Edsger W. Dijkstra Carel S. Scholten

Predicate Calculus and Program Semantics

谓词演算和程序语义〔英〕

Springer-Verlag
World Publishing Corp

Edsger W. Dijkstra Carel S. Scholten

Predicate Calculus and Program Semantics

Springer-Verlag
World Publishing Corp

Edsger W. Dijkstra
University of Texas at Austin
Austin, TX 78712-1111
USA

Carel S. Scholten
Klein Paradys 4
7361 TD Beekbergen
The Netherlands

Series Editor

David Gries
Department of Computer Science
Cornell University
Ithaca, NY 14853
USA

Library of Congress Cataloging-in-Publication Data
Dijkstra, Edsger Wybe.
Predicate calculus and program semantics / Edsger W. Dijkstra, Carel S. Scholten.
p. cm. — (Texts and monographs in computer science)
ISBN 0-387-96957-8 (alk. paper)
1. Predicate calculus. 2. Programming languages (Electronic computers)—Semantics. I. Scholten, Carel S. II. Title. III. Series.
QA9.35.D55 1989
511.3—dc20 89-11540

Index prepared by Jim Farned of The Information Bank, Summerland, California.

Reprinted by World Publishing Corporation, Beijing, 1991
for distribution and sale in The People's Republic of China only
ISBN 7-5062-1044-4

ISBN 0-387-96957-8 Springer-Verlag New York Berlin Heidelberg
ISBN 3-540-96957-8 Springer-Verlag Berlin Heidelberg New York

CHAPTER 0
Preface

This booklet presents a reasonably self-contained theory of predicate transformer semantics. Predicate transformers were introduced by one of us (EWD) as a means for defining programming language semantics in a way that would directly support the systematic development of programs from their formal specifications.

They met their original goal, but as time went on and program derivation became a more and more formal activity, their informal introduction and the fact that many of their properties had never been proved became more and more unsatisfactory. And so did the original exclusion of unbounded nondeterminacy. In 1982 we started to remedy these shortcomings. This little monograph is a result of that work.

A possible —and even likely— criticism is that anyone sufficiently versed in lattice theory can easily derive all of our results himself. That criticism would be correct but somewhat beside the point. The first remark is that the average book on lattice theory is several times fatter (and probably less self-contained) than this booklet. The second remark is that the predicate transformer semantics provided only one of the reasons for going through the pains of publication.

Probably conditioned by years of formal program derivation, we approached the task of designing the theory we needed as an exercise in formal mathematics, little suspecting that we were heading for a few of the most pleasant surprises in our professional lives. After a few notational adaptations of the predicate calculus —so as to make it more geared to our manipulative needs— and the adoption of a carefully designed, strict format for our proofs, we found ourselves in possession of a tool that surpassed our wildest expectations. As we got used to it, it became an absolute delight to work with.

The first pleasant —and very encouraging!— experience was the killing of the myth that formal proofs are of necessity long, tedious, laborious, error-prone, and what-have-you. On the contrary, our proofs turned out to be short and simple to check, carried out —as they are— in straightforward manipulations from a modest repertoire.

For quite a while, each new and surprisingly effective proof was a source of delight and excitement, although it was intellectually not fully satisfactory that each of them was a well-crafted, but isolated, piece of ingenuity. We had our second very pleasant surprise with the development of heuristics from which most of the arguments emerged most smoothly (almost according to the principle "There is really only one thing one can do."). The heuristics turned the design of beautiful, formal proofs into an eminently teachable subject.

This experience opened our eyes for further virtues of presenting formal proofs in a strict format. Proofs thus become formal texts meeting precise consistency criteria. The advantage of this transition is traditionally viewed as a gain in trustworthiness: given the formal text, there need not be an argument whether the proposed proof is a proof or not since it can be checked objectively whether the text meets the criteria. (This is the aspect stressed in mechanical proof verification.) A further advantage of the transition is that it enables a meaningful comparison between alternative proofs of the same theorem, viz., by comparing the texts. The final advantage of the transition is the much greater homogeneity it introduces into the task of proof design, a task that becomes a challenge in text manipulation that is more or less independent of what the theorem was about. It is this greater homogeneity that opens the way for heuristics that are more generally applicable.

In the course of the process we profoundly changed our ways of doing mathematics, of teaching it, and of teaching how to do it. Consequently, this booklet is probably as much about our new appreciation of the mathematical activity as it is about programming language semantics. It is certainly this second aspect of the booklet that has induced us to go through the aforementioned pains of publication. It has taken us some time to muster the courage to admit this, for a revision of mathematical methodology seemed at first a rather presumptuous undertaking. As time went on, however, we were forced to conclude that the formal techniques we were trying out had never been given a fair chance, the evidence being the repeated observation that most mathematicians lack the tools needed for the skilful manipulation of logical formulae. We gave them a fair chance; the reader is invited to share our delight.

* * *

Before the reader embarks on the study of the material proper of this booklet, we would like to give him some advice on reading it.

The most important recommendation is to remember all the time that this is *not* a treatise on logic, or on the foundations of mathematics. This warning is all the more necessary since superficially it might look like one: it has logical symbols all over the place and starts with a long introduction about notation. The frequent occurrence of logical symbols has a simple, pragmatic explanation: we use them so extensively because they are so well-suited for our job. The long introduction on notation has an equally simple, pragmatic explanation: some new notation had to be introduced and, more importantly, existing notations had to be adapted to our manipulative needs. Our free use of the logical connectives before their formal introduction stresses once more that this is not a book on logic. In short, the logician is perfectly free to be taken aback by our naive refusal to make some of his cherished distinctions.

Our second recommendation to the reader is to approach this little monograph with an open mind and not to get alarmed whenever it deviates from the traditions in which he happens to have been educated. In particular, he should not equate convenience with convention. Doing arithmetic in Arabic numerals is objectively simpler —i.e., more convenient— than doing it in Roman numerals. The transition from verbal arguments appealing to "intuition" or "common sense" to calculational reasoning admits also in that area an equally objective notion of simplicity —i.e., of convenience— . We know from experience that for readers from some cultures it will be hard to accept that we leave all sorts of (philosophical or psychological) questions unanswered; the only answer we can offer is that we are from a pragmatic culture that deals with such questions by not raising them.

Our third recommendation to the reader is really a request, viz., to honour this booklet's brevity by reading it slowly. Our texts have a tendency of being misleadingly smooth and unusually compact. When, at the end, you wonder "Was this all?", we shall answer "Yes, this was all. And we hope you travelled long and far.".

* * *

The above was written a year ago, before we had started on our manuscript. It reflects our expectations. Now, 12 months and 420 handwritten pages later, we can look back on what we have actually done besides breaking in a new fountain pen. Needless to say, the major discrepancy between dream and reality has been the size: had we foreseen a 420-page manuscript, we would not have referred to a "booklet". Our only excuse is that, at the time, we had not firmly decided yet to include the material now covered in the last three chapters.

The trouble with writing a book is that it takes time and that, consequently, at the time of completion, authors are older —and perhaps wiser— than when they started. Our fate has been no exception: there are several things we would have done differently had we written them a year later. (For instance, concerns about punctuality would have been more concentrated and our treatment of the implication in Chapter 5 might have relied less heavily on the "Little Theory".) With the exception of a complete rewriting of Chapter 1, which at the end was no longer a proper introduction to the rest of the book, we have, however, abstained from any major overhauls of the manuscript. They would not have solved the problem that authors may continue to grow and that, consequently, texts have an unavoidable tendency to be more or less dated. It is in this connection that we would like to include a general disclaimer: though our enthusiasm might sometimes suggest differently, we nowhere pretend that our work leaves no room for improvement, simplification, or meaningful generalization. (We did, for instance, not explore what meaningful partial orders on programs could be introduced.)

Apart from exceeding the originally envisaged size, we have remained faithful to our original intentions, in particular to our intention of writing a monograph reflecting the current state of our art. Though we have successfully covered most of its material in graduate courses at both sides of the Atlantic Ocean, this book should not be regarded (or judged) as a textbook, because it was not intended that way. (Hence, for instance, the total absence of exercises.) There were several reasons for not wishing to write a textbook. A practical reason was that different educational systems promote totally different perceptions of student needs to be catered for in an "acceptable" textbook, and that we did not want to waste our time trying to meet all sorts of conflicting requirements. A more fundamental reason was that we think the whole notion of "a textbook tailored to student needs" certainly at graduate level much too condescending. At that stage there is nothing wrong with confronting students with material for which they have not been adequately prepared, for how else are they going to discover and to learn how to cope with the unavoidable gaps in their education? So, though we know that this book can provide the underlying material for a fascinating and highly instructive course, it is *not* a textbook and its "target audience" is just "To whom it may concern".

We have, of course, our idea about whom it concerns: the mathematically inclined computing scientists and the mathematicians with methodological and formal interests. We most sincerely hope to reach them, to thrill them, and to inspire them to improve their own work or ours. Honesty compels us to add to this wish that there is one possible —and, alas, likely— "improvement" we are not waiting for, viz., the translation of our theory into set-theoretical terminology —by interpreting predicates as characteristic

functions of subsets of states— so as to make it all more familiar. Little is so regrettable as to see one's work "improved upon" by the introduction of traditional complications one has been very careful to avoid. Hybrid arguments, partly conducted in terms of a formal system and partly conducted in terms of a specific model for that formal system, present a typical example of such confusing complications. In this connection we would like to stress that the existence of individual machine states enters the picture only when our theory is applied to program semantics, i.e., to an environment in which the individual machine state is a meanigful concept; the theory itself does not need a postulate of the existence of individual states and, therefore, should not be cluttered by their introduction.

* * *

It is a pleasure to mention here our great appreciation for our former employers, Burroughs Corporation and N.V. Philips, respectively, which loyally supported us when we embarked in 1982 on the investigations reported in this volume.

More profound gratitude than to anyone else is due to W. H. J. Feijen and A. J. M. van Gasteren who were at that time close collaborators of one of us (EWD). They did not only witness from close quarters our first formal efforts at establishing a theory of predicate transformer semantics, they can trace their contributions and their influence all through this book. Feijen is essentially the inventor of our proof format; he took the decision to write the hint between the formulae connected by it and he was the one who insisted on the discipline of allotting at least a full line to each hint. (He probably realized that this insistence was necessary for establishing a tradition in which the hints given would be as explicit as he wanted them to be.) Van Gasteren provided the rationale for this invention (and also for the invention of the square brackets to denote the "everywhere" operator): her earlier explorations had convinced us that the type of brevity thus obtained is indispensable. Later she insisted on exorcizing mathematical rabbits —pulled out of a hat— and provided the "follows from" operator as one of the means to that end.

Furthermore, we thank all colleagues, students, and members of the Tuesday Afternoon Clubs in Eindhoven and Austin, whose reactions to the material shown have helped in shaping this book. We are particularly grateful for the more detailed comments that Jayadev Misra and Lincoln A. Wallen gave on the almost final version of the manuscript; the decision to rewrite Chapter 1 has been triggered by their comments.

Finally, we express our gratitude to W. H. J. Feijen, David Gries, and Gerhard Rossbach of Springer-Verlag, New York. In their offers of assistance

in the final stages of book production, each of them has gone beyond the call of duty.

July 1988
Austin, TX, USA
Beekbergen, The Netherlands

Edsger W. Dijkstra
Carel S. Scholten

Contents

CHAPTER 1
On structures

The proofs in this book are much more calculational than we were used to only a few years ago. As we shall explain later, the theorems are (or could be) formulated as boolean expressions, for which, in principle, *true* and *false* are the possible values; the proofs consist in calculations evaluating these boolean expressions to *true* . We shall return to this later, focussing, for the time being, our attention on some of the notational consequences of this approach.

The advantages of such a calculational style are a fairly homogeneous proof format and the possibility of obtaining brevity without committing the "sin of omission", i.e., making such big leaps in the argument that the reader is left wondering how to justify them. In fact, all our steps are simple and they are taken from a repertoire so small that the reader can familiarize himself with it as we go along. We could, however, harvest these advantages of calculation only by adoption of carefully chosen notational conventions that tailored our formulae to our manipulative needs. (Among those needs we mention nonambiguity, brevity, and not being forced to make needless distinctions.)

One of our notational conventions may strike the reader as a gross overloading of all sorts of familiar operators: for instance, we apply the operators from familiar two-valued logic to operands that in some admissible models may take on uncountably many distinct values. The justification for such a notational convention is a major purpose of this introductory chapter; we feel that we owe the reader such a justification, all the more so because in the world of programming the dangers of overloading are well known.

We can get some inspiration —and, if we feel the need for it, even some reassurance— from the field of physics. Every classical physicist, for instance, is thoroughly familiar, be it in his own way, with the notion of a vector in three-dimensional Euclidean space, independently of the question of whether the vector is a displacement, a velocity, a force, an acceleration or a component of the electromagnetic field. Also, he is equally familiar with the sum $v + w$ of two vectors v and w. But that sum raises a question, in particular if one adopts the view —as some physicists do— that the variables stand for the physical quantities themselves and not for their measure in some units. The question is, how many different vector additions are used by the physicist: is the sum of two velocities the same sort of sum as the sum of two forces? Well, the answer seems negative in the sense that no physicist that is well in his mind will ever add a velocity to a force.

Given the fact that in some way we can distinguish those different sorts of additions, we could feel tempted or intellectually obliged to introduce as many different addition symbols as we can distinguish additions, say $+_v$ for the addition of velocities, $+_a$ for the addition of accelerations, etc. In a purist way, this would be very correct, but we all know that the physical community has decided against it: it has decided that a single symbol for addition will do.

When challenged to defend that decision, the physicist will give the following reasons. Firstly, the purist convention would complicate manipulation: the single rule that differentiation distributes over addition, i.e.,

$$\frac{d}{dt}(v + w) = \frac{dv}{dt} + \frac{dw}{dt}$$

would emergy in many forms, such as

$$\frac{d}{dt}(v +_v w) = \frac{dv}{dt} +_a \frac{dw}{dt} \quad ,$$

in each of which the distribution law has practically been destroyed. Secondly, the physicist would point out that in every physical context the subscripts of the addition symbols are really redundant because they follow from the type of vectors added. And, finally, he would point out that the use of a single addition symbol never seduces him to add a velocity to a force since the incentive to do so never arises. The defence is purely pragmatic.

The physicist goes further. With respect to a point mass m, he introduces a gravitational potential G, which in some considerations is treated as a single physical object, for instance, in the sentence "the dimension of the gravitational potential is $length^2/time^2$". Also potentials can be "added" by the physicist: a system consisting of two point masses, more precisely of point mass $m0$ with gravitational potential $G0$ and a point mass $m1$ with

gravitational potential $G1$, gives rise to a gravitational potential $G0 + G1$. What kind of addition is that?

A possible answer is to shrug one's shoulders and to say "Mostly the usual one: it is symmetric and associative and there is a zero potential 0 satisfying $G + 0 = G$ for any potential G . Furthermore, it has some special properties in connection with other operators that are specific for potentials, e.g., the nabla operator ∇ distributes over it:

$$\nabla(G0 + G1) = \nabla G0 + \nabla G1 \quad ,$$

but that is really another story that more belongs to the nabla operator to be introduced later.".

The helpful physicist will certainly give you a much more detailed answer: he will tell you that a potential assigns a scalar value to each point of three-dimensional space and, conversely, is fully determined by those values —i.e., two potentials are the same potential if and only if they are everywhere equal— ; he will furthermore tell you that, by definition, in any point of three-dimensional space, the value of $G0 + G1$ equals the sum of the values of $G0$ and $G1$ in that point. By the convention of "point-wise addition" he thus defines the addition of potentials in terms of addition of real numbers.

Remark In the same vein he will define the nabla operator as a differentiation operator. That the nabla distributes over addition then emerges as a theorem. (*End of Remark.*)

Once we have chosen a coordinate system for the three-dimensional Euclidean space, say three orthogonal coordinates x , y , z , another view of a potential presents itself. We then have the option of viewing the potential as an expression in the coordinates, i.e., we equate for some function g ,

$$G = g.(x,y,z) \quad ,$$

where the function is such that, for any triple (a,b,c) , the value of $g.(a,b,c)$ equals the value of the potential G at the point that has that triple (a,b,c) as its coordinates.

This last view gives a very familiar interpretation to the plus sign in $G0 + G1$: the latter formula now stands for the expression

$$g0.(x,y,z) + g1.(x,y,z) \quad ,$$

i.e., our plus sign just adds two expressions to form a new expression that is their sum.

This view thus allows a very familiar interpretation for the plus sign in $G0 + G1$; the price we have to pay for that convenience is the introduction of

names, like $G0$ and $G1$, that stand for expressions in, say, x, y, and z but, being just names, do not state that dependence explicitly (as a functional notation like $g.(x,y,z)$ would have done).

Modern mathematical usage freely introduces names for all sorts of mathematical objects such as sets, points, lines, functions, relations, and alphabets, but is reluctant to introduce a name for an expression in a number of variables. There is a very good reason for that reluctance: because of their hidden dependence on some variables, those names may become quite tricky to manipulate. (An example of how tricky it may become is provided by the names "yesterday", "today", and "tomorrow", which admit sentences such as "Tomorrow, today will be yesterday.".)

When physicists call a potential "G", they do precisely what, for good reasons, the mathematicians are very reluctant to do; they adopt a mathematically dubious convention. The reason why they get away with it —at least most of the time— is probably that three-dimensional space (plus time) is the standard context in which almost all of classical physics is to be understood. With that understanding, any notation —like $g.(x,y,z)$ — that indicates that dependence explicitly is unnecessarily lengthy.

We mentioned the physicists because we are partly in the same position as they are. For the sake of coping with programming language semantics we shall develop a theory, and we may seem to have adopted the "dubious convention" of the physicists in the sense that, when the theory is applied to programming language semantics, things that were denoted by a name in our theory stand in the application for expressions in programming variables.

Remark Once we have chosen a set of Cartesian coordinates for the three-dimensional space, we have introduced a one-to-one correspondence between the points of space and the triples of coordinate values. For programs, this has given rise to the metaphor that is known as "the state space". For a program that operates on n variables, the corresponding "state space" is visualized as an n-dimensional space with the n variables of the program as n Cartesian coordinates. Thus the metaphor introduces a one-to-one correspondence between the points in state space and all combinations of values for the n variables. Since each combination of these values corresponds to a state of the store consisting of those variables, there is a one-to-one correspondence between the states of the store and the points in state space; hence the latter's name. It is a well-established metaphor, and we shall use it freely. Instead of "for any combination of values of the program variables" we often say "for any point in state space", or "everywhere in state space" or "everywhere" for short. Another benefit of the metaphor is that we can describe the sequence of states corresponding to a computation as a "path" through state space, which is traversed by that computation. Further-

more, the metaphor allows us to view certain program transformations (in which program variables are replaced by others) as coordinate transformations.

In short, the notion of state space has its use. The reader that encounters the term for the first time should realize that this use of the term "space" represents a considerable generalization of normal three-dimensional space: there is in general nothing three-dimensional about a state space, its coordinates are rarely real, and Euclidean distance between two states is not a meaningful concept (not even if the program variables are of type integer and the state space can be viewed as consisting of grid points). (*End of Remark.*)

Upon closer inspection it will transpire that our convention is not half as dubious as it may appear at first sight. Hidden dependence on a variable makes manipulation tricky only in contexts in which the variable occurs explicitly as well. We are safe in the sense that our theory is developed independently of its application, and that it is only in the application that names occurring in the theory are made to stand for expressions in program variables. We beg the reader to remember that the state space is not an intrinsic ingredient of our theory and that it only enters the picture when we apply our theory to programming language semantics.

In fact we go further and urge the reader to try to read the formulae of our theory without interpreting names as expressions in the coordinates of some state space. Such an interpretation is not only not helpful, because it is confusing, but is even dangerous, because expressions on a state space provide an overspecific model for what our theory is about, and as a result the interpretation might inadvertently import relations that hold for the specific model but do not belong to the theory.

Because the theory is to be understood independently of its application to programming language semantics it would not do to call those names in the theory variables of type "expression". We need another, less committing, term. After quite a few experiments and considerable hesitation, we have decided to call them variables of type "structure". So, in our theory we shall use names to stand for "structures".

Our notion of a "structure" is an abstraction of expressions in program variables in the sense that the state space with its individually named dimensions has been eliminated from the picture. We do retain, however, that expressions in program variables have types, i.e., they are boolean expressions or integer expressions, etc.; similarly our theory will distinguish between "boolean structures", "integer structures", etc.

The reader who is beginning to wonder what our structures "really" are should control his impatience. The proper answer to the question consists in the rules of manipulation of formulae containing variables of type structure. In due time, these rules will be given in full for boolean structures, which are by far the most important structures with which we shall deal. (We leave to the reader the exercise of moulding Peano's Axioms into the manipulative rules for integer structures.) The impatient reader should bear in mind that this introductory chapter's main purpose is to give the reader some feeling for our goals and to evoke some sympathy for our notational decisions.

* * *

The next notational hurdle to take has to do with the notion of equality. Difficulty with the notion of equality might surprise the unsuspecting reader, who feels —not without justification— that equality is one of the most fundamental, one of the most "natural", relations. But that is precisely the source of the trouble! The notion of equality came so "naturally" that for many centuries it was quite common not to express it explicitly at all.

For instance, in Latin, in which the verb "esse" for "to be" exists, it is not unusual to omit it (e.g., "Homo homini lupus."). In mathematical contexts equality has been expressed for ages either verbally or implicitly, e.g., by writing two expressions on the same line and leaving the conclusion to the intelligent reader. We had in fact to wait until 1557, when Robert Recorde introduced in his "Whetstone of witte" the —in shape consciously designed!— symbol = to denote equality. In the words of E. T. Bell: "It remained for Recorde to do the right thing.".

Yes, Recorde did the right thing, but it took some time before it began to sink in. It took in fact another three centuries before the equality sign gained in principle the full status of an infix operator that assigned a value to expressions of the form $a = b$. The landmark was the publication of George Boole's "Laws of Thought" in 1854 (fully titled *An Investigation of The Laws of Thought on which are founded The Mathematical Theories of Logic and Probabilities*). Here, Boole introduced what is now known as "the boolean domain", a domain comprising two values commonly denoted by "*true*" and "*false*", respectively. In doing so, he gave $a = b$ the status of what we now call a "boolean expression".

Yes, Boole too did the right thing, but we should not be amazed that his invention, being less than 150 years old, has not sunk in yet and that, by an large, the boolean values are still treated as second-class citizens. (We should not be amazed at all, for the rate at which mankind can absorb progress is strictly limited. Remember that Europe's conversion from Roman numerals to the decimal notation of Hindu arithmetic took at least six centuries.)

After this historical detour, let us return for a moment to our physicist with his potentials. In terms of two given potentials $G0$ and $G1$, he is perfectly willing to define a new potential given by

$$G = G0 + G1 \quad .$$

Similarly, a mathematician is perfectly willing to define in terms of two given 10-by-10 real matrices A and B a new 10-by-10 real matrix C by

$$C = A + B \quad .$$

We said "similarly", and rightly so. A potential associates a real value with each of the points of three-dimensional, "physical" space; a 10-by-10 real matrix associates a real value with each of the 100 points of the two-dimensional 10-by-10 space spanned by the row index and the column index. The additions of potentials and of matrices are "similar" in the sense that they are to be understood as "point-wise" additions. That in the case of matrices, where the underlying space is discrete, it is customary to talk about "element-wise" addition is in this case an irrelevant linguistic distinction.

With this convention of point-wise application in mind it would have stood to reason to let $A = B$ stand for the 10-by-10 boolean matrix formed by element-wise comparison.

But this is not what happened. People were at the time unfamiliar with the boolean domain. Consequently, a boolean matrix was beyond their vision and hence a formula like the above $C = A + B$ was not read as a boolean matrix but was without hesitation read as a statement of fact, viz., the fact that C and $A + B$ were "everywhere" —i.e., element by element— equal. Similarly, $G = G0 + G1$ is interpreted as a statement of fact, viz., the fact that the potentials G and $G0 + G1$ are "everywhere" equal and not as an expression for a "boolean potential", *true* wherever G equals $G0 + G1$ and *false* elsewhere.

For real expressions A and B, $A = B$ stands for a boolean expression and, hence, for real structures we would like $A = B$ to denote a boolean structure, but the conventional interpretation reads $A = B$ as the fact that A and B are "everywhere" equal, that A and B are "the same". In retrospect we can consider this conventional interpretation an anomaly, but that is not the point. The point is: can we live with that anomaly in the same way as physicists have lived with it since potentials were invented and mathematicians have lived with it since matrices were invented?

It would be convenient if the answer were affirmative, as might be the case in a context in which there is no need for dealing with boolean structures. But ours is a very different context, for almost all our structures will be boolean. Hence we cannot live with the notational anomaly and we have to do something about it.