UPC Compilers Testing Strategy v1.03 pre

Tarek El-Ghazawi, Sébastien Chauvi, Onur Filiz, Veysel Baydogan, Proshanta Saha
George Washington University

14 March 2003

Abstract

The purpose of this effort is to develop a UPC compiler validation suite. It is intended to test the functionality of any implementation of the UPC compiler and allow the user to measure the degree of its conformance to the UPC standard. The suite should contain a set of portable test programs. These programs fall under either of the following categories:

– Positive tests: These tests are to verify that UPC features work properly according to the syntax and semantics described in the UPC specifications.

– Negative tests: These tests are to determine the error detection capabilities of a UPC compiler implementation.

The section numbers in this testing strategy correspond to the section numbers in the UPC specification v1.1. Each test case verifies its corresponding statement in the UPC specification.

Programs will be categorized by the UPC functionality under testing as follows. Every test program should have a name in the following format: `section_subsection_casenum.c`. Every test program named, `section_subsection_casenum.c` , will be run through the usage of the `UpcTest.sh` shell-script. This script can be configured for the tested environment by setting the appropriate values to the configuration variables inside the script. After a test run, the following output files will be generated for every `section_subsection_casenum.c` file when applicable:

• `section_subsection_casenum.c.exec` will be the compiled UPC program ;
• `section_subsection_casenum.c.out` will contain the output generated when running the UPC program.

The following is the description of each testing program. These descriptions have been extracted from the program files.

Table of Contents

5. Environment

5.1 Conceptual Models

5.1.2 Execution Environment

Rule	3
Purpose	To check that each thread may access shared data that have affinity to any thread.
Type	Positive
How	Declare a shared variable A as integer. Thread 1 assigns an integer value (i.e. 5), than print the value of integer variable. After that thread 2 assigns new integer value for the variable (i.e. 7) and print the value again. Use barrier between the thread assignment operations. While execution of the code, check the A values for thread 1 and thread 2 as 5 and 7 respectively.

Rule	4
Purpose	To check that there is no implicit synchronization among the threads.
Type	Positive
How	Declare a shared array A[THREADS] of integers. This array will automatically be initialized to 0. At the beginning of the program, every thread reads all the elements of the array. If a thread cannot access an element of the array that has affinity to another thread, this means that this other thread has not been created yet. Therefore, the implicit barrier at the beginning of the program is not working properly.

5.1.2.1 Program Startup

Rule	1
Purpose	To check that each thread calls the UPC program's main () function.
Type	Positive
How	Compile the test program and run it for THREADS and verify the printed strings for all THREADS.

5.1.2.2. Program Termination

Rule	1
Purpose	To check that there is an implied barrier at program end
Type	Positive
How	Put barrier synchronization at the end of program and verify that the test program terminates.

Rule	1
Purpose	To check that program can be terminated by a call to the function upc_global_exit ().
Type	Positive
How	At the end of the program, synchronize all threads using upc_barrier, then on a single thread (ex. Thread 0) call upc_global_exit() and verify that the test program terminates.

Rule	2
Purpose	To check that a thread is terminated by reaching the "}" at the end of main() function.
Type	Positive
How	Run the test program and verify that all threads are terminated after the end of main function's parenthesis.

Rule	2
Purpose	To check that a thread is terminated by exit() function.
Type	Positive
How	Put exit() function in the test program and verify that all threads are terminated after the exit() function.

Rule	2
Purpose	To check that a thread is terminated by return function.
Type	Positive
How	Put return function in the test program and verify that all threads are terminated after return function.

6. Language

6.1 Notations

6.2 Predefined Identifiers

6.2.1 THREADS

Rule	1
Purpose	To check that the predefined identifier THREADS is equal to the number of threads running.
Type	Positive
How	Compile the test program with different number of threads and verify that THREADS has the correct value.

6.2.2 MYTHREAD

Rule	1
Purpose	To check that the predefined identifier MYTHREAD takes all the values in the interval [0..THREADS-1].
Type	Positive
How	Declare a shared array a[THREADS] of integers. Let each thread increase a[MYTHREAD] by 1. At the end of the program, verify that all elements of the shared array are equal to 1.

6.2.3 UPC_MAX_BLOCK_SIZE

Rule	1
Purpose	To check that the predefined identifier UPC_MAX_BLOCK_SIZE is defined and is equal to the maximum possible block size.
Type	Negative
How	Declare a shared array with block size UPC_MAX_BLOCK_SIZE+1. Verify that a compiler error is generated.

6.3 Expressions

6.3.1 The upc_localsizeof operator

Rule	Semantics 1a
Purpose	To check that the upc_localsizeof operator returns the size, in bytes, of the local portion of its operand and that value is the same for all threads.
Type	Positive
How	Declare a shared array and is sue upc_localsizeof in all threads. Verify that all threads return the correct value.

Rule	Semantics 2
Purpose	To check that the type of the result from upc_localsizeof operator is size_t.
Type	Negative
How	Assign the value of upc_localsizeof to a variable with type not compatible with size_t. Verity that the compiler reports an error.

6.3.2 The upc_blocksizeof operator

Rule	Semantics 1
Purpose	To check that the upc_blocksizeof operator returns the block size of the operand. If there is no layout qualifier, the result should be 1.
Type	Positive
How	Declare a shared array with a layout qualifier and another shared array without layout qualifier. Issue a upc_blocksizeof statement for both arrays. Verify that the block size is returned for the first array, and 1 is returned for the second array.

Rule	Semantics 2
Purpose	To check that if the operand has indefinite block size, the upc_blocksizeof operator returns 0.
Type	Positive
How	Declare a shared array with indefinite block size. Issue a upc_blocksizeof statement. Verify that it returns 0.

Rule	Semantics 3
Purpose	To check that the type of the result from upc_blocksizeof operator is size_t.
Type	Negative
How	Declare a shared array with a layout qualifier. Issue a upc_blocksizeof statement. Assign the value of upc_blocksizeof to a variable with type not compatible with size_t. Verity that the compiler reports an error.

6.3.3 The upc_elemsizeof operator

Rule	Semantics 1
Purpose	To check that the upc_elemsizeof operator returns the size, in bytes, of

	the highest-level (leftmost) type that is not an array. For non-array objects, upc_elemsizeof returns the same value as sizeof.
Type	Positive
How	Declare a shared array of arrays of a struct, that consists of integer and char fields. Issue a upc_elemsizeof statement. Verify that the returned value is the size of the struct.

Rule	Semantics 2
Purpose	To check that the type of the result from upc_elemsizeof operator is size_t.
Type	Negative
How	Declare a shared array. Issue a upc_elemsizeof statement. Assign the value of upc_elemsizeof to a variable with type not compatible with size_t. Verity that the compiler reports an error.

6.3.4 Shared Pointer Arithmetic

Rule	1a
Purpose	To check that when an expression that has integer type is added to or subtracted from a shared pointer, the result has the type of the shared pointer operand.
Type	Positive
How	Declare a shared pointer pointing to a shared array. Add/Subtract an integer type expression to/from the shared pointer. Verify that the result has the type of the shared pointer, and other threads can access this pointer.

Rule	1b
Purpose	To check that when an expression that has integer type is added to or subtracted from a shared pointer pointing to an element of a shared array object, the result points to an element of the shared array.
Type	Positive
How	Declare a shared pointer pointing to a shared array. Add/Subtract an integer type expression to/from the shared pointer. Verify that the result still points to the shared array.

Rule	2
Purpose	To check that after shared pointer arithmetic, the resulting shared pointer has the correct phase and has the affinity to the correct thread.
Type	Positive

How	Shared [B] int *p, *p1;
	Int I;
	P1 = p + I;

	The following equations must hold: Upc_phaseof(p1) == (upc_phaseof(p) + I) % B Upc_threadof(p1) == (upc_threadof(p) + (upc_phaseof(p) + I) / B) % THREADS Check a range of blocks to verify their phase relationships

Rule	3a
Purpose	To check that pointers to shared objects can be cast to pointers to local objects, provided that the shared object has affinity to the executing thread.
Type	Positive

How	The following assignments must hold:
	T *P1;
	Shared T *S1;
	
	P1 = (T*) S1;
	
	Ensure that the private pointer points to shared space which has affinity to.

Rule	4
Purpose	To check that shared pointers point to correct elements after being cast to local pointers.
Type	Positive
How	S1 and P1, and S2 and P2 must point to the same object, after following assignments: T *P1, *P2; Shared T *S1, *S2; P1 = (T*) S1; P2 = (T*) S2; The following expressions should evaluate to TRUE: ((upc_addrfield(S2) – upc_addrfield(S1)) == ((P2 - P1) * sizeof(T)) S1<S2 : upc_addrfield(S1) < upc_addrfield(S2) ? upc_addrfield(S1) > upc_addrfield(S2) Ensure that the private pointer points to shared space which has affinity to.

Rule	Testcase 4.1.1
Purpose	To check that the pointers to shared objects move in the proper manner across all thread memories.
Type	Positive
How	Declare a pointer to a shared array of integers, and two private pointer point to integer. Increment the pointer by a certain value, i. Check that the thread that has this pointer, is actually the one given by the formula: upc_threadof(ptr+i)==(upc_threadof(ptr)+i)%THREADS; Also check that the virtual address satisfies: upc_addrfield(ptr+i) = upc_addrfield(ptr)+(upc_threadof(ptr)+i)/THREADS*sizeof(*ptr). Two private shared pointer pS1,pS2 point to the two element of the array respectively, and upc_addrfield(pS1)>upc_addrfield(pS2), pS1 pS2 are point to elements of the same array and they both have affinity to the thread on which pL1 and pL2 are defined. Make the shared pointer arithmetic, pL1=pS1, pL2=pS2. Check that: upc_addrfield(pS2)-upc_field(pS1)=(pL2-pL1)*sizeof(*pL1);

Rule	Testcase 4.2.1
Purpose	To check that pointers to blocked shared objects move in the proper manner across all threads memories.
Type	Positive
How	Declare a blocked pointer to a shared array of integer. Increment the pointer by a certain value, i.Check that the thread that has this pointer, is actually the one given by this formula: upc_threadof(ptr+i)= (upc_threadof(ptr)+(upc_phaseof(ptr)+i)/BLOCK)%THREADS; Also check that the phase satisfies: upc_phaseof(ptr+i)=(upc_phaseof(ptr)+i)%BLOCK; Check that: upc_addrfield(pS2)-upc_addrfield(pS1)=(pL2-pL1)*sizeof(*pL1);

6.3.5 Cast and Assignment Expressions

Rule	Constraints 1
Purpose	To check that non-shared type qualifier objects cannot be cast to shared qualifier objects.
Type	Positive
How	Declare a non-shared object. Verify that it can not be cast to shared type.

Rule	Semantics 1
Purpose	To check that casts or assignments from one shared pointer to another, in which either the type size or the block size differs, results in a pointer with a zero phase, unless one of the types is “shared void*”.
Type	Positive

How	Declare two shared arrays with same type size but different block size. Declare two shared pointers that point to elements of these shared arrays. Ensuring that the source pointer has a non-zero phase, assign one shared pointer to another using a cast. Verify that upc_phaseof(pointer) is equal to zero. Declare two shared arrays with same block size but different type size. Declare two shared pointers that point to elements of these shared arrays. Cast one shared pointer to another. Verify that upc_phaseof(pointer) is equal to zero.

Rule	Testcase 5.1.1
Purpose	To check the assignment from shared to private objects.
Type	Positive
How	Declare a shared integer and a private integer variable. Thread 0 assigns initial predefined value to the shared variable. All the threads assign this value to their local integer variable and check that they see the same value.

Rule	Testcase 5.2.1
Purpose	To check assignment of private data to shared data.
Type	Positive
How	Define a shared integer and a local integer variable. Each thread assigns MYTHREAD to the local integer. Sequentially, each thread assign its local integer to the shared integer and the other threads check the value of the share integer.

Rule	Testcase 5.3.1
Purpose	To check the assignment of private pointers to shared.
Type	Negative
How	Define a pointer to shared and a private pointer to private data. Assign the private pointer to the shared pointer. Verify the compiler reports an error.

Rule	Testcase 5.3.2
Purpose	To check the assignment of private pointers to blocked shared pointers.
Type	Negative
How	Define a pointer to blocked-shared and a private pointer to private data. Make the private pointer point to the private data. All threads assign the private pointer to the shared blocked pointer and there will be a

	compilation error message.

Rule	Testcase 5.4.1
Purpose	To check the assignment of shared pointers to private ones.
Type	Positive
How	Define a shared integer data, a shared pointer and a private pointer to integer. Thread 0 assign the shared data a defined value. Place a barrier. All threads assign the shared pointer to their private pointers. The thread 0 should see the same value through their private pointers.

Rule	Testcase 5.4.2
Purpose	To check the assignment of blocked shared pointers to private pointers.
Type	Positive
How	Define a blocked shared pointer and a blocked shared array with the same blocking factor. Define a private pointer. All threads assign the blocked shared pointer to an element of the blocked array a. Assign the blocked shared pointer to private pointer.

Rule	Testcase 5.5.1
Purpose	To check that the phase becomes 0 and that pointer arithmetic obeys the original type of the shared pointer to be manipulated.
Type	Positive
How	Declare two blocked shared pointers to integer, with different blocking factors. Cast one to the other. Check that the phase is equal to 0, for the assigned pointer. Increment the pointer by a certain value, through the use of upc_addrfield().

6.4 Declarations

Rule	Constraints 1
Purpose	To check that the declaration specifiers in declarations cannot include, either directly or indirectly, both “strict” and “relaxed”.
Type	Negative
How	Declare a shared type that has strict access. Declare a shared type using the first type that has relaxed access. Declare a shared variable using the second type. Verify that the compiler reports an error.

Rule	Constraints 2
Purpose	To check that the declaration specifiers in declarations cannot include, either directly or indirectly, more than one block size.
Type	Negative
How	Declare a shared array type using a definite block size. Declare another shared array type using the first type, using a different block size. Declare an array using the second type. Verify that the compiler reports

	an error.

6.4.1 Type Qualifiers

6.4.2 The Shared and Reference Type Qualifiers

Rule	Constraints 1
Purpose	To check that a reference type qualifier can appear in a qualifier list only when the list also contains a shared type qualifier.
Type	Negative
How	Declare a non-shared type that has strict/relaxed access. Declare a private variable of this type. Verify that the compiler reports an error.

Rule	Constraints 2a
Purpose	To check that a shared type qualifier cannot appear in the specifier qualifier list of a structure declaration unless it qualifies a pointer type.
Type	Negative
How	Declare a structure type with the “shared” keyword and a shared variable of this type. Verify that the compiler reports an error.

Rule	Constraints 2b
Purpose	To check that a shared type qualifier cannot appear in the specifier qualifier list of a structure declaration unless it qualifies a pointer type.
Type	Positive
How	Declare a structure type with the “shared” keyword and a shared pointer variable of this type. Verify that the program compiles successfully.

Rule	Constraints 3
Purpose	To check that the layout qualifier “*” cannot appear in the declaration specifiers of a pointer.
Type	Negative
How	Declare a pointer with “*” layout qualifier. Verify that the compiler reports an error.

Rule	Semantics 1
Purpose	To check that any thread can reference a shared object.
Type	Positive
How	Declare a shared variable and let every thread read its value.

Rule	Semantics 2
Purpose	To check that access to “strict” shared objects are according to strict access guidelines.

Type	Positive
How	Include "upc_strict.h". In a for loop, thread 0 will be assigning the index of the loop to a first variable, x, and the index+1 to another variable, y, in that order. Other threads will be computing y-x. If, in any iteration, y-x>1, then the strict consistency model in not working properly. An error message should be issued.

Rule	Semantics 4a
Purpose	To check that the block size dictates the number of consecutive elements have affinity to the same thread. If the block size is zero, or no block size is given, all objects must have affinity to the same thread. If there is no layout qualifier, the block size should default to 1.
Type	Positive
How	Declare a shared array with block size n. Verify that n consecutive elements in the array have affinity to the same thread.

Rule	Semantics 4b
Purpose	To check that the block size dictates the number of consecutive elements have affinity to the same thread. If the block size is zero, or no block size is given, all objects must have affinity to the same thread. If there is no layout qualifier, the block size should default to 1.
Type	Positive
How	Declare a shared array with block size zero (or no block size specified). Verify that all elements in the array have affinity to the same thread.

Rule	Semantics 6
Purpose	To check that the block size is verified as part of the type compatibility.
Type	Negative
How	Declare two shared arrays with different block sizes. Declare two pointers to these arrays. Assign one pointer to the other without casting. Verify that the compiler reports an error.

Rule	Semantics 7
Purpose	To check that "shared void *" pointers are type compatible with any shared pointer type.
Type	Positive
How	Declare two shared arrays with different types and block size. Declare two pointers to these arrays. One pointer's type should be "shared void*". Verify that assignment of one pointer to another generates no errors.

Rule	Semantics 8
Purpose	To check that shared objects with layout qualifier of "[*]" are distributed to threads correctly.
Type	Positive

How	For a shared object a with layout qualifier [*], the object must be distributed as if it had a block size of (sizeof(a) / upc_elemsizeof(a) + THREADS – 1) / THREADS

Rule	Semantics 10a
Purpose	To check that shared scalars cannot be declared with automatic storage class.
Type	Negative
How	Declare a shared scalar in the context of a function. Verify that the comp iler reports an error.

Rule	Semantics 10b
Purpose	To check that shared pointers can be declared with automatic storage class.
Type	Positive
How	Declare a shared pointer in the context of a function. Verify that the compiler reports an error.

Rule	Semantics 10c
Purpose	To check that pointers that are shared themselves cannot be declared with automatic storage class.
Type	Negative
How	Declare a pointer with shared keyword in a function. Verify that the compiler reports an error.

Rule	Semantics 11a
Purpose	To check that shared scalars cannot be declared inside structures.
Type	Negative
How	Declare a shared scalar in the context of a structure. Verify that the compiler reports an error.

Rule	Semantics 11a2
Purpose	To check that shared scalars cannot be declared inside unions.
Type	Negative
How	Declare a shared scalar in the context of a union. Verify that the compiler reports an error.

Rule	Semantics 11b
Purpose	To check that shared pointers can be declared inside structures.
Type	Posit ive
How	Declare a shared pointer in the context of a structure. Verify that the program compiles successfully.

Rule	Testcase 3.1.1

Purpose	To check that the qualifying an object as shared shared through the use of typedef is permitted.
Type	Positive
How	Define a new type shared_integer as being shared integer. Declare a shared variable of type shared_integer. Test the affinity to thread 0. Verify that a specific value written into the variable by thread 0 can be seen by all threads.

6.4.3 Declarators

Rule	Constraints 1
Purpose	To check that the declaration specifiers in declarations cannot include, either directly or indirectly, both “strict” and “relaxed”.
Type	Negative
How	Declare a typedef with the strict qualifier. Declare a variable of this type with the relaxed qualifier. Verify that the compiler reports an error.

Rule	Constraints 2
Purpose	To check that the declaration specifiers in declarations cannot include, either directly or indirectly, more than one block size.
Type	Negative
How	Declare a typedef with a block size. Declare a variable of this type with another block size. Verify that the compiler reports an error.

Rule	Constraints 3
Purpose	To check that “shared” can not appear in a declarator which has automatic storage duration, unless it qualifies a pointer type.
Type	Negative
How	Declare a shared scalar in the context of a function. Verify that the compiler reports an error.

Rule	Semantics 1
Purpose	To check that all static non-array shared-qualified objects have affinity with thread zero.
Type	Positive
How	Declare a static shared scalar and verify its affinity is with thread zero.

Rule	Testcase 9.1.1

Purpose	To check that all shared objects are automatically initialized to 0.
Type	Positive
How	Declare a shared array of integer at the beginning of main(), check that all array values are equal to 0. Otherwise, an error message should be generated.

Rule	Testcase 1.1.2
Purpose	To check that function parameters can not be shared scalars.
Type	Negative
How	Call a function with shared scalars as parameters. Verify that the compiler reports an error.

Rule	Testcase 1.3.1
Purpose	To check that shared structures have affinity to thread 0 and the structure and its members can be accessed by all threads.
Type	Positive
How	Declare a shared structure made of two or more fields. Thread 0 initializes members to some predefined value. Verify that all threads can read that predefined value. Verify that the struct variable and its members have affinity to thread 0.

Rule	Testcase 1.4.1
Purpose	To check that shared unions have affinity to thread 0 and the union and its members can be accessed by all threads.
Type	Positive
How	Declare a shared union made of two or more fields. Thread 0 initializes members to some predefined value. Verify that all threads can read that predefined value. Verify that the union variable and its members have affinity to thread 0.

6.4.3.1 Pointer Declarators

Rule	Semantics 2
Purpose	To check that shared objects with affinity to a given thread can be accessed by either shared pointers or private pointers of that thread.
Type	Positive
How	Declare a shared array and verify its elements can be accessed using a shared pointer by each thread, and using private pointers of the thread with which they have affinity.

Rule	Testcase 2.1.2

Purpose	To check that shared pointers can not point to private scalars.
Type	Negative
How	Declare a shared pointer and a non-shared scalar. Make the pointer point to the private scalar. Verify that the compiler reports an error.

Rule	Testcase 2.2.1
Purpose	To check that a shared pointer to shared array declaration will result in a pointer variable that has affinity to thread 0. This pointer should be pointing to a shared array. Check that this pointer is accessible to all threads.
Type	Positive
How	Declare a shared pointer. Assigns to the shared pointer the vase address of the shared array in every thread. Thread 0 initializes the array to some predefined value. Place a barrier. All the threads should see the same values of the pointer and the array elements.
Rule	Testcase 2.2.2
Purpose	To check that shared pointers can not point to non-shared array.
Type	Negative
How	Declare a non-shared array, and a shared pointer. In thread 0, make the shared pointer point to the array. Verify that the compiler reports an error.

Rule	Testcase 2.3.1
Purpose	To check that a private to shared pointer declaration will result in a pointer to a structure variable. Both the pointer and the variable should have affinity to thread 0. This pointer can point to the shared structure variable.
Type	Positive
How	Declare a private pointer to a shared structure. Assign to the shared pointer the address of the structure variable by all threads. Thread 0 initializes its fields to some predefined values. Place a barrier. All threads should see the same values of the pointer, through upc_addrfield(ptr), and the same value for the struct variable members.

Rule	Testcase 2.3.2
Purpose	To check that shared pointers can not point to non-shared structures.
Type	Negative
How	Declare a shared pointer that points to a non-shared structure. Assign to the shared pointer the address of the non-shared structure variable. Thread 0 initializes the fields to a predefined value.

6.4.3.2 Array Declarators

Rule	Constraints 1a

Purpose	To check that for shared-qualified but not indefinite layout qualified arrays translated in dynamic THREADS environment, the THREADS lvalue occurs exactly once in one dimension of the array declarator, including typedefs.
Type	Negative
How	Declare type of a shared definite-layout array with THREADS identifier as dimension. Declare a shared array of this type with THREADS identifier as dimension. Verify that the compiler reports an error.

Rule	Constraints 1b
Purpose	To check that for shared-qualified but not indefinite layout qualified arrays translated in dynamic THREADS environment, the THREADS lvalue occurs only alone or multiplied by a constant expression.
Type	Negative
How	Declare a shared definite-layout array with dimension (THREADS + expression). Verify that the compiler reports an error.
Rule	Semantics 1a
Purpose	To check that shared array elements with no block size are distributed in round-robin fashion, by one element.
Type	Positive
How	Declare a shared array with no block size and verify that the I-th element has affinity with thread (I % THREADS).

Rule	Semantics 1b
Purpose	To check that shared array elements are distributed in round-robin fashion, by chunks of block-size elements.
Type	Positive
How	Declare a shared array and verify that the I-th element has affinity with thread (floor(I / blocksize) % THREADS).

Rule	Semantics 2
Purpose	To check that in an array declaration, the type qualifier applies to the elements.
Type	Positive

How	Typedef int S[10];
	Shared [3] S T[3*THREADS];
	The 2 dimensional array T should be blocked as if it were declared:
	Shared [3] int T[3*THREADS][10];

Rule	Testcase 1.5.1
Purpose	To check that the elements of the shared array of a structure can be accessed by all threads and the affinity of the members of structure in the shared array.
Type	Positive
How	Declare a shared array of structure. Assign the initial value to the

	elements of the array by each thread, with a forall statement. All threads check that they can see the same values. Otherwise, an error message should be returned.Check the affinity of the elements of the array using upc_threadof*(. If its affinity is not correct, an error message should be returned.

Rule	Testcase 1.5.2
Purpose	To check that the elements of the blocked shared array of a structure can be accessed by all threads and the affinity of the members of structure in the shared array.
Type	Positive
How	Declare a blocked shared array of structure. Assign the initial value to the elements of the array by each thread, with a forall statement. All threads check that they can see the same values. Otherwise, an error message should be returned. Check the affinity of the elements of the array using upc_threadof*(. If its affinity is not correct, an error message should be returned.

6.5 Statements and Blocks

6.5.1 Barrier Statements

Rule	Constraints 1
Purpose	To check that the expressions used as operands to upc_notify, upc_wait, upc_barrier and upc_fence must be integer expressions.
Type	Negative
How	Call upc_notify, upc_wait, upc_barrier and upc_fence with non-integer expressions. Verify that the program faces a runtime error.

Rule	Constraints 2
Purpose	To check that upc_notify and upc_wait statements should be called in alternating sequence, starting with a upc_notify and ending with a upc_wait statement.
Type	Negative
How	Call upc_notify n-times in a row and call upc_wait n-times in a row. Verify that the compiler reports an error.

Rule	Semantics 1
Purpose	To check that a upc_wait statement does not complete until all threads have completed the corresponding upc_notify statements.
Type	Positive
How	Declare a shared array a[THREADS] of integers. This array will be automatically initialized to 0. Just before reaching the notify statement, each thread will set a[MYTHREAD] to 1. Right after the wait statement, verify that all elements of the array is equal to 1. Otherwise,

	generate an error.

Rule	Semantics 2
Purpose	To check that accesses to all shared references are completed before the thread exits the upc_fence statement.
Type	Positive
How	Declare a shared array a[THREADS] of integers. This array will be automatically initialized to 0. Just before reaching the notify statement, each thread will set a[MYTHREAD] to 1. Right after the upc_fence statement, verify that all elements of the array is equal to 1. Otherwise, generate an error.

Rule	Semantics 4
Purpose	To check that accesses to all shared references are completed before a upc_notify statement and after a upc_wait statement.
Type	Positive
How	Declare a shared array a[THREADS] of integers. This array will be automatically initialized to 0. Just before reaching the notify statement, each thread will set a[MYTHREAD] to 1. Right after the upc_wait statement, verify that all elements of the array is equal to 1. Otherwise, generate an error.

Rule	Semantics 5
Purpose	To check that a runtime error is generated if the value of the expression of the upc_wait statement is not equal to the value of the expression of the corresponding upc_notify statement.
Type	Positive
How	Call upc_wait and upc_notify statements with different expression values and verify a runtime error is generated.

Rule	Semantics 6
Purpose	To check that a runtime error is generated if the value of the expression of the upc_wait statement is not equal to expressions of any upc_wait and upc_notify statements issued by any thread in the current synchronization phase.
Type	Positive
How	Call upc_wait with an out of range value (i.e. not corresponding to any upc_notifies such as -1). Verify that this results in a runtime error.

Rule	Semantics 7
Purpose	To check that a upc_barrier statement is equal to a {upc_notify; upc_wait} pair, that is, no thread proceeds after the barrier until all the

	other threads have reached that statement as well.
Type	Positive
How	Declare a shared array a[THREADS] of integers. This array will be automatically initialized to 0. Just before reaching the barrier, each thread will set a [MYTHRREAD] to 1. Just after the barrie r, every thread computes the sum of the entries in the array. Verify that the sum is equal to THREADS. Otherwise, generate an error.

Rule	Semantics 8
Purpose	To check that a runtime error is generated if a upc_barrier statement is issued between a upc_notify - upc_wait pair.
Type	Positive
How	Call {upc_notify; upc_barrier; upc_wait;} in this order and verify a runtime error is generated.

Rule	Testcase 7.4.1
Purpose	To check that shared work can be done between upc_notify and upc_wait.
Type	Positive
How	Declare a shared array a[THREADS] of integers. Right after the upc_notify statement, every thread sets a[MYTHREAD] to 1. After upc_wait, every thread computes the sum of the elements of the array. This will not test the accuracy of the computations but rather the feasibility of computation between upc_notify and upc_wait.

6.5.2 Iteration Statements

Rule	Constraints 1
Purpose	To check that the expression for affinity in upc_forall statement must be a pointer to a shared object or an integer expression.
Type	Negative
How	Issue a upc_forall statement with a private pointer and non-integer scalar value. Verify that the compiler reports an error.

Rule	Semantics 2
Purpose	To check that when affinity field in a upc_forall statement is a reference to shared memory space, the loop body is executed for each iteration in which the value of MYTHREAD equals the value of the affinity field.
Type	Positive
How	Issue a upc_forall statement that has a reference to shared memory space in the affinity field. In the loop body, verify that the expression

	(upc_threadof(affinity)=MYTHREAD) is true for each executed iteration.

Rule	Semantics 3
Purpose	To check that when affinity field in a upc_forall statement is an integer expression, the loop body is executed for each iteration in which the value of MYTHREAD equals the value pmod(affinity, THREADS), where pmod(a,b) is evaluated as (a>=0) ? (a%b) : (((a%b) + b) % b).
Type	Positive
How	Issue a upc_forall statement that has an integer expression in the affinity field. In the loop body, verify that the expression (pmod(affinity, THREADS)=MYTHREAD) is true for each executed iteration.

Rule	Testcase 8.1.2
Purpose	To check that when affinity field in a upc_forall statement is a constant, the loop body is executed for each iteration in which the value of MYTHREAD equals that constant.
Type	Positive
How	Issue a upc_forall statement that has an integer expression in the affinity field. In the loop body, verify that the expression (affinity=MYTHREAD) is true for each executed iteration.

Rule	Semantics 4
Purpose	To check that when affinity field in a upc_forall statement is "continue", the loop body of the upc_forall statement is executed for every iteration on every thread.
Type	Positive
How	Declare a two dimensional array of integers, a[N][THREADS]. Write a upc_forall statement that has "continue" as the affinity field, with N total iterations. This array will be automatically initialized to 0. In the forall statement, every thread sets a[I][MYTHREAD] to 1, where I is the current iteration. At the end of the forall loop, check that every entry in this two dimensional array is equal to 1.

Rule	Semantics 5
Purpose	To check that when no affinity field is specified in a upc_forall statement, the loop body of the upc_forall statement is executed for every iteration on every thread.
Type	Positive
How	Declare a two dimensional array of integers, a[N][THREADS]. Write a upc_forall statement that has no affinity field, with N total iterations. This array will be automatically initialized to 0. In the forall statement, every thread sets a[I][MYTHREAD] to 1, where I is the current iteration. At the end of the forall loop, check that every entry in this two dimensional array is equal to 1.

Rule	Semantics 6
Purpose	To check that in a nested upc_forall statement, the upc_forall statements which are not “controlling” behave as if their affinity expressions were “continue”.
Type	Positive

How	main () {
	    int i,j,k;
	    shared float *a, *b, *c;
	    upc_forall(i=0; i<N; i++; continue)
	        upc_forall(j=0; j<N; j++; &a[j])
	            upc_forall (k=0; k<N; k++; &b[k])
	                a[j] = b[k] * c[i];
	}
	Verify that this example executes all iterations of the “i” and “k” loops on every thread, and executes iterations of the “j” loop on those threads where upc threadof (&a[j]) equals the value of MYTHREAD.

Rule	Testcase 8.2.1
Purpose	To check that a thread can skip its current iteration.
Type	Positive
How	In the body of the loop, put a condition that, when satisfied, the thread should skip that iteration. At the end of the loop, check if the iterations have been actually skipped.

6.6 Preprocessing Directives

6.6.1 UPC Pragmas

7. Library

7.2 UPC Utilities

7.2.1 Termination of All Threads

Rule	Description 1
Purpose	To check that upc_global_exit() flushes all I/O and terminates execution for all active threads.
Type	Positive
How	After issuing I/O operation, call upc_global_exit(). Verify that the operation is complete and the program exists successfully.

7.2.2 Shared Memory Allocation Functions

7.2.2.1 The upc_global_alloc function

Rule	Description 1
Purpose	To check that upc_global_alloc(size_t nblocks, size_t nbytes) function allocates a contiguous shared memory space of (nblocks*nbytes) size distributed by (nbytes) bytes to each thread.
Type	Positive
How	Allocate memory with upc_global_alloc function and check the affinity of each element in the array. Verify that all elements in the array can be read by all threads.

Rule	Description 2
Purpose	To check that if called by multiple threads, all threads which make the call get different locations.
Type	Positive
How	Allocate memory with upc_global_alloc function in every thread and verify that the function returns different pointers for every thread.

7.2.2.2 The upc_all_alloc function

Rule	Description 2
Purpose	To check that upc_all_alloc(size_t nblocks, size_t nbytes) function allocates a contiguous shared memory space of (nblocks*nbytes) size distributed by (nbytes) bytes to each thread.
Type	Positive
How	Allocate memory with upc_all_alloc function and check the affinity of each element in the array. Verify that all elements in the array can be read by all threads.

Rule	Description 3
Purpose	To check that the upc_all_alloc function returns the same pointer value on all threads.
Type	Positive
How	Allocate memory with upc_all_alloc function in every thread and verify that the function returns the same pointers for every thread.

Rule	Description 4
Purpose	To check that the dynamic lifetime of allocated object extends from the time any thread completes the call to upc_all_alloc function until all threads have deallocated the object.
Type	Positive
How	

7.2.2.3 The upc_local_alloc function

Rule	Description 1
Purpose	To check that the upc_local_alloc(size_t nblocks, size_t nbytes) function returns a pointer to a (nblocks*nbytes) bytes of shared memory with affinity to the calling thread.
Type	Positive
How	Allocate memory with upc_local_alloc function and check the affinity of each element in the array. Verify that all elements in the array can be read by the executing thread.

Rule	Description 3
Purpose	To check that each thread calling upc_local_alloc gets a different pointer.
Type	Positive
How	Call upc_local_alloc() to allocate memory in all threads. Verify that all returned pointers point to different memory blocks.

7.2.3 Pointer-to-shared manipulation functions

7.2.3.1 The upc_threadof function

Rule	Description 1
Purpose	To check that the upc_threadof function returns the number of the thread that has affinity to the shared object pointed to by the operand.
Type	Positive
How	Declare a shared array, ensure that return value of upc_threadof is as expected, that is to say that upc_threadof(array[i]) should return the expected thread number.

7.2.3.2 The upc_phaseof function

Rule	Description 1
Purpose	To check that the upc_phaseof function returns the phase field of the shared pointer operand.
Type	Positive
How	Declare a shared array, ensure that return value of upc_phaseof is as expected, that is to say that upc_phaseof(array[i]) should return the expected phase value.

7.2.3.4 The upc_addrfield function

Rule	Description 1
Purpose	To check that the upc_addrfield function returns an implementation-defined value reflecting the "local address" of the object pointed to by the shared pointer operand.
Type	Positive
How	Declare a shared array, ensure that return value of upc_addrfield(array[i]) returns the local address of array[i].

7.2.4 Locks

7.2.4.2 The upc_global_lock_alloc function

Rule	Description 1
Purpose	To check that the upc_global_lock_alloc function dynamically allocates a lock and returns a pointer to it.
Type	Positive
How	Allocate a lock with upc_global_lock_alloc and verify that it returns a pointer of type *upc_lock_t.

Rule	Description 2
Purpose	To check that the upc_global_lock_alloc function is not a collective function, and that calls by multiple threads returns different allocations.
Type	Positive
How	Let multiple threads allocate a lock with upc_global_lock_alloc and verify that the pointer returned to each thread is different.

7.2.4.3 The upc_all_lock_alloc function

Rule	Description 1

Purpose	To check that the upc_all_lock_alloc function dynamically allocates a lock and returns a pointer to it.
Type	Positive
How	Allocate a lock with upc_all_lock_alloc and verify that it returns a pointer of type *upc_lock_t.

Rule	Description 2
Purpose	To check that the upc_all_lock_alloc function is a collective call that dynamically allocates a lock and returns the same pointer to all threads.
Type	Positive
How	Allocate a lock with upc_all_lock_alloc and verify that it returns a pointer of type *upc_lock_t and all threads get that same pointer value.

7.2.4.5 The upc_lock function

Rule	Description 1
Purpose	To check that the upc_lock function locks the lock pointed to by its operand of type (*upc_lock_t).
Type	Positive
How	Declare a lock and lock it using upc_lock(). Issue upc_lock_attempt() in a different thread. Verify that the function returns a 0.

Rule	Description 2
Purpose	To check that if the lock is owned by another thread, the call to upc_lock blocks the executing thread and continues only after the owner thread unlocks the lock.
Type	Positive
How	Declare a lock and lock it using upc_lock(). Issue upc_lock() in a different thread. Verify that the call returns only after issuing upc_unlock() in the first thread.

7.2.4.6 The upc_lock_attempt function

Rule	Description 1
Purpose	To check that the upc_lock_attempt function takes a pointer of type (*upc_lock_t) as operand.
Type	Positive
How	Declare a pointer of type (*upc_lock_t). Issue a upc_lock_attempt function call. Verify that the program compiles successfully.
Rule	Description 2a
Purpose	To check that when any other thread does not use the lock, the upc_lock_attempt function gets the lock and returns 1.
Type	Positive
How	Declare a lock and lock it using one of the threads. Unlock the lock

	using upc_unlock(). Issue upc_lock_attempt() in another thread. Verify that the function returns a 1. Issue upc_lock_attempt() in a different thread. Verify that the function returns a 0.

Rule	Description 2b
Purpose	To check that when the lock is used by any other thread, the upc_lock_attempt function returns 0.
Type	Positive
How	Declare a lock and lock it using one of the threads. Issue upc_lock_attempt() in a different thread. Verify that the function returns a 0.

7.2.4.7 The upc_unlock function

Rule	Description 1
Purpose	To check that the upc_unlock function frees the lock pointed to by the argument of type (*upc_lock_t).
Type	Positive
How	Allocate and then free a lock using upc_unlock(). Verify that the lock variable is freed.

7.2.5 Shared String Handling Functions

7.2.5.1 The upc_memcpy function

Rule	Description 1
Purpose	To check that the upc_memcpy function copies a block of memory from one shared memory area to another shared memory area.
Type	Positive
How	Initialize a shared array. Copy the shared array to another shared array using the upc_memcpy function. Verify that the two arrays are identical.

7.2.5.2 The upc_memget function

Rule	Description 1
Purpose	To check that the upc_memget function copies a block of memory from a shared memory area to a private memory area on the calling thread.
Type	Positive
How	Initialize a shared array. Copy the shared array to a private array using the upc_memget function. Verify that the two arrays are identical.

7.2.5.3 The upc_memput function

Rule	Description 1
Purpose	To check that the upc_memput function copies a block of memory from the calling thread's private memory area to a shared memory area.

Type	Positive
How	For each thread, initialize a private array. Copy the private array to a shared array using the upc_memput function. Verify that the two arrays are identical.

7.2.5.4 The upc_memset function

Rule	Description 1
Purpose	To check that the upc_memset function copies a given value converted to an unsigned char, to a shared memory area.
Type	Positive
How	Initialize a shared array using the upc_memset function. Verify that it has been initialized correctly.

Appendix

1. Composite Feature Testing

1.1 Shared Memory Allocation and Iterations

Rule	1
Purpose	To check that memory allocation and iteration statements work together correctly.
Type	Positive
How	Allocate shared memory with a upc_all_alloc() call with block size greater than one and number of blocks greater than the number of threads. Iterate through the elements of the array with a upc_forall statement and check the affinity of each element with upc_threadof() function. Verify that each thread only iterates through elements that have affinity to it.

1.2 Shared Pointer Arithmetic and Block Sizes

Rule	1
Purpose	To check that shared pointer arithmetic works correctly on dynamically allocated shared memory with different block sizes.
Type	Positive
How	Allocate shared memory with a upc_all_alloc() call with block size greater than one and number of blocks greater than the number of threads. Iterate through the elements of the array with a while loop, using private shared pointers as loop index. Calculate the next available iteration for the current thread with pointer arithmetic. Verify that each thread only iterates through elements that have affinity to it.

1.3 Barriers and Iterations

Rule	1
Purpose	To check that barriers work correctly when used within iterations.
Type	Positive
How	Iterate through a shared memory space with upc_forall. Place upc_notify() and upc_wait() statements in the loop body, such that each thread will wait and synchronize with other threads after finishing each iteration. Verify that barriers work correctly. *Warning: It is not guaranteed that all threads will have the same number of iterations .*

1.4 Shared String Handling and Barriers

Rule	1
Purpose	To check that shared string hand ling functions work correctly when some of the threads are blocked.
Type	Positive
How	Issue upc_barrier() statements to block some of the threads. In one of the unblocked threads, call upc_memget() to read shared memory. Verify that memory with affinity to blocked threads are copied correctly to the private memory.