Pascal-F Verifier Internal Design Document

Enforcement of Language Restrictions

John Nagle

FACC / Palo Alto

1. Introduction

The validity of verification depends on the way the language is used. We find it necessary to impose various restrictions on the use of Pascal-F in order that the verification process remain valid. This document details these restrictions and the way in which they are enforced.

The primary purpose of this document is to help insure that no restriction is unenforced. The Verifier is expected to enforce all restrictions required for the verification process to be valid. Some of these restrictions are enforced in the compiler pass (CPCI #1), some are enforced during preverification checking (CPCI #2), and some are enforced in the verification condition generator, (CPCI #3).

2. Restrictions on program structure

2.1 Routine nesting

RESTRICTION	Routines may not be defined within other routines
ENFORCEMENT	This restriction has been dropped.

2.2 Non-terminating loops

RESTRICTION	The only legitimate way to write a non-terminating loop for a process is ‘‘WHILE true DO BEGIN <loop body> END; or ‘‘REPEAT <loop body> UNTIL false;’’.
ENFORCEMENT	Enforced in Jcode generation.

2.3 FOR control variable

RESTRICTION	The iteration variable of a FOR loop may not be modified from within the loop.
ENFORCEMENT	Preverification checking generates FREEZE and THAW operations as required during Jcode generation.

3. Variant Records

Variant records are presently not implemented, because of the difficulty of determining from Icode which variant is being accessed. This is not a fundamental problem and could be overcome; however, the restrictions below would then need to be enforced.

3.1 Tag field presence

RESTRICTION	Undiscriminated variants are not permitted.
ENFORCEMENT	Considered a syntax error to define a record with undiscriminated variants.

3.2 Variant access

RESTRICTION	A field in a variant record may be accessed only when the tag field of the record has the same value as the constant that labels the variant.

ENFORCEMENT A REQUIRE (EQUAL <tag field> <variant #>) will be generated during Jcode generation for each reference to a field in a variant record.

3.3 Variant erasure

RESTRICTION Whenever the tag field is changed, all the fields of all the variants selected by the tag field cease to be defined.

ENFORCEMENT A NEW <field> will be generated for each field subordinate to the changed tag. Performed during Jcode generation.

4. Exception handling

4.1 RAISE statement

RESTRICTION The Verifier will try to prove that no RAISE statement is ever executed.

ENFORCEMENT REQUIRE (false) will be generated for each RAISE statement.

5. Aliasing and side effects

RESTRICTION No distinct symbolic names are ever allowed to refer in the same context to overlapping storage.

ENFORCEMENT The only place this problem can occur in PASCAL-F is via the procedure call mechanism. The processing done in the safeexpr routine generated includes a check sufficient to prevent aliasing.

RESTRICTION No variable can be modified twice in the same statement, or modified and referenced in two different parts of the same statement.

ENFORCEMENT Once again, the only culprit is the routine call. The processing done in the safeexpr routine includes a check sufficient to prevent conflicting side effects.

6. Multiprogramming

6.1 Data sharing between processes

RESTRICTION Non-process-local data cannot be shared between code executing at different priorities under any circumstances whatsoever. This is to be enforced by the restriction that no variable other than a simple variable of type SIGNAL may be imported into or exported from a monitor.

ENFORCEMENT Enforced during compiler processing of IMPORT lists.

6.2 Records containing signals

RESTRICTION Signals may not be components of arrays or records.

ENFORCEMENT [NOT PRESENTLY ENFORCED]

6.3 Arrays of signals

RESTRICTION Arrays of signals will be supported. However, where an array of signals is referenced by WAIT operations from two processes, the subscripts must be constant, so that a static check can insure that no more than one WAIT can reference a given signal. This restriction does not apply to SEND operations.

ENFORCEMENT [NOT PRESENTLY ENFORCED]

6.4 Signals may not be passed as function or procedure arguments

RESTRICTION Signals may not be passed as function or procedure arguments.

ENFORCEMENT Enforced during compiler processing of argument lists.

6.5 Export of procedures and functions

RESTRICTION Procedures and functions may be exported from a monitor only into a lower priority environment.

ENFORCEMENT Enforced in both the compiler and Jcode generation phases.

6.6 Import of procedures and functions

RESTRICTION Procedures and functions may be imported into a monitor only into a lower priority environment.

ENFORCEMENT Enforced in both the compiler and Jcode generation phases.

6.7 Data is shared between processes only through procedures or functions

RESTRICTION Data is shared between processes only through procedures or functions exported from the higher priority monitor and called from the lower priority monitor.

This is a consequence of the above decisions.

ENFORCEMENT Implicit in rules above.

6.8 Export of signals

RESTRICTION Simple variables of type SIGNAL may be exported from a monitor only into an environment of lower priority. The additional restriction is imposed that the SEND operation is permitted only on SIGNAL variables explicitly defined in the current environment, i.e. not imported into the current environment.

ENFORCEMENT Enforced in the compiler. (to be supplied)

6.9 Import of signals

RESTRICTION Simple variables of type SIGNAL may be imported only into a monitor of lower priority than that of the monitor's environment.

ENFORCEMENT (to be supplied)

6.10 Synchronous signals

RESTRICTION Signals which are not imported or exported across a monitor boundary are referred to as ''synchronous signals''. A SEND on such a signal must cause immediate activation of a process waiting on the signal.

Synchronous signals will not be implemented in Pascal-F at this time. This is an implementation restriction.

ENFORCEMENT Treated as asynchronous signals

6.11 Asynchronous signals

RESTRICTION Signals which are imported or exported across a monitor boundary are referred to as ''asynchronous signals''. A SEND on such a signal must never cause deactivation of the sending process. (Note that the restrictions on export and import of signals imply that such a send must be a ''send down''.)

ENFORCEMENT (to be supplied)

6.12 Use of the AWAITED function is limited to synchronous signals

RESTRICTION Use of the AWAITED function is limited to synchronous signals.

Initially, the AWAITED function will not be implemented.

ENFORCEMENT Compiler considers any use of AWAITED to be an error

6.13 Signal / wait relationship

RESTRICTION Each SIGNAL variable is to be mentioned in exactly one WAIT statement.

ENFORCEMENT This restriction appears to be unnecesary.

6.14 Reachability of wait statements

RESTRICTION No WAIT statement is to be statically reachable by more than one process.

ENFORCEMENT See below.

6.15 WAIT operation in routines

RESTRICTION The initial compiler implementation will also enforce the restriction that the WAIT operation is forbidden within procedures and functions. This implicitly implies that no WAIT statement can be reached by more than one process.

ENFORCEMENT During compilation of WAIT statements

6.16 Calls out of monitors

RESTRICTION Routines imported into monitors are classified as either ‘‘atomic’’ or ‘‘non-atomic’’. Atomic routines do not perform WAIT or SEND operations, nor do they call routines that do. For non-atomic routines, each call of the routine is considered an exit from the monitor and the monitor invariant must be proven.

ENFORCEMENT Enforced by check performed after call graph has been closed.

7. PROOF and EXTRA information

7.1 Use of EXTRA variables

RESTRICTION EXTRA variables may not be mentioned in non-PROOF executable statements, except that EXTRA variables may be passed to EXTRA formal arguments to non-PROOF procedures.

ENFORCEMENT In compiler, along with usual type checking.

7.2 Assignment to non-EXTRA variables by PROOF code

RESTRICTION No PROOF statement or component of a compound PROOF statement may perform an assignment upon a non-PROOF variable.

ENFORCEMENT The compiler, when compiling statements, is either in ‘‘PROOF mode’’ or ‘‘non-PROOF mode’’. PROOF mode is on during PROOF statements and when compiling EXTRA procedures. When in PROOF mode, assignments to non-PROOF variables are forbidden.

7.3 Calls to routines from PROOF code

RESTRICTION PROOF code may not call non-EXTRA procedures. [OPEN ITEM: Levelless functions]

ENFORCEMENT When in PROOF mode, the compiler considers calls to non-EXTRA procedures to be errors.

8. Scope of names

8.1 ENTRY and EXIT name usage

RESTRICTION The only variable names permitted in ENTRY and EXIT declarations are those of formal arguments, constants, and global variables visible in all scopes to which the routine name has been exported.

ENFORCEMENT Enforced in Jcode generator.

RESTRICTION If a name appears in an ENTRY or EXIT declaration in a routine R, that name must appear in an executable statement or EXTRA statement in R routine, or in an a routine that can be (directly or indirectly) called by R.

ENFORCEMENT When routine calls are augmented by implicit parameters, every name appearing in the entry or exit condition must be an implicit or explicit parameter of the routine.

8.2 Variables in INVARIANT declarations

RESTRICTION Variables in module and monitor INVARIANT declarations must be defined in the same module or monitor header as is the invariant.

ENFORCEMENT Enforced in Jcode generator.

8.3 Calls out of modules and monitors

RESTRICTION Routines imported into monitors or modules may not make calls to routines exported from that same monitor, either directly or through intervening routines.

ENFORCEMENT Enforced by check performed after call graph has been closed.

8.4 Open items - scope

[1] What about functions in ENTRY, EXIT, and EFFECT declarations?

[2] What about functions in INVARIANT statements?

[3] What about routine INVARIANTS?

9. Specification statements

9.1 STATE statement in loops

RESTRICTION One and only one STATE statement must be written for every loop. The STATE statement must be contained in the loop body, at the top level.

ENFORCEMENT Enforced in Jcode generation.

9.2 MEASURE statement in loops

RESTRICTION Every WHILE, REPEAT, and LOOP loop must contain a MEASURE statement. The MEASURE statement must immediately follow the STATE statement associated with the loop.

ENFORCEMENT Enforced in Jcode generation.

10. Forms in expressions

10.1 Conditional expression type matching

RESTRICTION The expression after the THEN and the expression after the ELSE must have identically the same type.

ENFORCEMENT Enforced in compiler, during normal expression type checking.

10.2 OLD psuedofunction use

RESTRICTION OLD can only be used in assertions, can only be used within routines, and can only be applied to parameters of the routine and variables global to the routine.

ENFORCEMENT Enforced in the compiler.