : y \in W : g.y \rangle]$
for any bag W of predicates.

Remark We indicate function application not implicitly by juxtaposition but explicitly by an infix period of high syntactic binding power. With the square brackets we denote the "everywhere operator", a function from "predicates" to the "boolean scalars" true and false . For further elaboration

we refer the reader for instance to [0]. (End of Remark.)

For brevity's sake, the range "$y \in W$" is omitted in the following calculation; we should remember that it is a scalar range. We shall first give the proof and then discuss it.

Proof We observe for any x

$$
\begin{array}{ll}
 & [\, x \Rightarrow g.\langle \forall y :: y \rangle \,] \\
= & \quad \{(0) \text{ with } y := \langle \forall y :: y \rangle\} \\
 & [\, f.x \Rightarrow \langle \forall y :: y \rangle \,] \\
= & \quad \{\text{pred. calc.: } Q\Rightarrow \text{ over } \forall\} \\
 & [\, \langle \forall y :: f.x \Rightarrow y \rangle \,] \\
= & \quad \{\text{pred. calc.: interchange (NB. range is scalar)}\} \\
 & \langle \forall y :: [\, f.x \Rightarrow y \,] \rangle \\
= & \quad \{(0)\} \\
 & \langle \forall y :: [\, x \Rightarrow g.y \,] \rangle \\
= & \quad \{\text{pred. calc.: interchange (NB. range is scalar)}\} \\
 & [\, \langle \forall y :: x \Rightarrow g.y \rangle \,] \\
= & \quad \{\text{pred. calc.: } Q\Rightarrow \text{ over } \forall\} \\
 & [\, x \Rightarrow \langle \forall y :: g.y \rangle \,] \qquad ;
\end{array}
$$

having thus established

$$[\, x \Rightarrow g.\langle \forall y :: y \rangle \,] \equiv [\, x \Rightarrow \langle \forall y :: g.y \rangle \,] \quad \text{for all } x,$$

we conclude (1) on account of

$$(2) \quad \langle \forall x : \text{true} : [\, x \Rightarrow P \,] \equiv [\, x \Rightarrow Q \,] \rangle \equiv [\, P \equiv Q \,] \quad ,$$

which is a well-known lemma that follows from implication's reflexivity and antisymmetry.

(End of Proof.)

Let us now analyse the extent to which the structure of the above proof has been forced upon us.

Firstly, do we need (0) in order to show (1)? Yes, we do, because (with $W := \emptyset$) a consequence of (1) is $[g.\text{true} \equiv \text{true}]$, and that definitely does not hold for any g, for instance not for the predicate transformer g defined by $[g.x \equiv \text{false}]$ for all x.

Next, being obliged to use (0), we observe that our <u>only</u> way of exploiting that the g in the left-hand side $g.\langle \forall y :: y \rangle$ of (1) satisfies (0), is to instantiate the latter by $y := \langle \forall y :: y \rangle$; this <u>forces</u> us to explore $[x \Rightarrow g.\langle \forall y :: y \rangle]$, this now being the smallest expression that contains the left-hand side of (1) and can be rewritten using (0). The first step then applies (0) with that instantiation.

In view of the right-hand side of (1), it is from here on our duty to eliminate f and to reintroduce g, which can <u>only</u>

be done by a second appeal to (0). Because there is no point in undoing the first step, we must apply (0) to a subexpression of the form $[f.x \Rightarrow ?]$ with a consequent different from $\langle \forall y :: y \rangle$. The next two steps of the predicate calculus form such a subexpression, and then our second appeal to (0) reintroduces g.

Our final task is to form $\langle \forall y :: g.y \rangle$, the right-hand side of (1), as subexpression; the next two steps of the predicate calculus form $\langle \forall y :: g.y \rangle$ as consequent, and, on account of (2), we are done.

The moral of the story is that, apart from the choice of syntax for proof presentation, we had no choice.

§ 2. The full-blown principle of Leibniz states (to the best of my knowledge) that for x and y of some type, and f ranging over the functions on that type

$$(3) \qquad x = y \equiv \langle \forall f :: f.x = f.y \rangle \quad .$$

Reading the above equivalence as a mutual implication, we observe that "$\Leftarrow$" follows by instantiating the right-hand side with for f the identity function. For this con-

clusion, even the existence of other functions is irrelevant. For the general function it is the implication in the other direction that is relevant:

$$(4) \qquad x = y \;\Rightarrow\; f.x = f.y \quad ,$$

and this is what is usually referred to as "Leibniz's Principle".

Remark Appealing to

$$q \Rightarrow \langle \forall x :: x \rangle \;\equiv\; \langle \forall x :: q \Rightarrow x \rangle$$

— itself an immediate consequence of "$\vee$ distributes over $\forall$" —, we have taken the universal quantification over f outside. The possibility of thus removing a universal quantification from a consequent often provides the incentive to split up an equivalence involving a universally quantified equivalent into a mutual implication. (End of Remark.)

The connection between function application and equality is very tight: function application preserves equality and, conversely, — besides the two constant relations *true* and *false* — equality is the *only* relation that is preserved by function application. In Section 0, we referred to the interplay between calculating with equalities and with

inequalities; the above observation tells us that when we have to capture the essence of function application, i.e. when we have to apply Leibniz's Principle, we may be forced to introduce an equality so as to be able to do so. The next section gives two examples of this phenomenon.

§ 3. In a wider context, Jamey Leifer had to deal with two functions f and g whose domains and ranges are all the same, and whose applications commute, i.e.

(5) $$f.(g.x) = g.(f.x) \quad \text{for "all" } x \ .$$

In order to appeal to (5), we have to apply -say- g to an f-value, say $f.x$. In order to appeal to Leibniz's Principle, that f-value has to equal something else, say y, for the time being. So we observe for any x, y

$$\begin{array}{ll} & f.x = y \\ \Rightarrow & \quad \{ \text{Leibniz's Principle} \} \\ & g.(f.x) = g.y \\ = & \quad \{(5)\} \\ & f.(g.x) = g.y \ . \end{array}$$

The last line tells us that a nice choice for y is x, and thus we have proved

$$f.x = x \;\Rightarrow\; f.(g.x) = g.x \quad ,$$

or, in words: if x is a fixed point of f, so is $g.x$. After the above observation, Jamey Leifer conducted all the rest of his (beautiful) argument in terms of fixed points, a concept that did not occur at all in the theorem he had set out to prove.

§ 4. The other example I would like to show is the proof of Perry Moerland's lemma. (Also this came up in a wider context.) About predicate transformer f we are given that it is monotonic and has a left-inverse g, i.e.

(6) $\quad [x \Rightarrow y] \Rightarrow [f.x \Rightarrow f.y]$ for all x,y

(7) $\quad [g.(f.x) \equiv x]$ for all x;

we have to show that as a consequence

(8) $\quad [x] \Leftarrow [f.x]$ for all x.

We are looking for a strengthening chain of booleans, starting at $[x]$ and ending with $[f.x]$. Along the way we have to introduce f; for doing so, (6) is not a good candidate, but (7) is. The advantage of introducing f with the aid of (7) is that then g is introduced as

well. (Note that (6) by itself does not suffice for the demonstration of (8), i.e. that (7) has to be exploited, but that exploitation obviously requires that g enters the picture.)

Having introduced g and still aiming for $[f.x]$, we have to get rid of g again. Since the only other thing known about g is that it is a function, the proper candidate for the removal of g is Leibniz's Principle. Since an appeal to the latter requires equalities, the introduction of an equality was Perry Moerland's first concern; one observes for any x

$$
\begin{array}{ll}
 & [\,x\,] \\
= & \quad \{\text{predicate calculus}\} \\
 & [\,x \equiv \text{true}\,] \\
= & \quad \{(7)\,;\,(7) \text{ with } x := \text{true}\} \\
 & [\,g.(f.x) \equiv g.(f.\text{true})\,] \\
\Leftarrow & \quad \{\text{Leibniz's Principle}\} \\
 & [\,f.x \equiv f.\text{true}\,] \\
= & \quad \{\text{predicate calculus}\} \\
 & [\,f.x \Leftarrow f.\text{true}\,] \wedge [\,f.x \Rightarrow f.\text{true}\,] \\
= & \quad \{[\,x \Rightarrow \text{true}\,] \text{ and } (6) \text{ with } y := \text{true}\} \\
 & [\,f.x \Leftarrow f.\text{true}\,] \\
\Leftarrow & \quad \{\text{predicate calculus}\} \\
 & [\,f.x\,]
\end{array}
$$

§5. With a relation like $\geq$ (and its transpose $\leq$) comes the notion of monotonicity with respect to it, i.e. "preserving it": "f is monotonic with respect to $\geq$" means

(9) $\quad x \geq y \;\Rightarrow\; f.x \geq f.y \quad$ for all x, y .

(Note that in this terminology, Leibniz's Principle states that each function is "monotonic with respect to $=$".)

We now restrict ourselves to such domains and relations $\geq$ such that for any set of values the lowest higher bound "$\uparrow$" —and similarly the highest lower bound "$\downarrow$"— exists, i.e. we assume that for any range of the dummy x and any function t of the appropriate type we can define $\langle \uparrow x :: t.x \rangle$ by

(10) $\quad w \geq \langle \uparrow x :: t.x \rangle \;\equiv\; \langle \forall x :: w \geq t.x \rangle$ for all w.

The important theorem is that, for any f that is monotonic with respect to $\geq$,

(11) $\quad f.\langle \uparrow x :: t.x \rangle \geq \langle \uparrow x :: f.(t.x) \rangle$;

its proof is another example of "there is only one thing you can do".

Proof We observe for any monotonic f, etc.

$$
\begin{array}{ll}
 & f.\langle \uparrow x :: t.x\rangle \geq \langle \uparrow x :: f.(t.x)\rangle \\
= & \quad \{ \text{(10) with } w,t := f.\langle \uparrow x :: t.x\rangle, f\circ t \} \\
 & \langle \forall x :: f.\langle \uparrow x :: t.x\rangle \geq f.(t.x)\rangle \\
\Leftarrow & \quad \{ f \text{ monotonic with respect to } \geq , \\
 & \qquad \forall \text{ monotonic with respect to } \Leftarrow \} \\
 & \langle \forall x :: \langle \uparrow x :: t.x\rangle \geq t.x\rangle \\
= & \quad \{ \text{(10) with } w := \langle \uparrow x :: t.x\rangle \} \\
 & \langle \uparrow x :: t.x\rangle \geq \langle \uparrow x :: t.x\rangle \\
= & \quad \{ \geq \text{ is reflexive} \} \\
 & \text{true}
\end{array}
$$

Note that (10), when viewed as mutual implication, is used in either direction.

(End of Proof.)

The above proof is so simple, and so completely forced by the definitions, that the usefulness of the theorem may come as a surprise. Predicate transformers that are monotonic with respect to implication are very common, and for such an f, which satisfies (6), we have

(12) $[f.\langle \forall x :: x\rangle \Rightarrow \langle \forall x :: f.x\rangle]$ for all ranges

(13) $[f.\langle \exists x :: x\rangle \Leftarrow \langle \exists x :: f.x\rangle]$ " ,

with the special cases

(14) $[f.(x \wedge y) \Rightarrow f.x \wedge f.y]$ for all x,y

(15) $[f.(x \vee y) \Leftarrow f.x \vee f.y]$ " .

In fact, (12) through (15) are more than useful consequences of f's monotonicity, they are restatements of it.

§6. In section 1 we proved - in our appeal to (2) - $[P \equiv Q]$ by showing for any x

$$[x \Rightarrow P] \equiv [x \Rightarrow Q] \quad .$$

We may well ask what has been gained by the introduction of the dummy x. A possible answer is given by the observation that, in order to derive $[P \equiv Q]$ from the above, the latter has to be instantiated twice, viz. with $x := P$ and with $x := Q$. This double instantiation strongly suggests that the the introduction of dummy x has enabled us to avoid a ping-pong argument (i.e. a twofold case analysis).

But this is not the whole story, for, analogously to (2), we have, for instance,

(16) $\langle \forall x :: [P \Rightarrow x] \Leftarrow [Q \Rightarrow x] \rangle \equiv [P \Rightarrow Q]$.

This authorizes us to conclude $[P \Rightarrow Q]$ by showing for any x

(17) $[P \Rightarrow x] \Leftarrow [Q \Rightarrow x]$;

the conclusion, however, requires a single instantiation only, viz. $x := Q$. Yet, the introduc-

tion of dummy x as in (17) is justifiable in more than one way. We can (i) prove (17) for arbitrary x, and then instantiate (17) with $x := Q$, or (ii) carry out the argument not for arbitrary x, but right from the start for Q, thus deriving, instead of (17),

(18) $$[P \Rightarrow Q] \Leftarrow [Q \Leftarrow Q] \quad .$$

Here (i) is to be preferred above (ii) for more than one reason:

- the dummy x is shorter than the expression Q, hence (i) saves ink and paper;
- argument (ii) obscures the existence of demonstrandum (17), obscures the fact that the internal structure of the first Q and of the last Q in (18) is totally irrelevant for the latter's demonstration; argument (i) is clearer than (ii) in that it heads for (17), which clearly states that the first and the last Q's in (18) could be "anything", provided they are the same. In the jargon: argument (i) is better disentangled than argument (ii).
- the full force of this disentanglement manifests itself when, besides the proof, we design the theorem as well and Q emerges as the result of our calculations.

We shall illustrate the findings of the last two sections (and a little bit more) in the next section.

§7. In this section, we are looking, by way of example, for a nontrivial theorem about relational composition. Composition will be denoted by an infix ";" with a syntactic binding power higher than that of the logical infix operators but lower than that of the unary operators "¬" and "~".

We shall use that the transposition "~" distributes over the logical operators, that composition is monotonic in both its arguments, and that composition and transposition are coupled by the "right-exchange", for which we choose here the formulation that for all x, y, z

(19) $[x;y \wedge z \Rightarrow \text{false}] \equiv [\sim x; z \wedge y \Rightarrow \text{false}]$.

The exchange rules enable us to manipulate a composition that occurs as antecedent, but there are <u>no</u> analogous manipulative opportunities for a composition that occurs as consequent, and we may therefore expect a problem when trying to prove a theorem of the form

(20) $[a \Rightarrow b;c]$.

This problem, however, can be overcome with the aid of (16), which states that we can demonstrate (20) by proving - in a variation

on (17) – for arbitrary z

(21) $[b;c \wedge z \Rightarrow \mathit{false}] \Rightarrow [a \wedge z \Rightarrow \mathit{false}]$,

and in this formulation of our proof obligation, the composition "b;c" occurs as (conjunct in an) antecedent, and is thus amenable to manipulation via an exchange rule!

We shall now first give our calculation that transforms $[b;c \wedge z \Rightarrow \mathit{false}]$ without strengthening into $[a \wedge z \Rightarrow \mathit{false}]$, and shall discuss the proof later. In the course of the calculation, suitable values for c and a will be chosen. We observe for any z

$$
\begin{array}{lll}
 & [b;c \wedge z \Rightarrow \mathit{false}] & \\
= & \quad \{\text{right-exchange}\} & (A) \\
 & [\sim b;z \wedge c \Rightarrow \mathit{false}] & \\
= & \quad \{\text{choose } c\text{: } [c \equiv \sim d;e \wedge f]\} & (B) \\
 & [\sim b;z \wedge \sim d;e \wedge f \Rightarrow \mathit{false}] & \\
\Rightarrow & \quad \{\text{monotonicities}\} & (C) \\
 & [(\sim b \wedge \sim d);(z \wedge e) \wedge f \Rightarrow \mathit{false}] & \\
= & \quad \{\sim \text{ distributes over } \wedge\} & (D) \\
 & [\sim(b \wedge d);(z \wedge e) \wedge f \Rightarrow \mathit{false}] & \\
= & \quad \{\text{right-exchange}\} & (E) \\
 & [(b \wedge d);f \wedge e \wedge z \Rightarrow \mathit{false}] & \\
= & \quad \{\text{choose } a\text{: } [a \equiv (b \wedge d);f \wedge e]\} & (F) \\
 & [a \wedge z \Rightarrow \mathit{false}] \quad , &
\end{array}
$$

and thus we have proved (20) with the choices made in steps (B) and (F), i.e.

(22) $[(b \wedge d);f \wedge e \Rightarrow b;(\sim d;e \wedge f)]$,

a theorem, even uglier than "the first ugly theorem" of [1].

Remark We have used (19) instead of the formulation of the right-exchange from [2]

(23) $[x;y \Rightarrow z] \equiv [\sim x;\neg z \Rightarrow \neg y]$

and (21) instead of what (17) would have given, viz.

$$[b;c \Rightarrow z] \Rightarrow [a \Rightarrow z] \quad ,$$

thus avoiding the need of introducing negations and moving terms back and forth with the shunting rule

$$[x \Rightarrow y \vee z \equiv x \wedge \neg y \Rightarrow z].$$

In [1], this economy is achieved by the introduction of the "somewhere" operator. (End of Remark.)

Step A is not surprising because (21) was intentionally chosen so as to make the right-exchange applicable to its antecedent.

The purpose of step B is to apply the most general substitution for b or c

that makes continued manipulation possible. For b, I could not find a productive substitution: after b:= d ∨ e, we can use composition's monotonicity as in (15) but that creates 2 occurrences of dummy z, and after b:= d ∧ e, we are stuck because then composition's monotonicity as in (14) works in the wrong direction. If, however, we substitute a composition like ~d;e for c, then monotonicity as in (14) works in the right direction. The inclusion of the additional "∧ f" is, in a sense, obligatory if we wish to keep our options open: since c occurs as conjunct and conjunction is associative, the additional "∧ f" does not hamper the combination of the two compositions, and, furthermore, omission of the conjunct "∧ f" can be interpreted as a possibly premature instantiation f:= true. (The choice of "~d;e" instead of "d;e" is an irrelevant matter of elegance; it allows us to ignore that "~" is its own inverse.)

Step C' exploits the monotonicity of ";" (plus the usual ones from predicate calculus), and, in view of our

target, our remaining duty is to extricate dummy z from the composition.

Step E performs this extraction by – what else? – a second appeal to right-exchange after step D has prepared the ground. (Step D could have been postponed, but then later simplification would have needed that "~" is its own inverse.)

Our final step F embodies the recognition of what to choose for "a" and thus we have designed and proved theorem (22).

§ 8. In [2], left- and right-conditions are defined by

(24a) (p is a left-condition) ≡ [p; true ≡ p]

(24b) (q is a right-condition) ≡ [true; q ≡ q] ;

we now focus our attention on the right-condition.

In [2], Feijen & van Gasteren point out that – because [true; x ⇐ x] for all x – definition (24b) is equivalent to

(25) (q is a right-condition) ≡ [true; q ⇒ q] ,

the right-hand side of which is formally weaker than that of (24b). Lincoln A. Wallen has taught us a greater awareness of monotonicity arguments and has urged us to notice when we used only one direction of a mutual implication. To prove that a relation is a right-condition, (25) is more convenient for the formulation of the demonstrandum than (24b).

When trying to use that a relation is a right-condition, formally stronger characterizations are, in general, to be preferred. The stronger (24b) captures $[true; x \Leftarrow x]$ – which follows from the existence of a neutral element of the composition and the latter's monotonicity –. Here I propose another strengthening of (25), one that captures composition's monotonicity (in its left argument):

(26) (q is a right-condition) $\equiv \langle \forall x :: [x;q \Rightarrow q] \rangle$.

We shall use this definition (twice) to prove the following theorem:

(q is a right-condition) $\Rightarrow$ ($\neg q$ is a right-condition)

Proof We observe for arbitrary relation q

$$
\begin{array}{ll}
 & (\neg q \text{ is a right-condition}) \\
= & \quad \{(26) \text{ with } q := \neg q\} \\
 & \langle \forall x :: [x; \neg q \Rightarrow \neg q] \rangle \\
= & \quad \{(23), \text{ i.e. right-exchange}\} \\
 & \langle \forall x :: [\sim x; q \Rightarrow q] \rangle \\
\Leftarrow & \quad \{\text{instantiation } x := \sim x\} \\
 & \langle \forall x :: [x; q \Rightarrow q] \rangle \\
= & \quad \{(26)\} \\
 & (q \text{ is a right-condition})
\end{array}
$$

(End of Proof.)

In both [1] and [2], the proof of this theorem appeals to $[\sim true \equiv true]$, whereas the above proof does not use any properties of $\sim$ or true. The manipulative disadvantage of (25) is that it contains a constant whose properties we have to use.

The heuristic significance of formulation (26) is that it almost dictates monotonicity arguments because the fact that prefixing q by an arbitrary "$x;$" weakens it, fully captures that q is a right-condition. In other words, if, for instance, we set ourselves to prove that for a right-condition q

$$[q; true; \sim q \equiv q; \sim q] \quad , \tag{27}$$

the implication sign in (26) tells us that a ping-pong argument is appropriate.

Remark When faced with a demonstrandum of the form $[A \equiv B]$ we have a demonstrandum that does not depend (anti)monotonically on either A or B. Hence, proving this directly can only be done by appealing to Leibniz's Principle, i.e. by value-preserving transformations. (We have seen an example of this in section 1.) Rewriting the demonstrandum as mutual implication $[A \Rightarrow B] \wedge [A \Leftarrow B]$ yields two conjuncts that do depend (anti)monotonically on both A and B! Because we like to avoid avoidable ping-pong arguments, it is nice to recognize circumstances under which they are indicated. (End of Remark.)

In proving (27) we observe for ping:

$$
\begin{array}{ll}
 & [q;true;\sim q \Leftarrow q;\sim q] \\
\Leftarrow & \quad \{\text{monotonicity of } ;\} \\
 & [q;true \Leftarrow q] \\
= & \quad \{[x;true \Leftarrow x]\} \\
 & true\ ,
\end{array}
$$

a demonstration in which we did not use that q is a right-condition. For pong we observe that we can strengthen the right-hand side:

$$[q;true;\sim q \Rightarrow q;\sim q]$$

$\Leftarrow$ { (26) with $x := q;true;\sim q$ }
$[q;true;\sim q \Rightarrow q;true;\sim q;q;\sim q]$
$\Leftarrow$ { monotonicity of ; }
$[\sim q \Rightarrow \sim q;q;\sim q]$
$=$ { (28) with $b := \sim q$ }
true ,

where the "seesaw lemma" – see [1] –

(28) $[b \Rightarrow b;\sim b;b]$

follows from (22) with $d,e,f := b,b,true$.
(Knowledge of the seesaw lemma admittedly helps in the choice of how to instantiate x in (26) .)

§ 9. In [1] "middle-conditions" are introduced by

(29) (c is a middle-condition) $\equiv [c \Rightarrow J]$,

where "J" is the neutral element of ";" .
Also here the constant can be eliminated from the definition: we could have defined

(30) (c is a middle-condition) $\equiv \langle \forall x :: [c;x \Rightarrow x] \rangle$

or

(31) (c is a middle-condition) $\equiv \langle \forall x :: [x;c \Rightarrow x] \rangle$.

In other words, c being a middle-condition is captured by the fact that the prefix operator "c;" and the postfix operator ";c"

are strengthening operators. The implication signs in these definitions tell us that, for instance,

(32) $[c \equiv \sim c]$ for middle-condition c

has to be proved by mutual implication. Since $[\sim x \Rightarrow x] \equiv [x \Rightarrow \sim x]$, it suffices to prove $[c \Rightarrow \sim c]$. We observe to this end

$$
\begin{array}{ll}
 & \sim c \\
\Leftarrow & \quad \{(30) \text{ with } x := \sim c\} \\
 & c ; \sim c \\
\Leftarrow & \quad \{(31) \text{ with } x := c ; \sim c\} \\
 & c ; \sim c ; c \\
\Leftarrow & \quad \{(28) \text{ with } b := c\} \\
 & c \qquad .
\end{array}
$$

Our wish to apply strengthening operators suggested to start at the consequent and the first two steps then followed. (And we were just lucky that we did not try to prove $[\sim c \Rightarrow c]$ according to the above scheme.)

§ 10. Let me conclude. We set out to explore for antisymmetric and reflexive relations the interplay between calculating with equalities and inequalities, the latter drawing our attention to monotonicity arguments. In passing we encountered a number of instances – (2), (3), (16), (26), (30), (31) –

where, sometimes to clear advantage, a formula or definition could be rewritten as a universally quantified expression.

A closely related topic is the design of calculational proofs of theorems about extreme solutions. Since this topic has been dealt with in [0] we shall not pursue it here. To close I shall borrow an example from [1] to show that also here universal quantification can be used to eliminate a constant. Instead of defining the reflexive transitive closure by

$$(s^* \text{ is the reflexive transitive closure of } s) \equiv (s^* \text{ is the strongest solution of } x: [\,] \vee s;x \Rightarrow x])$$

we can define s^* equivalently by

$$\langle \forall t :: s^*;t \text{ is the strongest solution of } x: [\,t \vee s;x \Rightarrow x] \rangle \quad .$$

We shall not repeat the standard proof here (which is, as is to be expected, a ping-pong argument). We do wish to point out that the instantiation $t := J$ returns the original definition.

Acknowledgements Anyone familiar with their writings will realize that I am greatly endebted to Rutger M. Dijkstra,

W.H.J. Feijen, and A.J.M. van Gasteren.

[0] Predicate Calculus and Program Semantics,
Edsger W. Dijkstra & Carel S. Scholten,
Springer-Verlag, 1990

[1] Relational Calculus and Relational
Program Semantics,
Rutger M. Dijkstra
1992

[2] An Introduction to the Relational
Calculus,
W.H.J. Feijen & A.J.M. van Gasteren
Appeared in:
C.S. Scholten Dedicata: Van oude
machines en nieuwe rekenwijzen,
W.H.J. Feijen & A.J.M. van Gasteren (Eds.)
Schoonhoven, Academic Service, 1991

Austin, 11 April 1993

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
USA