1 APPENDIX

1.1 Applicability Study (Long Version)

To find the Python examples, which were selected with the aim of directly applying our tool, we searched GitHub using the string "topic:python3" and examined the 20 most "starred" results. Due to our tool's restricted support for data-types, we manually narrowed the search to projects whose behavior can be faithfully represented by integers, filtering out projects that rely heavily on floating-point numbers, string manipulations, or overly complex data structures and interfaces (e.g. databases). Two authors then ordered the projects based on quality of documentation. We examined the 100 most recent commits from the first project, *Delorean*, and left the remaining projects for future work.

Out of these 100 commits, only 24 deal with Python source code. Nine Of these 24 commits contain a semantic change to a function that does not modify its signature. Inside of these nine commits we found six function updates that can be entirely understood through integer reasoning. Two of the function updates applied to the same function, *Delorean()*. See Sec. 6.4 for detail on behavior preserving transformations to integers that had to be performed.

To find clients for these six functions we searched GitHub using the string "Delorean". This search yielded 1,307 Python code results. Filtering out projects with fewer than five "stars", we found five unique clients in the first 200 results. Among these, the functions *Speculator.date_to_delorean()*, *Speculator.get_end_start_epochs()*, and *bii-server.UtcDatetime()*, all calling *Delorean()*, were identified for analysis.

In all six cases – three client functions considered with two library updates each – the change in the library is not exercised. That is, for each client-library pair, the update to the library is client-specific equivalent. [5]

***Delorean.__init__* (Delorean).** The *Delorean.__init__* library function receives two parameters: *datetype* and *timezone* and returns a Delorean object that provides users with the datetime manipulation functionality. The behavior of the constructor is entirely defined by the type of arguments it receives. That is, all conditionals branch based on the type of *datetime* and *timezone*. There are two changes of interest for this library (`#679596a`, `#064bc8d`). Both changes, which concern the setting of an instance variable *_tzinfo* (timezone info), occur inside a check for *timezone* and *datetime* being *None*; the latter change occurs inside an additional check for *timezone* being of type *tzinfo*. Both these changes are irrelevant to clients that either

(1) construct a Delorean object with a *None datetime*, or *timezone*; or
(2) construct a Delorean object with a string *timezone*; or
(3) manually reset *_tzinfo* after creating the Delorean object.

To find clients for these two updates, we searched GitHub using the string "Delorean". This search yielded 1307 Python code results. Filtering out projects with fewer than five "stars", we found three unique client functions, *Speculator.date_to_delorean()*, *Speculator.get_end_start_epochs()*, and *bii-server.UtcDatetime()*. All six cases – three client functions considered with two library updates each – are unaffected by the library updates because they call the Delorean constructor with values of *datetime*, or *timezone* that avoid the change, as discussed above.

***BN_is_prime_fasttest_ex* (OpenSSL).** Library *BN_is_prime_fasttest_ex* receives four parameters: an integer *a*, an integer flag *do_trial_division*, and two structs used for call back procedure and context that are irrelevant to the change. The function aims to return 1 if *a* is prime, and 0 otherwise. *do_trial_division* specifies whether the function should attempt to divide *a* by a constant list of small primes. The change of interest (`#6e64c560`) fixes a bug in which the original function considered small primes as composites because they are evenly divisible by a prime (themselves). After the commit, aptly titled "Small primes are primes too", the function checks that a candidate composite is not in the list of small primes. This change is irrelevant to clients that:

(1) call *BN_is_prime_fastest_ex* with *do_trial_division = 0*;
(2) call *BN_is_prime_fastest_ex* with *a* greater than the largest prime in the list of small primes; or
(3) independently check if *a* is divisible by a small prime.

To find clients for this library, we searched GitHub using the string "BN_is_prime_fastest_ex", resulting in 11,500 C files. Of the first 1,000 code results, we found 10 unique clients (see Table 1). Five of them were unaffected by the change, calling *BN_is_prime_fastest_ex* with *do_trial_division = 0* (case #1). One of the remaining five (OpenSSL-Elgamal/elegamal_gen.c) is "almost unaffected" as it calls *BN_is_prime_fastest_ex* with a large *a* that just falls short of the largest small prime. No client independently checks *a* over the list of small primes.

***RSA_check_key* (OpenSSL).** The *RSA_check_key* function takes in a pointer to a RSA key and decides its validity. RSA keys are composed of five integer fields: *p*, *q*, *n*, *e*, and *d*. The modification that we considered (`#534e5fa`) adds a check that returns 0 (bad key) if any of these five components are null. This change is irrelevant to clients that:

(1) call *RSA_check_key* with neither *p*, *q*, *n*, *e*, or *d* being null; or
(2) will fail due to null value anyway.

To find clients for this library, we searched GitHub using the string "RSA_check_key". The first 1,000 search results out of the found 24,741 C files contained 32 unique clients (see Table 1). Of these, 27 were unaffected by this library change. 24 clients construct an RSA key by calling either *PEM_read_RSAPrivateKey*, *EVP_PKEY_get1_RSA*, or *RSA_generate_key* and then call *RSA_check_key* with this key. According to the documentation, these three helper functions successfully populate the RSA fields with non-null values or return null. The former situation corresponds to the first property of clients we are looking for, and the latter corresponds to the second (the library will fail given a null value before or after the update). In both situations, the change to the library is irrelevant to the client. There were clients (`#fbf15c7`) that attempted to access the fields before calling *RSA_check_key*. The change did not affect these clients because they will cause a segmentation fault before calling the library in the cases relevant to the change.

The five clients that are affected by this change receive the RSA key as an input parameter or use an unknown function to generate it (e.g., *parse_pk_file*(dudders/crypt_openssl.c), and then call *RSA_check_key*.

[5] For full results, see https://client-specific-equivalence-checker.github.io/

***gcd* (Linux).** The Linux project's *gcd* function calculates the greatest common denominator of two unsigned integer values using the standard Euclidean algorithm. This function, in its original implementation, was vulnerable to division by zero. To circumvent this issue, an update (`#e968756`) was made to check that the smaller of the two input values is not zero. The change is irrelevant to clients that:

(1) call *gcd* with non-zero values; or
(2) are vulnerable to division-by-zero independently of this library.

The string "gcd" is too generic to be used for effectively searching GitHub. We thus limited our search to the Linux project itself, in which we found 11 clients. Of these, three are unaffected by the change. These clients either check that the inputs to *gcd* are non-zero directly (case #1), or use provably positive values (case #2). The remaining clients call the *gcd* function with values set by parameters. In such cases, we cannot be sure that the arguments to *gcd* are non-zero and thus conservatively classify these clients as affected (see Table 1).

***mpf_get_d_2exp* (GMP).** We also considered the function *mpf_get_d_2exp*. Sec. 1. For a partial code listing, see Fig. 2. This change affects the sign of the return when the input is negative, and is irrelevant to clients that:

(1) do not use the returned double (i.e., only use the exponent);
(2) call *mpf_get_d_2exp* with non-negative parameters; or
(3) change the sign of the return of *mpf_get_d_2exp* when the input to this function is negative.

To find clients for this library, we searched GitHub using the string "mpf_get_d_2exp". This search yielded 7,632 C files. Of the first 1,000 code results, we found 7 unique clients, 6 of which were unaffected by the change. Three of the clients did not use the returned double (case #1), one always called the library with positive values (case #2), and one, shown in Fig. 3a, changed the sign of the return when necessary (case #3).

The one client affected by the change, shown in Fig. 3b, calls a function that is undefined on negative inputs with the result returned by *mpf_get_d_2exp*.

1.2 Benchmark Details

We now give a brief description of the examples we created for our experiments. *BN_is_prime_fasttest_ex* (see Sec. 2) inspired the first three programs: *is_prime1*, *is_prime2* and *is_prime3*. These programs use the same library but have different clients. The library is a simple prime checker that takes an unsigned short x and a flag. If the flag is non-zero, the library returns 0; otherwise, it checks whether the input is prime by trial division over the first eight primes. The original version of the library does not check if a composite candidate is one of the first eight primes; the updated library does. The client for *is_prime1* always calls the library with the flag equal to 0; the client for *is_prime2* always calls the library with $x > 19$; the client for *is_prime3* checks whether x is in the list of small primes before calling the library.

RSA_check_key and *gcd* (see Sec. 2) inspired the next two programs, *divide1*, and *divide2*. The programs share a library that was patched to avoid a division by zero. *divide1*'s calls the library with safe arguments, while *divide2* is victim to the same divide-by-zero error as the library. Both programs are client-specific equivalent. In the latter case, the library update preserves partial functional equivalence since the original version is undefined on the value that differs.

mpf_get_d_2exp (see Sec. 2) inspired *order*, *pos1*, and *pos2*. The library in *order* calculates an integer value whose sign depends on the order of its arguments. The update rectifies this by swapping values if necessary to ensure that the return is positive. The client for *order* is aware of this pitfall in the original library, and so it calls the library with arguments in the correct order to ensure a positive return. *pos1* and *pos2* share a client but have distinct library updates. In their case, the client always calls the library with a negative value. Since the libraries differ only on positive inputs, they are equivalent with regards to this client.

Of the remaining examples, *oneN1* is a case where the library changes significantly but the client remains unaffected. *oneN2* and *get_sign* add to our collection of non-equivalent cases. *get_sign2*, *multiple*, and *ltfive* are straightforward equivalences. *odd* provides a case where an infinite loop in the library cannot be avoided but the client is unaffected. Finally, *factorial* and *fib* provide four cases where the library versions are alternate implementations of the same mathematical function. In two cases, the alternate implementation introduces a bug which breaks equivalence.

1.3 Case-study Details

Table 3: Case-study execution times.

Benchmark	time (s)	result
bii_1	1.945	pattern
bii_1_mod	1.913	counter
bii_1_error	2.007	counter
bii_2	1.873	pattern
spec_1	0.043	pattern
spec_1_mod	0.063	counter
spec_1_error	0.041	counter
spec_2	2.118	pattern
spec_2_mod	2.375	assertion
spec_2_error	1.324	counter
spec_2_alt	2.186	pattern
spec_1_alt_mod	2.404	counter
spec_1_alt_error	1.318	counter
spec_3	0.056	pattern
spec_4	2.439	pattern
spec_4_alt	2.169	pattern

Table 3 lists the execution times of Clever on every client-library function pair considered in Sec. 6.4. All pairs are available at https://client-specific-equivalence-checker.github.io/.