Journal of Universal Computer Science, vol. 7, no. 8 (2001), 631-640
submitted: 18/5/01, accepted: 21/8/01, appeared: 28/8/01 © Springer Pub. Co.

The Transition from VDL to VDM

C. B. Jones
(Department of Computing Science
University of Newcastle
NE1 7RU UK
cliff.jones@ncl.ac.uk)

In gratitude to Peter Lucas who is a generous and challenging colleague who (twice) aided me in moving to a delightful city which changed my life.

Abstract: This paper describes (one person's view of) how the *Vienna Development Method* grew out of the earlier work on the *Vienna Definition Language*. Both of these activities were undertaken at the IBM Laboratory Vienna during the 1960s and 70s.
Key Words: formal methods, language definition, VDL, VDM, operational semantics, denotational semantics.
Category: F.3 Logics and Meanings of Programs; F.3.2 Semantics of Programming Languages; D.2.4 Program Verification; D.3.1 Formal Definitions and Theory

1 Introduction

The so-called "Vienna Development Method" (VDM)[1] evolved –at the IBM Laboratory in Vienna– from the earlier work known as the "Vienna Definition Language" (VDL). It is often said that the key contribution of VDM (over VDL) is that the latter is based on denotational semantics whereas the former uses operational semantics. This statement somewhat trivialises the distinction and –at the same time– fails to record in detail the debt of the research in the 1970s to that of the 1960s. Furthermore, the glib characterisation completely ignores the fact that VDM has a far wider area of application than language semantics. The symposium in honour of Peter Lucas' retirement from Graz presented an ideal opportunity to reflect on the transition from VDL to VDM.

Of necessity, this is a personal view and I think it fair to emphasise this fact by breaking with normal scientific convention so that I can write in the first person singular. One reason that a look back at the VDL work was particularly appealing was that I have recently taught a course on "Understanding Programming Languages" and chose to base most of the lectures on operational semantics whereas, in the past, I had always taught denotational semantics. This afforded the opportunity to reflect on the real distinctions and contributions.

[1] References are given in the subsequent, more detailed, sections.

2 VDL and the 1960s

The most accessible detailed publication on VDL is [LW69]. The language which became known as "PL/I" was initially to have been called "New Programming Language" until the UK National Physical Laboratory pointed out that they had prior claim on the acronym "NPL". It was clear from its inception (for an account see [Rad81]) that PL/I was going to be a large language and it had also become obvious that even the semantics of a smaller, more focussed, language such as FORTRAN was beyond precise description by natural language alone. An effort was mounted by IBM researchers in the Hursley (UK) and Vienna laboratories to give a precise semantics to PL/I. The main contribution of the Hursley group was a series of "LDH" notes[2] which sketched models and commented on the more completely formal description being created in Vienna. One of the Hursley models has a key role in the story below.

2.1 Language definition

In the early 1960s, the idea of defining the semantics of a programming language was seen by key men of insight as an essential step to putting programming on a sound footing. Professor Heinz Zemanek –who was the director of the IBM Vienna Group– convened the first ever IFIP Working Conference at Baden-bei-Wien on the subject of "Formal Language Description Languages". The proceedings of the 1964 conference (published as [Ste66]) contain seminal papers and a record of the fascinating discussions (recorded by people in and around the Vienna group including Professor Hermann Maurer who thanked Peter Lucas publicly for this opportunity at the Graz Symposium).

A cornerstone of the subsequent VDL approach is John McCarthy's paper [McC66]. This indicates both the level of ambition and the main scientific idea of 1964. McCarthy proposes describing the semantics of a language (micro-ALGOL) by writing a recursive function that takes a program and a starting state and computes (if possible) a final state. This is of course the purpose of any interpreter for a language. But –rather than being written to run on a machine– McCarthy's *abstract interpreter* used abstractions of both the program object and of the state of the computation. (In fact, McCarthy's "abstract objects" have a specific part to play in the comparison of VDM with other approaches.)

It is important to note that in [McC66] there was no mention of handling errors, there were no abnormal jumps in Micro-ALGOL and that even the essential notion of ALGOL scope had not been handled. There was still work to be done. In the ensuing discussion, on hearing that Micro-ALGOL did not even have conditional statements, Christopher Strachey commented "All right.

[2] "Language Definition Hursley"; there was a similar series of "Language Definition Vienna" notes.

Minute Micro-ALGOL." McCarthy however claimed in Section 7 of his paper that "All of these difficulties can be resolved"; that is, he believed that the extension of his abstract interpreter idea to cover the whole of the semantics of ALGOL-60 was achievable and he went on to claim that this "will clarify the problem of compiler design".

From this seed, the Vienna group grew a huge tree. In fact, they have always insisted on also acknowledging the stimulus of Cal Elgot (e.g. [ER64]) and Peter Landin. One landmark was the publication of "Tentative Steps" [Ban65] which was an edited collection of views. Overall, this period of research produced definitions which were "operational" in the sense that they described the steps of an (abstract) interpreter: a program in a language had to be understood from the steps of its computation from a particular state.

The acronym "ULD" is for "Universal Language Document"; ULD-I was the name given to the natural language description of PL/I; ULD-II to Hursley's version; ULD-III was the internal name for the series of PL/I descriptions which came from the Vienna group. The first version was printed in 1966; the third and final version in April 1969. JAN Lee coined the name "Vienna Definition Language" and definitions of several other languages were written in VDL as well as a number of related books by researchers outside IBM.

2.2 Some evaluation of VDL

It is useful to catalogue some of the contributions to operational semantics made by the Vienna group in the 1960s.

- An appropriate collection of generic abstract collections was chosen (sets, maps and sequences).
- McCarthy's notion of abstract objects was enriched with a modification operator (μ).
- A way of handling jumps (and other abnormal terminations) was chosen.
- An approach to non-determinism was worked out.
- The consequences of the realisation that non-determinism was an adequate model of the parallelism inherent in PL/I's tasking concept were incorporated.
- An implicit characterization of the various ways that storage mapping could be done in PL/I was thought through (see [BW71]).

In addition to this list of resolved technical challenges, a notation had to be devised which made the overall description readable.

It is natural to ask what was the contribution in Gordon Plotkin's 1981 Arhus lecture notes [Plo81] which resulted in a revival of work on "Structured Operational Semantics". If one were to single out the most dramatic practical change it would have to be the inference rule style of presentation. This single piece of genius offered a natural way of handling non-determinism.

2.3 Justifying compiling algorithms

It is only at this point –1968– that I had any involvement in the Vienna story. I had been working on the testing of the first PL/I compiler in Hursley. We saw 635 hand-written test cases run successfully and we had automatic tools to generate unlimited numbers of further test cases. The PL/I compiler was debugged around these test cases, shipped, and fell over on an embarrassing number of customer programs. I became convinced that testing could never substantially increase the dependability of a product and that quality had to begin at the earliest stages of the design process. In April 1968, I went on a course about ULD-III in Vienna (fell in love with the city) and immediately expressed a strong interest in joining the Vienna group to understand how their formal descriptions could be used in compiler design. My first (two-year) stay in Vienna began in August 1968.

Hursley and Vienna had chosen different models to explain the idea of reference to local variables in blocks and procedures. There arose naturally the question of whether these two models were equivalent. Peter Lucas had the inspiration to link the two mechanisms into a more complicated machine which essentially combined and updated both sets of state components; he then proved a *data type invariant* which expressed that the hypothesised linkage was preserved by all operations. It was then argued that unnecessary "ghost variables" could be erased. For the subsequent discussion, it is important to note that this approach was general in that an arbitrary relation could be handled.

It was also telling that Peter Lucas attempted to single out this result from the whole language definition: he promoted the idea of separating proofs about implementations of *language concepts* which could be considered one at a time.

In the period 1968–70, we conducted many experiments in how a language description could be used as the basis for the design of a compiler. One of my contributions was to show that a representation (in the implementation) could be related to an abstraction (in the description) by means of a *retrieve function* (i.e. a homomorphism from the representation to the abstraction). Interestingly, this approach (subsequently used as the main approach to *data reification* in VDM) was strictly less powerful than Lucas' twin machine. We understood this but saw it as a useful heuristic that an abstraction should have no *implementation bias*. It was not until much later that the research with colleagues in Manchester (notably Tobias Nipkow and Lynn Marshall) showed me that there were occasions where the more general method was required. (An account of the work on data abstraction and reification is given in [Jon89].)

A contribution which was to have a more direct influence on the language semantics research in VDM resulted from my dissatisfaction with the way VDL definitions handled abnormal changes of sequence like goto statements (PL/I also has a complicated exception handling mechanism called "on units"). Derived from an earlier internal report, [HJ71] was the external publication which introduced the *exit* construct; this was to play an interesting part in the debate between Oxford denotational semantics and VDM.

It is difficult to convey the excitement of that time. We had frequent seminars at which we presented new ideas for proofs about –or based on– language descriptions. I remember one where Wolfgang Henhapl trailed a "mathematician's approach" to the proof of the block concept and showed one line consisting of a citation to an earlier proof of the result: the non-trivial point was to question whether we were actually building on each others' work or just playing with the same theorem time and again; this argument was countered with the claim that we were interested in method and not just results.

The most stimulating seminars were those given by Dana Scott on a visit late in 1969. We in Vienna had been struggling to understand fully Floyd's method and actually invited Scott –who had attended an IFIP WG2.2 meeting in Vienna– to spend a week with us to discuss [Flo67]. Dana was fortunately not constrained by our intentions and actually presented his evolving work with Jaco de Bakker; the manuscript [dBS69] is a gem and was one of our first exposures to what was to become the denotational approach.

The appeal for a different approach could not have found more fertile soil. In our proofs, we had found on a number of occasions that the potential flexibility of an operational definition could make it far harder to prove results. For example, [JL71] represents a rather careful argument for the correctness (with respect to an operational description of the relevant language concept) of a standard compiling technique for reference to local block variables: the axiom which was most tedious to prove established only that the *environment* was the same before and after any statement that was executed. In spite of seeing the need for an alternative approach, I was less than convinced by the mathematics that Dana needed and he claimed he was "cut to the quick" on one occasion when I asked if it was all really necessary.

But, this brings the story to the point of Section 3 before which a few other Vienna contributions are worthy of note.

2.4 Other gems

The VDL definition of PL/I was huge; many researchers thought the enterprise a waste of time (and paper). The 500 copies printed (on very thin paper) would –it was claimed– be higher than *Stephansdom* if stacked. It would have been difficult to type and impossible to control the layout when changed had it not been for a wonderful automatic layout system "Formula 360" [KS69] (and what a pleasure it was to meet –among many old friends– Fritz Schwarzenberger in Graz). This pearl of a system automatically chose line breaks within long formulae by cutting the parse tree as high as possible: a brilliantly simple and effective rule.

An almost completely overlooked fact is that the Vienna group published work on an axiomatic approach in the 1960s. In fact they used the *stack* as an example in a paper given to patent lawyers in 1969. Remember also that the storage component in VDL definitions was characterised axiomatically.

A whole series of papers from this research discussed various aspects of compiler design from formal descriptions; in addition to those cited above, ones which came readily to hand include [Hen68, Luc69, HJ70, Jon70, Luc71, HJ71, Luc72].

Although much of the research was conducted in Belfast under Tony Hoare's supervision, it is also fair to list [Lau71] as one of the first major attempts to link language definitions with proof rules for results about programs written in the language. Peter Lauer's research was undoubtedly helped by colleagues in Vienna who are acknowledged in his thesis.

3 Transitional steps

There was then, in Vienna by 1970, a strong awareness that the operational VDL definitions were a possible –but not ideal– basis for formal compiler derivation. Such definitions could perhaps be compared to Roman numerals which were an adequate way of recording numbers but were far from ideal for their manipulation. Where were we to find our equivalent of the Arabic representation?

Hans Bekič had spent a year with Peter Landin at Queen Mary College, London from November 1968 to November 1969 and was keen that a more denotational approach should be taken. Hans was a mathematician by training (see [Bek84]) and had far less difficulty than other members of the group in understanding the role of, say, fixed points. It is perhaps one of the missed opportunities that Hans was in London when Dana Scott gave his seminars in Vienna during August 1969.

For my part, I returned to IBM's laboratory near Winchester for the years 1971/2 and ran an "Advanced Technology" group. One product of our work was to write a *functional semantics* of ALGOL 60. This report [ACJ72] combined the *exit* concept with the clear separation of the *environment* from the *state* and produced a description in which some of the properties which were messy to prove about a VDL description were immediately apparent. Functional semantics had many of the advantages of structured operational semantics.

The other lasting piece of work that was initiated at this time was the ideas (in particular, what became known as *data reification*) on program –as distinct from compiler– derivation: see [Jon72, Jon73]).

The Vienna group itself spent much of the period 1970/2 on the oft-repeated, but ultimately quixotic, venture of finding potential parallelism hidden in FORTRAN programs.

4 VDM and the 1970s

4.1 Language definition

In late 1972, the Vienna Laboratory was given the task of building a PL/I compiler for an evolving, novel, machine. I remember vividly the call from Peter Lucas when he told me about this; the invitation to transfer back to Vienna was hardly out of his mouth before I agreed.

We immediately started an exchange of notes on the style of a definition that would serve as a formal basis for the derivation of the compiler and the discussion converged on a sugared denotational style. The basic idea of a *denotational definition* is to map constructs of the language (to be described) homomorphically to some space of understood objects. For simple sequential languages, the chosen space of denotations could be functions from states to states. Although Hans Bekič was actively thinking about handling concurrency in the denotational approach, we were fortunate that the ECMA/ANSI committee who were standardising PL/I chose to drop the tasking feature of the language thus leaving us with a basically sequential language.

Once the group was all together, we had intensive discussions (one might even say arguments) about how various difficulties were to be tackled. The eventual decision to adopt a version of the earlier *exit* idea was to set us apart from the Oxford denotational school which used *continuations*. It has also been pointed out by Peter Mosses [Mos01] that the "combinators" used in [BBH$^+$74] are a form of the idea later known as *monoids*). We also chose not to use the *disjoint sum* idea in our abstract syntax, preferring to make an explicit distinction as to whether or not tags were inserted. This decision fitted well with the old VDL definition of abstract objects. In fact, the ways of building the basic (non-functional) objects passed almost unchanged from VDL to VDM.

One effect of the level of sugaring was that is was in nearly all cases possible to read the VDM descriptions as though they were operational. What then was the key advantage of the denotational style? I suppose I always felt that it was a way of cutting down on options, a way of keeping the definer honest by forcing thought about which things were really important. In practice, in say a compiler design, it was more important that one could see immediately that something could not change (because it was an auxiliary argument rather than in the state) than whether, say, procedure denotations were fully abstract. This is fortunate since full abstraction results have taken a long time to come and are not likely to be used on large definitions.

For the PL/I description, the *state* (or *semantic objects*) finally crystallised in one long coffee session (lasting to a late lunch) and this made it possible to fix the types of the main semantic functions. From this point, we were able to work fairly independently on separate parts of the definition. The eventual description [BBH+74] is almost 100 pages of formulae (accompanied by a "Part II" of similar length which provides commentary). Once again, many researchers questioned the wisdom of investing so much brain power in what was obviously an overly Baroque language but I think an enormous amount was learnt by confronting the description of a language which we could not bend to suit our formalism.

4.2 Compiler design

Our task was not simply to write a formal description of (ECMA/ANSI) PL/I but to build a compiler. Achieving this objective was made more difficult by the frequent changes in the architecture of the machine that was being designed in Poughkeepsie. We had an enormous number of telephone conversations and more stays in the Hudson Valley than I care to remember. It was key to our (evolving) approach that we had a firm grasp of the machine architecture. Initially, we were delighted with the fact that a group in Poughkeepsie led by Tony Peacock was writing a formal description of the machine. Unfortunately, US management decided that so much effort was being invested in this that it ought be an executable (and later an efficient) interpreter of the machine's instruction code. The consequent obfuscation of the description destroyed its value as a thinking aid and left us with no choice but to write our own formal description of the machine architecture. I wish I had to hand a copy of Hans Bekič's hand written description (in minute handwriting) which covered only a couple of pages.

In 1975, IBM decided to cancel the project to build the machine in question. Fortunately, the group dispersed (just) gradually enough that we wrote reports summarising the main steps of how we had been working. Again, my list is bound to be biased by the internal documents that I can find but

- the description of PL/I itself [BBH+74],
- initial experiments in compiler justification [BIJW75], and
- an outline of a method of [Jon76]

are worthy of mention. The major published summary of the language description and compiler development work (which includes the first proof of equivalence of *exits* and *continuations*) is [BJ78] which, when it finally went out of print, was reworked into [BJ82].

4.3 Program development

Most applications of VDM have nothing to do with language definition nor with compiler development. The parts of the Vienna Development Method aimed at "normal" program development were, of course, influenced by the work of the early 1970s but these were first published in book form in [Jon80]. An account of the distinctive features of these aspects of VDM has been published as [Jon99].

5 Looking back in gratitude

One of the most scientifically gratifying aspects of the (VDL and) VDM research is the impact that it has had on other formal methods research. It cannot be unfair to claim an influence on VVSL, RAISE, Larch and B.

Personally, the Vienna group was the most stimulating prolonged collaboration of my career and I am grateful to all of my erstwhile colleagues but a special closing word of thanks must go to Peter Lucas without whom I might not have been there (nor have been late for an opera the only time in my life).

References

[ACJ72] C. D. Allen, D. N. Chapman, and C. B. Jones. A formal definition of ALGOL 60. Technical Report 12.105, IBM Laboratory Hursley, August 1972.

[Ban65] K. Bandat. Tentative steps towards a formal definition of semantics of PL/I. Technical Report TR 25.056, IBM Laboratory, Vienna, July 1965.

[BBH+74] H. Bekič, D. Bjørner, W. Henhapl, C. B. Jones, and P. Lucas. A formal definition of a PL/I subset. Technical Report 25.139, IBM Laboratory Vienna, December 1974.

[Bek84] H. Bekič. *Programming Languages and Their Definition*, volume 177 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

[BIJW75] H. Bekič, H. Izbicki, C. B. Jones, and F. Weissenböck. Some experiments with using a formal language definition in compiler development. Laboratory Note LN 25.3.107, IBM Laboratory, Vienna, December 1975.

[BJ78] D. Bjørner and C. B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978.

[BJ82] D. Bjørner and C. B. Jones. *Formal Specification and Software Development*. Prentice Hall International, 1982.

[BW71] H. Bekič and K. Walk. Formalization of storage properties. In E. Engeler, editor, *[Eng71]*, pages 28–61. 1971.

[dBS69] J. W. de Bakker and D. Scott. A theory of programs. Manuscript notes for IBM Seminar, Vienna, August 1969.

[Eng71] E. Engeler. *Symposium on Semantics of Algorithmic Languages*. Number 188 in Lecture Notes in Mathematics. Springer-Verlag, 1971.

[ER64] C. C. Elgot and A. Robinson. Random access stored-program machines: An approach to programming languages. *Journal of the ACM*, 11:365–399, October 1964.

[Flo67] R. W. Floyd. Assigning meanings to programs. In *Proc. Symp. in Applied Mathematics, Vol.19: Mathematical Aspects of Computer Science*, pages 19–32. American Mathematical Society, 1967.

[Hen68] W. Henhapl. A proof of correctness for the reference mechanism to automatic variables in the F-compiler. Technical Report LN 25.3.048, IBM Laboratory Vienna, Austria, November 1968.

[HJ70] W. Henhapl and C. B. Jones. The block concept and some possible implementations, with proofs of equivalence. Technical Report 25.104, IBM Laboratory Vienna, April 1970.

[HJ71] W. Henhapl and C. B. Jones. A run-time mechanism for referencing variables. *Information Processing Letters*, 1:14–16, 1971.

[JL71] C. B. Jones and P. Lucas. Proving correctness of implementation techniques. In E. Engeler, editor, *[Eng71]*, pages 178–211. 1971.

[Jon70] C. B. Jones. Yet another proof of the correctness of block implementation. Technical Report LN 25.3.075, IBM Laboratory, Vienna, August 1970.

[Jon72] C. B. Jones. Formal development of correct algorithms: an example based on Earley's recogniser. *ACM SIGPLAN Notices*, 7(1):150–169, January 1972.

[Jon73] C. B. Jones. Formal development of programs. Technical Report 12.117, IBM Laboratory Hursley, April 1973.

[Jon76] C. B. Jones. Formal definition in compiler development. Technical Report 25.145, IBM Laboratory Vienna, February 1976.

[Jon80] C. B. Jones. *Software Development: A Rigorous Approach.* Prentice Hall International, 1980.

[Jon89] C. B. Jones. Data reification. In J. A. McDermid, editor, *The Theory and Practice of Refinement*, pages 79–89. Butterworths, 1989.

[Jon99] C. B. Jones. Scientific decisions which characterize VDM. In *FM'99 – Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 28–47. Springer-Verlag, 1999.

[KS69] K. Koch and F. Schwarzenberger. Introduction to Formula 360. Technical Report TR 25.101, IBM Lab Vienna, 12th December 1969.

[Lau71] P. E. Lauer. *Consistent Formal Theories of the Semantics of Programming Languages.* PhD thesis, Queen's University of Belfast, 1971. Printed as TR 25.121, IBM Lab. Vienna.

[Luc69] P. Lucas. Equivalence of the verification conditions of Floyd and Scott. LN 25.3.055, IBM Laboratory Vienna, 18th September 1969.

[Luc71] P. Lucas. Formal definition of programming languages and systems. In C. V. Freiman, editor, *Information Processing 71. Proceedings of the IFIP Congress 1971*, volume 1, pages 291–297. North-Holland, 1971.

[Luc72] P. Lucas. On the semantics of programming languages and software devices. In *[Rus72]*, pages 41–57. 1972.

[LW69] P. Lucas and K. Walk. *On The Formal Description of PL/I*, volume 6, Part 3 of *Annual Review in Automatic Programming.* Pergamon Press, 1969.

[McC66] J. McCarthy. A formal description of a subset of ALGOL. In *[Ste66]*, pages 1–12, 1966.

[Mos01] P. D. Mosses. What use is formal semantics? private communication, 2001.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[Rad81] G. Radin. PL/I. in [Wex81], 1981.

[Rus72] R. Rustin. *Formal Semantics of Programming Languages.* Prentice-Hall, 1972. Courant Computer Science Symposium 2, September 14-16, 1970.

[Ste66] T. B. Steel. *Formal Language Description Languages for Computer Programming.* North-Holland, 1966.

[Wex81] R. L. Wexelblat, editor. *History of Programming Languages.* Academic Press, 1981.