CURRIE84
I.F. Currie, "Orwellian Programming in Safety-Critical Systems"
in
System Implementation Languages: Experiences and Assessment
(sponsored by IFIP Working Group 2.4, cosponsored by IFORS) Conference Proceedings, University of Kent, September 17-19, 1984 (No continuous page numbering, articles are arranged in alphabetical order of author.)

(This article fits under the bibliography topic "Specific Languages." One should also make a cross-reference to this article under the bibliography topic "Design Criteria and Objectives".)
This article proposes a new solution to programming in safety-critical systems. I.F. Currie is not satisfied with the existing high level languages that are used in such systems, because these languages can lead to misconceptions and obscurities and hence to errors in safety-crit He proposes a new language called
NewSpeak
together with a certifying authority named
ThoughtPolice,
that watches over programs and their designers to ensure high-quality programs. The author coins this the "Orwellian solution". He is refering to George Orwell's calssical novel "Nineteen Eighty-Four".
A safety-critical system is one that must never fail because the lives of people depend on it. Two examples are fly-by-wire aircraft and computer control of the flaps of an aircraft.
The safety-critical programs in such systems are usually control programs written in current high level programming languages. To ensure program correctness, one must not only verify that the code matches external specifications of the job, but one must also make sure that the code produced by the compiler for this language matches the text. However, such a compiler does not lend itself to verification.
Further observations are that there is a trade-off between the difficulty of writing the program in a language and the complexity of implementation of the compiler for that language. Timing constraints form an important part in the specification of most of these programs. Thus the language used should be primitive recursive. Unrestricted interrupts are dangerous. Thus the safety critical program should consist of a single process.
The solution presented is "a restriction of language to limit one's capacity for dangerous thought". Such thoughts are for example, unrestructed looping and branching. Thus "computer
NewSpeak
must be a language in which such thoughts are not only forbidden, but unthinkable; the job of the
ThoughtPolicy
(in the shape of the certifying authority) then becomes much easier, not to say feasible."
The author then explains the key ideas of this language. Time taken by the program in
NewSpeak
should be bounded and space and data required for this program should be restricted. This implies a static storage allocation schema for the compilation of
NewSpeak
programs. Numbers handled in this language are bounded and arithmetic operations should never produce exceptional values. Values should be

named instead of creating variables containing them. Aliasing is illegal. All variable declarations should be initialised. In case of any violations one can get a call from the *ThoughtPolice.*

This article is well written. Currie does not give examples of *NewSpeak* programming text "so that the understanding, and hopefully, the acceptance of the philosophy is not confused by differing tastes in syntax." This article is relevant to the evolution and development of programming languages in an environment where program correctness is extremely important, as well as good solutions to these problems. The author concentrates on *ideas*, rather than on technicalities.