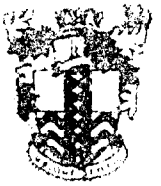

RSRE
MEMORANDUM No. 4545

ROYAL SIGNALS & RADAR ESTABLISHMENT

TEN15 DEVELOPMENTS TO SUPPORT PARALLELISM

Authors: P W Edwards, D J Tombs & D I Bruce

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
RSRE MALVERN,
WORCS.

RSRE MEMORANDUM No. 4545

CONDITIONS OF RELEASE

0116324 307617

********** DRIC U

********** DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

Defence Research Agency Electronics Division RSRE
Memorandum 4545

Title Ten15 Developments to Support Parallelism

Authors P W Edwards, D J Tombs, D I Bruce

Date: November 1991

Summary

This paper summarises the work done at RSRE as part of the COOTS collaborative project (IED3/1/1059, part funded by DTI/SERC) towards developing the Ten15 abstract machine to include facilities required for the expression of parallelism and techniques for implementation on parallel machines. The paper also indicates why this development was never taken to completion.

The content of this memorandum is reproduced from the COOTS project deliverable 2.1, June 1991.

INTENTIONALLY BLANK

Contents

INTENTIONALLY BLANK

1 Introduction

Ten15 [3] is an abstract machine founded on the use of strong typing for efficient integrity enforcement, intended to support the implementation of complete systems. The type system extends beyond the requirements of typical programming languages to support functions appropriate to operating systems. The unified type system ensures consistency both within and between separately compiled program modules, and between both static and dynamic program constructions.

As Ten15 system programming involves types that cannot be expressed correctly in any generally used language, a specially developed 'Ten15 Notation' is used. This has high level language features and provides an assembler for Ten15. A special programming development and demonstration environment is also required if the power and advantages of Ten15 are to be made easily apparent, though it is important that this environment should not be confused with Ten15 itself.

This paper describes the work started on development of the Ten15 abstract machine to include facilities for the expression of parallelism and of the techniques for its implementation on parallel machines. This work formed part of the COOTS (IED3/1/1059) collaborative project to develop and compare a variety of object-oriented languages and environments on a variety of MIMD parallel machines. Ten15 with its Notation and demonstration environment were to be developed and compared, for the support of parallel object oriented applications, with alternative systems being developed by the project partners Harlequin Ltd. and University College of London.

Continuation of the Ten15 portion of the COOTS project has been replaced however, as its completion was no longer sustainable within the project timescales - due to delays and reduced priority concerning the RSRE background Ten15 input still required for this work.

2 Background

The starting point for Ten15 development was Version 0 of Ten15 [1] with Notation [4], both implemented within RSRE's Perq Flex programming environment, together with an unimplemented extension for pseudo-parallelism, Version 0-P [2].

For the sake of portability to conventional architectures, a prototype translator from Version 0 Ten15 to VAX was written in Ten15 Notation, prior to the COOTS project, and provisionally (only partially tested) extended to Version 0-P. Development was also well under way towards creating a Ten15 demonstration environment on the VAX, essentially modelled on the Flex environment but written in Ten15 Notation. The use of Ten15 Notation throughout was to aid portability to any machine given an appropriate Ten15 translator. The VAX Ten15 environment was also intended to provide the starting point for the parallel Ten15 demonstration environment to be developed for COOTS. A major element still missing however, was the Notation compiler itself, so that all program development still required Perq Flex.

Ten15 is a total system whose implementation needs to provide all required system functions, and for which a high degree of integrity can be claimed. To use Ten15 as an implementation

technology the external interface of a compiler must use only Ten15 primitives rather than conventional operating system routines. The compiled program text can be used freely as an object within the Ten15 environment, but not outside. In particular for COOTS, parallel processes and communication between them must be compiled through new primitives, which must therefore be supplied with the demonstration environment.

A better engineered translator of Version 0 Ten15 was also underway, this time for a Transputer. This was to provide background to the COOTS transputer array implementation of Ten15 extended for full parallelism. The background single transputer translator was not complete however by the start of the COOTS project. The possibility of completing it was becoming uncertain under the tight physical resource limits imposed by Perq Flex, and it could not be transferred to the larger resources of VAX while a portable version of the Ten15 Notation compiler was still unavailable.

An early decision within the COOTS Ten15 task was to contribute effort towards a new Ten15 Notation compiler to be written in the Notation itself. This was required for COOTS for completion of Ten15 bootstrapping, to include bootstrapping the eventual parallel demonstration environment onto the transputer array, and more urgently to enable completion of the single transputer translator to be transferred to the VAX. The COOTS development of Ten15 also required the new compiler to be easily extensible, to enable easy development of concrete representations to assist the derivation of the required abstract machine development.

COOTS effort was also found necessary to provide new portable implementations of type checking and manipulation algorithms. These were required to cope with extensive polymorphism efficiently, as assumed by the new Notation compiler, and they also had to be designed for future extensions to the typing for expression of parallelism.

The additional COOTS effort expended on the new Notation compiler and type algorithms, plus reduced background resources available to complete the VAX Ten15 demonstration environment, necessarily slowed development of the abstract machine itself. The envisaged development was therefore restrained, eventually to the point that what could be achieved and implemented within the COOTS resource limits would not provide a satisfactory result.

3 The Potential for Ten15 Evolution

A wide ranging review of Ten15 was conducted at the start of the COOTS project and reported in a working paper [5], to highlight all areas of compromise and perceived deficiencies.

The review revealed most clearly that a huge amount of work still remained before Ten15 could be considered properly complete. Partly, it was functionally incomplete: several desirable features were absent, which could be incorporated into Ten15 relatively straightforwardly. A more refined definition of procedure closure is an example of such a feature.

More significantly, the semantic basis of Ten15 was very weakly defined. More research was necessary before Ten15 program could be constructed with sufficient ease and integrity. In particular, the type system displayed restrictions and machine dependencies in several places,

so impeding portability, extensibility and the use of algebraic techniques. In consequence it was difficult to produce Ten15 program and Ten15 systems, or to demonstrate their integrity. The latter is especially important when 'system' objects like memories and processors exist within Ten15, because the entire system is vulnerable to corrupted data in a single component. Much of the functionality of present Ten15 has been added on in a somewhat ad-hoc manner, with inadequate thought for the uniformity of the system as a whole.

A wide-ranging research program, generally known as Ten15 Version 2, was being undertaken to devise an adequate semantic basis. This research was leading towards a more powerful type system based on intuitionistic methods to provide a framework for Ten15 program. With such a framework better ways of constructing program, for example by homomorphic transformation, would be feasible. More powerful systems could then be built in a trustworthy manner using only a small set of kernel routines.

The anticipated developments of Ten15 were divided into three categories, according to how they affected the COOTS programme.

1. Simple extensions to Ten15 Version 0-P and other work necessary to support an implementation on parallel machines, but with lower than desired functionality and integrity. This work was considered a necessary part of the COOTS project, and comprised work on the following areas:

   Analysis of procedure non-locals for shared variables.

   Devise methods of debugging distributed Ten15.

   Out of memory exceptions.

   Develop memory models for shared and distributed memory, including garbage collection and atomic allocation and access of shared and distributed memory.

   Possibly, parallel on-the-fly garbage collection.

   Possibly, Ten15 on heterogeneous systems.

   Distribution of Ten15 values across homogeneous close-coupled networks.

   Possibly, unification of loose and close-coupled networks.

2. Work towards Ten15 Version 2, with emphasis on semantic basis and type system. This work is essential for the development of software using algebraic techniques, and would be required to demonstrate the potential of Ten15 software and so would have been useful during the later stages of the COOTS project. It was expected to be covered in background work, partly concurrent with COOTS:

   Demonstrate algebraic construction of Ten15 by programs.

   Demonstrate transformation of Ten15 to a homomorphic image.

   Representation-independent semantics for equality, integer types etc.

   Implement efficient algorithms to manipulate types.

   New type system, incorporating intuitionistic ideas.

   Methods for constructing general cyclic and polymorphic data, sharable between programs.

   Static analysis to find various properties.

   Consider functional power of command interpreter.

Write an interactive debugger.

Static subsets of Ten15.

Consider on-the-fly garbage collection, to eliminate embarrassing pauses.

Possibly, on-line datastore garbage collection.

Notation evolution to extend scope of language support.

Consider expressiveness of a new language.

3. A formally specified Ten15, implemented as a full network-wide environment, with a translator built using algebraic principles and therefore containing a very high degree of trustworthiness. The issues relating to this were postponed for longer term consideration.

   Demonstrate trustworthiness of translator.

   Write an interpreter.

   Consider granularity and breadth of scope of the type system.

   Need for multiple binding times.

   Consider level of data security.

   Transfer of untransferable values.

   Cross-store pointers.

   Persistence of unpersistable values.

   Cross-datastore pointers.

   Trustworthiness of data.

4 Minimal enhancement required for Parallelism

As indicated above, only a minimal enhancement to Ten15 could actively be considered. The pseudo-parallelism of version 0-P [2],[3] was taken as the basis for full parallelism, with consideration of both shared and distributed memory architectures.

Two constructions, Launch and Parallel, are available for the creation of new (pseudo-parallel) threads of control. Launch is used to initiate a new thread which will run in parallel with the thread that initiated it (similar to futures, or the creation of active objects), whereas Parallel is used to initiate a set of new threads to run in parallel with each other but which must all complete before the initiating thread may continue (similar to Occam PAR). The use of Parallel when initiating just a single thread is thus very similar to a procedure call. The unit used to provide the function for a new thread of control is a Task, which is a first class Ten15 value similar to a Procedure but with the addition of a communications interface. Unlike a parameter, available to a Task or Procedure alike, the communications interface is a constructional concept, not a Ten15 value.

Communication between pseudo-parallel threads may take place either implicitly through use of shared variable data, or by explicit message passing. The explicit message passing is achieved by a synchronising call/reply construct, whereby a calling thread calls for data to be transferred through a communications interface channel to a receiving thread and waits for data to be returned in reply. The receiving thread waits for the callers data and returns the

result of some operation on it. Further details and options may be found in [2] and [3]. Communication channels are created to be one-to-one between threads of control initiated by the Parallel construct. Message passing with a thread initiated by the Launch construct is achieved via special procedure values created when the new thread is launched. These procedures are first class values, able to be passed to other threads to allow many-to-one calling, the procedures incorporating mutual exclusion so that each call/reply synchronisation is locally one-to-one.

Extending the constructs above to true parallelism requires consideration of data sharing constraints. The type integrity required of Ten15 implementations requires absence of interference between reading and writing any Ten15 variable - i.e. reads and writes must be atomic. Within pseudo-parallel implementations this is achieved by cooperative scheduling between active threads of control, with rescheduling points avoiding the reads and writes.

In the case of a shared memory multiprocessor, the only simple and efficient solution to ensure atomic reads and writes, is to limit each variable such that access is only conceivable from a single thread of control. This is to avoid the need for mutual exclusion surrounding every variable access or sequence of accesses, or for the sophisticated analysis needed to reduce the inefficiency that would imply. With this limitation, the requirements of shared and distributed multiprocessors become identical, though advantage can still be made of shared memory for the implementation of message passing, and memory optimisation is possible for data known to be constant.

Data can be transferred from one thread of control to another either by explicit communication, or as parameter or non-locals bound into a task when a new thread is initiated, or as result when a thread completes. The required limitation on variable access can be achieved either by prohibiting all pointers within any data to be transferred, including any procedures with pointers within their bound non-locals, or else by including in the transfer a copy of all the data accessible through those pointers. The latter technique is less restrictive and is already adopted in Ten15 for transfer to and from persistent store, including full maintenance of cyclic data, though it does imply a semantic change as successive transfers of common data to one thread would produce separate copies of it.

The concept of a Ten15 Virtual Processor was emerging, with the idea that each thread of control belongs to a single Virtual Processor. Separate threads within a single Virtual Processor would be pseudo-parallel with unlimited data sharing, whereas full copying would be implemented for transfer of data between threads in distinct Virtual Processors. Separate Virtual Processors can run pseudo-parallel on a single physical processor or fully parallel on a multi-processor. A complete Virtual Processor would be the unit to consider for load balancing. Identification of Virtual Processor, and possibly its mapping to physical processor, would be specified at the Launch and Parallel constructs.

A number of implementation techniques required remained open for consideration, including the control of load balancing, and garbage collection of distributed memory systems. Further consideration was also required concerning object model implementation and characteristics of the proposed demonstration environment, which were likely to influence decisions on these techniques.

5 Ten15 Notation compiler

The previous Notation compiler [4] had been written in Algol68 and made use of numerous PerqFlex-specific operations. Since the PerqFlex operations are *a priori* not portable, and Algol68 was not yet available for the Ten15 system, a full re-write was necessary.

Given that the compiler had to be rewritten, it was opportune to improve a number of areas.

Some of these improvements were to the language itself:

The functionality was extended to cover the whole of Ten15. The major extension was, of course, the new constructs developed for pseudo-parallelism; though there were a few other, more minor, areas for which the functionality could not be expressed in the old Notation.

Several aspects of the syntax were unified, leading to a cleaner but also more expressive language. For example, by introducing fully general pattern matching, three constructs (Case, Deunite and most uses of When) become special cases of a new, more powerful facility.

Declarations were unified into a more general concept, which includes modules. In addition, by allowing patterns in the left-hand-side of some declarations, much of the notational convenience of functional languages was obtained at no extra effort.

The whole module system was radically revised. The previous system was unsatisfactory, in that the interfaces between modules were rather inflexible. Also, they were unacceptably limited by only allowing *values* to be kept, despite there being many other things of interest. (In particular, *types* could not be defined in modules, but had to come from a separate source, called a MoModule. This has proven clumsy in practice, especially since only one can be included in any module text. Furthermore, if we had decided to keep these MoModules, considerable extra effort would have been required to implement them for the Vax system.)

Other improvements involved the way in which the compiler was implemented:

Use of the existing Notation language was essential both to provide portability across Ten15 implementations and to access the full power of Ten15 — in particular, the ability to manipulate types directly as values was helpful.

The parser was generated from the (new) grammar with RSRE's SID tool, as before, but now using a different version that produces a polymorphic Ten15 parser instead of a specific Algol68 one.

The mechanism for output of Ten15 was made more general by polymorphically parametrising the compiler by an *encoder*. This meant that any required output can be obtained by simply 'plugging' an appropriate encoder into the compiler. Some encoders that have actually been built and used are: a disc-based byte-stream encoding that is the same as the one produced by the previous compiler; two that produce the result of applying the Ten15 translator (for Flex or Vax) directly; and a textual 'pretty-printer'. Construction of others is straightforward.

The internal structure of the compiler was radically altered, and is now in an almost-functional style that closely reflects the structure of the language.

The problems that were encountered were mostly related to resource limitations and other inadequacies of the PerqFlex development system. The disc was always almost full; sometimes the mainstore wasn't large enough; and the machine was very slow.

However, the main bottleneck concerned the implementation of the Ten15 type system on PerqFlex. Some of the algorithms were hopelessly inefficient ($\sim O(n!^2)$ time complexity),

some were just wrong, and some had never been implemented. Obviously no-one had tried to use the full power of Ten15's polymorphism before! Eventually some new routines were written that were good enough for work to continue, but then disaster struck: an as-yet unidentified fault was causing types to change spuriously underneath us. Some texts which had previously compiled no longer would. As the proposal to remove this work from COOTS had already been made, the attempt to diagnose this fault was discontinued.

When it was abandoned, the compiler was very nearly finished. A number of test fragments had been compiled using the pretty-printer to show the output, and appeared correct. A few had been translated for PerqFlex and seen to work.

Several features had not been completely implemented. Most were because of time constraints, and could easily have been finished. A couple were because they were inherently difficult, but were made usable by developing a temporary stopgap solution. The only aspect that was really preventing the compiler from being used was the actual mechanics of creating and loading modules (the form of the interface had been designed and extraction of components had been implemented).

6 Typing algorithms

As originally implemented on VAX the portable Ten15 type system provided only rudimentary support for polymorphic and cyclic types and none at all for parallel programming. The absence of the parallel extensions could be tolerated until a parallel system was required; the inadequacy of polymorphism and cycles could not, because many programs, notably the new notation compiler, made extensive use of these features.

Cyclic types had been found computationally complex to construct. The polymorphic types had a more fundamental problem: it proved impossible to infer the type resulting when applying a polymorphic function. On investigation it was decided that the best solution was to write an entire new type kernel.

Two new algorithms were designed: a lazy substitution for replacing a formal type with an actual, and a type comparison to check whether a function can legitimately be applied to an argument, and constrain its result type according to that argument. With these new algorithms polymorphic application can be performed fluently through bounded polymorphism, and the cycle complexity was resolved by adding a new multiple cycle constructor. Because the new types and algorithms demanded the construction, substitution and comparison of many more types than previously, greater reuse of already-existing types was admitted. At the same time the parallel extensions for Version 0-P were added.

The new type kernel was modelled and developed in Ten15 notation on Perq Flex, before porting to VAX. It is functionally complete and has been tested interactively using the Perq editor, but has not been debugged fully. When installed as a replacement for the Version 0 type system on VAX it is sufficiently powerful to bootstrap the VAX Version 0 system and translator, more rapidly than the original type system, but failure occurs shortly afterwards.

Apart from its general lack of robustness the system still contains a major flaw in the matching of formal parameter types, revealed by some fairly ordinary ML programs. The flaw has been

diagnosed, but as substantial alterations to the underlying data structure are required it has not yet been corrected.

The new type system represents an advance on previous type systems for Ten15 in its design, functionality and efficiency. However it still contains serious flaws and is not yet a usable product, and with the downgrading of Version 0 Ten15 may not be completed. A detailed description of the new system appears in [6]. The experience gained will be beneficial for writing type systems in future.

7 Conclusion

This paper has summarised the Ten15 work done within the COOTS project. Owing to reduced effort on the associated Ten15 background to COOTS and the large amount of work outstanding the developments described in this paper were not taken to completion.

The all-embracing total system nature of Ten15 is attractive for self-consistency enforcement and security, but hinders its exploitability. The background Ten15 activity at RSRE has been largely diverted to separating off a number of its facets for individual development independently from each other. TDF (Ten15 Distribution Format) is the foremost of these, developed from a target-independent portability layer for Ten15 translators but applicable to a much wider range of programming languages and hence more exploitable. The remaining Ten15 effort committed to COOTS has similarly been diverted within COOTS, to the development of TDF for parallel languages and parallel machines, with consequently greater exploitation potential.

The original objectives of Ten15 remain for the longer term. In preference to continued evolution, it is likely that a fresh Ten15-like system may be developed at some later stage, firmly based on TDF and other independently developed facets from Ten15, that will address the major criticisms of current Ten15.

References

1 The Ten15 Signature, Version 0
David Bruce
RSRE unpublished paper, March 1990

2 Extensions to the Ten15 Signature for Version 0-P
Margaret Stanley
RSRE unpublished paper, September 1990

3 Ten15 Prototype
J M Foster, I F Currie, N E Peeling, P W Edwards, M Stanley, P W Core, M Brandreth
RSRE Report 91025, November 1991

4 A Notation for Ten15
Kim Goodenough, Sally Rees
RSRE unpublished paper, May 1989

5 The Evolution of Ten15
David Tombs, David Bruce
RSRE COOTS Working Paper, September 1990
(subsequently revised and published as RSRE Memorandum 4543, November 1991)

6 The Version 0 Type System for Ten15
David Tombs
RSRE unpublished paper (incomplete)

REPORT DOCUMENTATION PAGE

DRIC Reference Number (if known) .....

Overall security classification of sheet .....UNCLASSIFIED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification eg (R), (C) or (S).

Originators Reference/Report No.	Month	Year
MEMO 4545	NOVEMBER	1991

Originators Name and Location
RSRE, St Andrews Road
Malvern, Worcs WR14 3PS

Monitoring Agency Name and Location

Title
TEN15 DEVELOPMENTS TO SUPPORT PARALLELISM

Report Security Classification	Title Classification (U, R, C or S)
UNCLASSIFIED	U

Foreign Language Title (in the case of translations)

Conference Details

Agency Reference	Contract Number and Period

Project Number	Other References

Authors	Pagination and Ref
EDWARDS, P W; TOMBS, D J; BRUCE, D I	11

Abstract

This paper summarises the work done at RSRE as part of the COOTS collaborative project (IED3/1/1059, part funded by DTI/SERC) towards developing the Ten15 abstract machine to include facilities required for the expression of parallelism and techniques for implementation on parallel machines. The paper also indicates why this development was never taken to completion.

The content of this memorandum is reproduced from the COOTS project deliverable 2.1, June 1991.

Abstract Classification (U,R,C or S)
U

Descriptors

Distribution Statement (Enter any limitations on the distribution of the document)

UNLIMITED