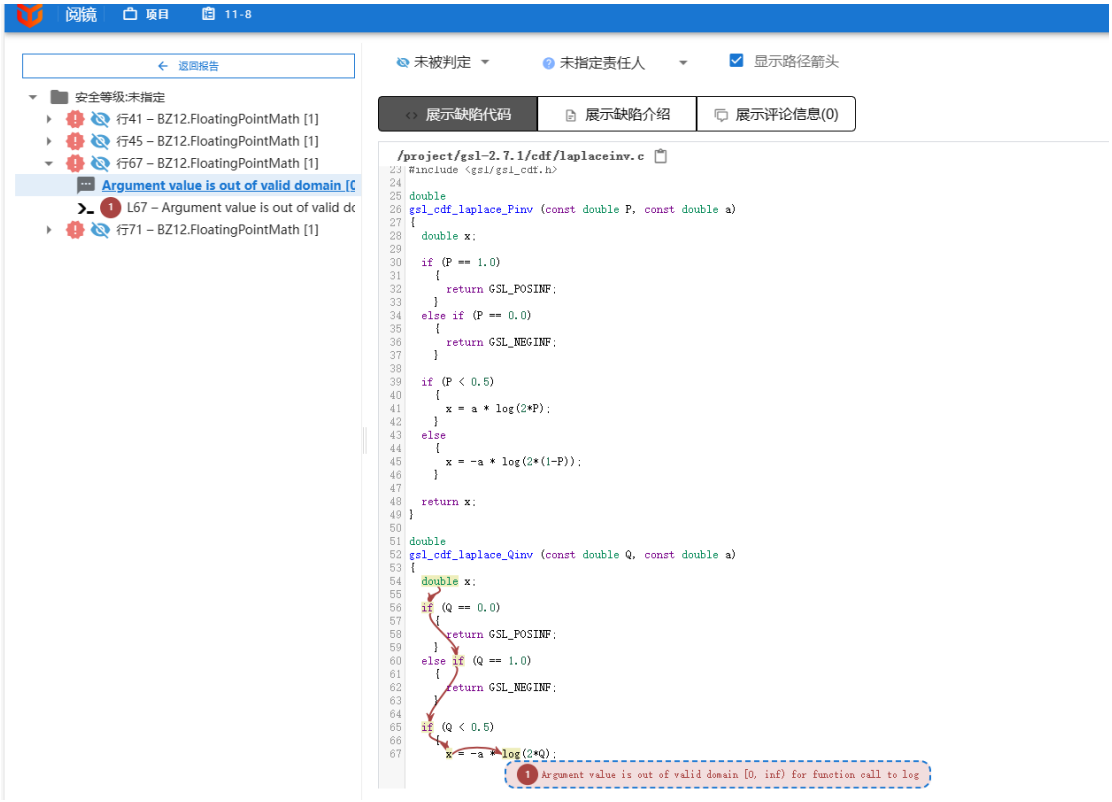
Category 1: invalid operation exception

1. The function gsl_sf_conicalP_xlt1_large_neg_mu_e throws an invalid exception when its input tau=1.0, mu=2.0, x=3.0 at line 221 in file /gsl-2.7.1/specfunc/legendre_con.c: double p = x/sqrt(beta2*(1.0-x*x) + 1.0); Because the parameter of sqrt cannot be negative numbers, invalid exception occurs in this occasion.

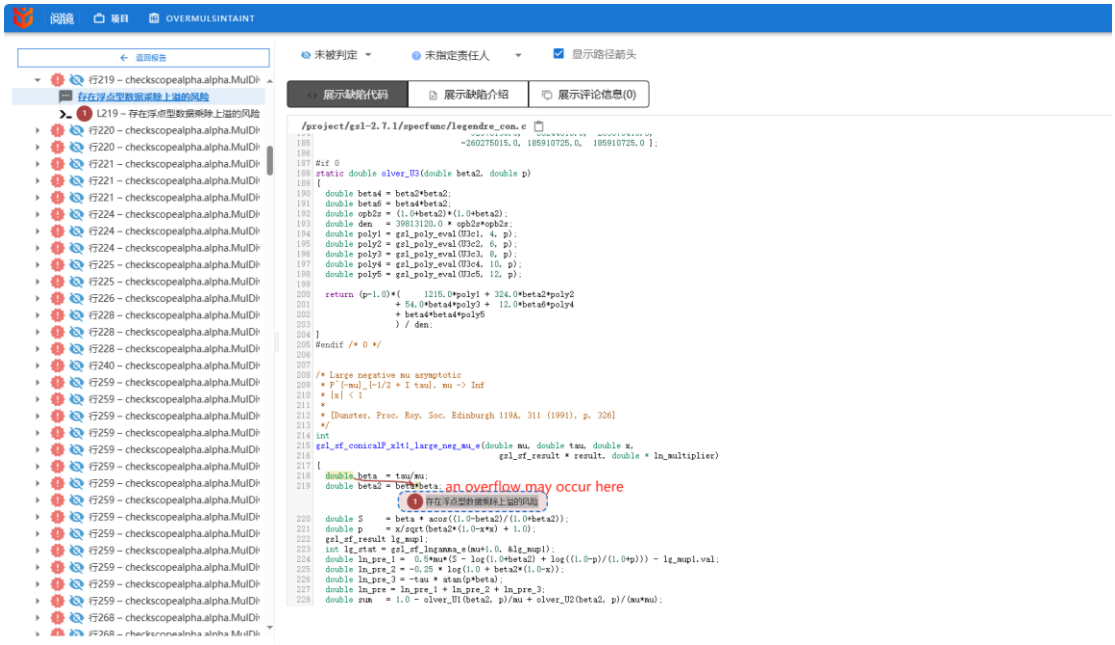
```
214 int
215 gsl_sf_conicalP_xlt1_large_neg_mu_e(double mu, double tau, double x,
216                                     gsl_sf_result * result, double * ln_multiplier)
217 {
218   double beta  = tau/mu;
219   double beta2 = beta*beta;
220   double S     = beta * acos((1.0-beta2)/(1.0+beta2));
221   double p     = x/sqrt(beta2*(1.0-x*x) + 1.0);
                    1 Argument value is out of valid domain [0, inf) for function call to sqrt
222   gsl_sf_result lg_mup1;
223   int lg_stat = gsl_sf_lngamma_e(mu+1.0, &lg_mup1);
224   double ln_pre_1 =  0.5*mu*(S - log(1.0+beta2) + log((1.0-p)/(1.0+p))) - lg_mup1.val;
225   double ln_pre_2 = -0.25 * log(1.0 + beta2*(1.0-x));
226   double ln_pre_3 = -tau * atan(p*beta);
227   double ln_pre = ln_pre_1 + ln_pre_2 + ln_pre_3;
228   double sum    = 1.0 - olver_U1(beta2, p)/mu + olver_U2(beta2, p)/(mu*mu);
229
230   if(sum == 0.0) {
231     result->val = 0.0;
232     result->err = 0.0;
233     *ln_multiplier = 0.0;
234     return GSL_SUCCESS;
235   }
236   else {
237     int stat_e = gsl_sf_exp_mult_e(ln_pre, sum, result);
238     if(stat_e != GSL_SUCCESS) {
239       result->val = sum;
240       result->err = 2.0 * GSL_DBL_EPSILON * fabs(sum);
241       *ln_multiplier = ln_pre;
242     }
243     else {
244       *ln_multiplier = 0.0;
245     }
246     return lg_stat;
247   }
248 }
```

2. The function gsl_cdf_laplace_Qinv throws an invalid exception when its input Q=-0.5 at line 67 in file /gsl-2.7.1/cdf/laplaceinv.c: x = -a * log(2*Q); Because the parameter of log cannot be negative numbers, invalid exception occurs in this occasion.



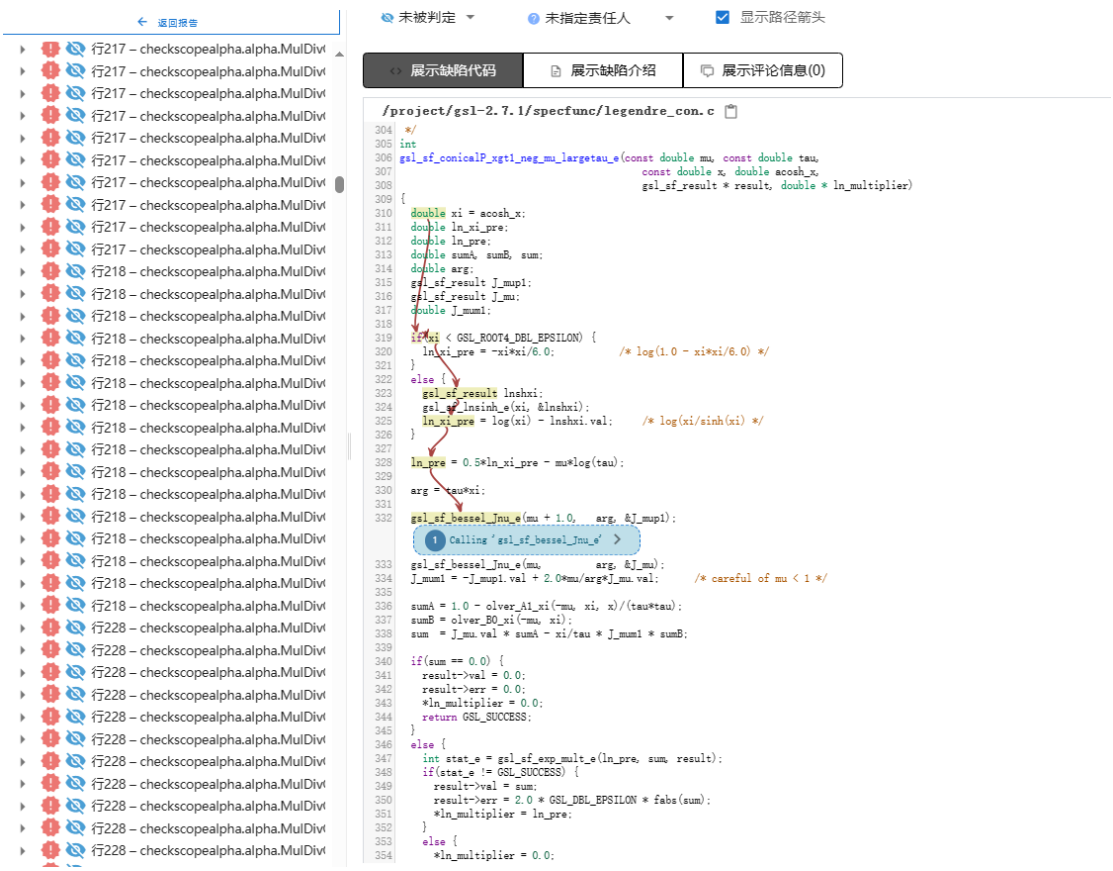
Category 2: overflow

4. The function gsl_sf_conicalP_xlt1_large_neg_mu_e throws a overflow exception when its input tau =1.23e189,mu=1e-2 at line 219 in file /gsl-2.7.1/specfunc/legendre_con.c: double beta2 = beta*beta;



Category 3: underflow

5. The function gsl_sf_bessel_Jnu_asympx_e throws an underflow exception when its input nu=1.23e-189, x=1.0 at line 217 in file /gsl-2.7.1/specfunc/bessel.c: double mu = 4.0*nu*nu;


返回报告
未被判定
未指定责任人
显示路径箭头
行217 – checkscopealpha.alpha.MulDiv
展示缺陷代码
展示缺陷介绍
展示评论信息(0)
/project/gsl-2.7.1/specfunc/legendre_con.c
*/
int
gsl_sf_conicalP_xgt1_neg_mu_largetau_e(const double mu, const double tau,
const double x, double acosh_x,
gsl_sf_result * result, double * ln_multiplier)
{
double xi = acosh_x;
double ln_xi_pre;
double ln_pre;
double sumA, sumB, sum;
double arg;
gsl_sf_result J_mup1;
gsl_sf_result J_mu;
double J_mum1;
if(xi < GSL_ROOT4_DBL_EPSILON) {
ln_xi_pre = -xi*xi/6.0;
/* log(1.0 - xi*xi/6.0) */
}
else {
gsl_sf_result lnshxi;
gsl_sf_lnsinh_e(xi, &lnshxi);
ln_xi_pre = log(xi) - lnshxi.val;
/* log(xi/sinh(xi) */
}
ln_pre = 0.5*ln_xi_pre - mu*log(tau);
arg = tau*xi;
gsl_sf_bessel_Jnu_e(mu + 1.0, arg, &J_mup1);
1 Calling 'gsl_sf_bessel_Jnu_e'
gsl_sf_bessel_Jnu_e(mu, arg, &J_mu);
J_mum1 = -J_mup1.val + 2.0*mu/arg*J_mu.val;
/* careful of mu < 1 */
sumA = 1.0 - olver_A1_xi(-mu, xi, x)/(tau*tau);
sumB = olver_B0_xi(-mu, xi);
sum = J_mu.val * sumA - xi/tau * J_mum1 * sumB;
if(sum == 0.0) {
result->val = 0.0;
result->err = 0.0;
*ln_multiplier = 0.0;
return GSL_SUCCESS;
}
else {
int stat_e = gsl_sf_exp_mult_e(ln_pre, sum, result);
if(stat_e != GSL_SUCCESS) {
result->val = sum;
result->err = 2.0 * GSL_DBL_EPSILON * fabs(sum);
*ln_multiplier = ln_pre;
}
else {
*ln_multiplier = 0.0;

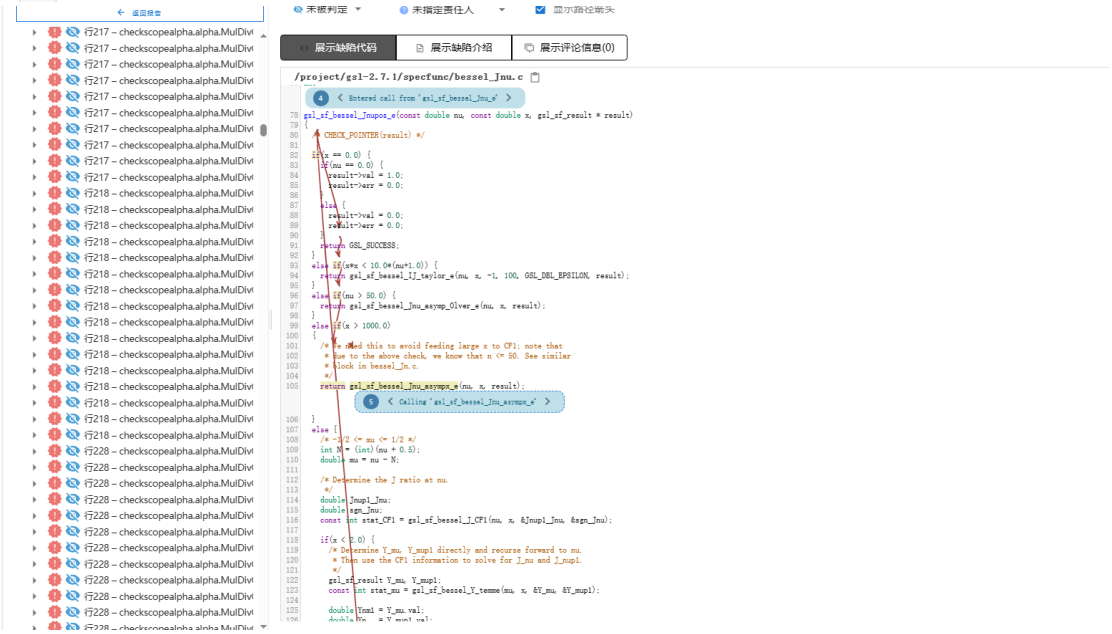
/project/gsl-2.7.1/specfunc/bessel_Jnu.c

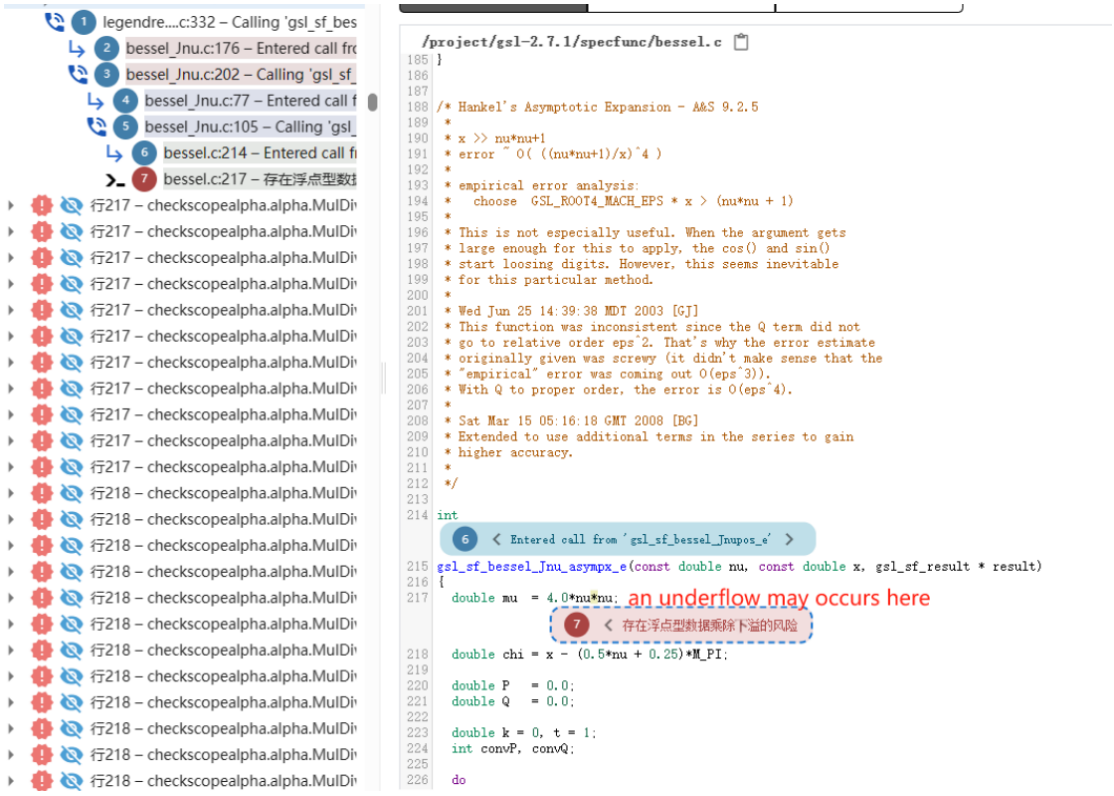
```
155       int n;
156       for(n=N; n>0; n--) {
157         Jnm1 = 2.0*(mu+n)/x * Jn - Jnp1;
158         Jnp1 = Jn;
159         Jn   = Jnm1;
160       }
161       Jmup1_Jmu = Jnp1/Jn;
162       sgn_Jmu   = GSL_SIGN(Jn);
163       Jmuprime_Jmu = mu/x - Jmup1_Jmu;
164
165       gamma = (P - Jmuprime_Jmu)/Q;
166       Jmu   = sgn_Jmu * sqrt(2.0/(M_PI*x) / (Q + gamma*(P-Jmuprime_Jmu)));
167
168       result->val = Jmu * (sgn_Jnu * GSL_SQRT_DBL_MIN) / Jn;
169       result->err = 2.0 * GSL_DBL_EPSILON * (N + 2.0) * fabs(result->val);
170
171       return GSL_ERROR_SELECT_2(stat_CP2, stat_CF1);
172     }
173   }
174 }
175
176 int
```

2 < Entered call from 'gsl_sf_conicalP_xgt1_neg_mu_largetau_e' >

```
177 gsl_sf_bessel_Jnu_e(const double nu, const double x, gsl_sf_result * result)
178 {
179   /* CHECK_POINTER(result) */
180
181   if(x <= 0.0) {
182     DOMAIN_ERROR(result);
183   }
184   else if (nu < 0.0) {
185     int Jstatus = gsl_sf_bessel_Jnupos_e(-nu, x, result);
186     double Jval = result->val;
187     double Jerr = result->err;
188     int Ystatus = gsl_sf_bessel_Ynupos_e(-nu, x, result);
189     double Yval = result->val;
190     double Yerr = result->err;
191     /* double s = sin(M_PI*nu), c = cos(M_PI*nu); */
192     int sinstatus = gsl_sf_sin_pi_e(nu, result);
193     double s = result->val;
194     double serr = result->err;
195     int cosstatus = gsl_sf_cos_pi_e(nu, result);
196     double c = result->val;
197     double cerr = result->err;
198     result->val = s*Yval + c*Jval;
199     result->err = fabs(c*Yerr) + fabs(s*Jerr) + fabs(cerr*Yval) + fabs(serr*Jval);
200     return GSL_ERROR_SELECT_4(Jstatus, Ystatus, sinstatus, cosstatus);
201   }
202   else return gsl_sf_bessel_Jnupos_e(nu, x, result);
```

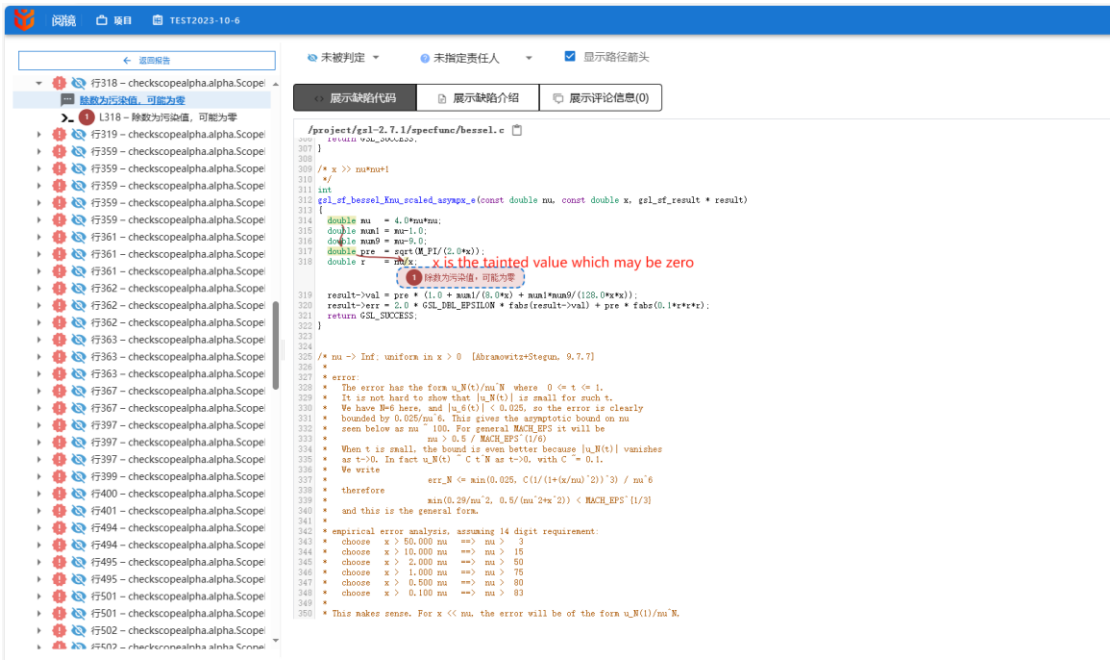
3 < Calling 'gsl_sf_bessel_Jnupos_e' >





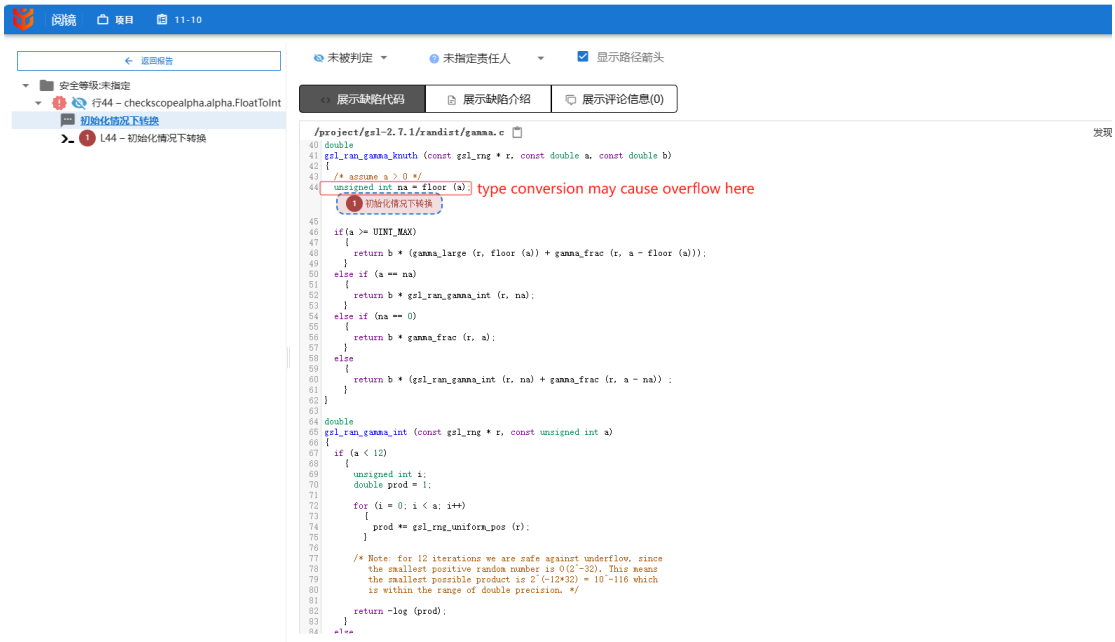
Category 4: division-by-zero

6. The function gsl_sf_bessel_Knu_scaled_asympx_e throws an division-by-zero floating-point exception when its input x=0.0 at line 318 in file /gsl-2.7.1/specfunc/bessel.c: double r = nu/x;



Category 5: type conversion

7. The function gsl_ran_gamma_knuth throws an overflow caused by type conversion exception when its input a=1.23e189 at line 44 in file /gsl-2.7.1/randist/gamma.c: unsigned int na = floor (a);


阅镜
项目
11-10
返回报告
安全等级:未指定
行44 – checkscopealpha.alpha.FloatToInt
初始化情况下转换
L44 – 初始化情况下转换
未被判定
未指定责任人
显示路径箭头
展示缺陷代码
展示缺陷介绍
展示评论信息(0)
/project/gsl-2.7.1/randist/gamma.c
发现
type conversion may cause overflow here
初始化情况下转换
40 double
41 gsl_ran_gamma_knuth (const gsl_rng * r, const double a, const double b)
42 {
43 /* assume a > 0 */
44 unsigned int na = floor (a);
45
46 if(a >= UINT_MAX)
47 {
48 return b * (gamma_large (r, floor (a)) + gamma_frac (r, a - floor (a)));
49 }
50 else if (a == na)
51 {
52 return b * gsl_ran_gamma_int (r, na);
53 }
54 else if (na == 0)
55 {
56 return b * gamma_frac (r, a);
57 }
58 else
59 {
60 return b * (gsl_ran_gamma_int (r, na) + gamma_frac (r, a - na)) ;
61 }
62 }
63
64 double
65 gsl_ran_gamma_int (const gsl_rng * r, const unsigned int a)
66 {
67 if (a < 12)
68 {
69 unsigned int i;
70 double prod = 1;
71
72 for (i = 0; i < a; i++)
73 {
74 prod *= gsl_rng_uniform_pos (r);
75 }
76
77 /* Note: for 12 iterations we are safe against underflow, since
78 the smallest positive random number is O(2^-32). This means
79 the smallest possible product is 2^(-12*32) = 10^-116 which
80 is within the range of double precision. */
81
82 return -log (prod);
83 }
84 else