Appendix A

FORM Programs

Here we show the FORM [59] programs used to calculate the second functional derivative matrix appearing in Eq. (4.3). Due to the great number of indices appearing in the calculation of functional derivatives, we shall use Greek indices instead of Latin ones. In all calculations the Greek indices take the values 1 and 2.

A.1 Calculation of functional derivatives

The following FORM program calculates the second functional derivative

$$\frac{\delta^2 E_0}{\delta\phi(x)\delta\phi(y)}, \tag{A.1}$$

according to the Euler-Lagrange formula, of the energy functional

$$E_0 = r_0 \int \mathrm{d}^2x \sqrt{1+(\partial\phi)^2} + \frac{\kappa_0}{2}\int \mathrm{d}^2x \left\{ \frac{(\partial^2\phi)^2}{[1+(\partial\phi)^2]^{1/2}} \right.$$
$$\left. -2\frac{\partial_\mu\phi\partial_\nu\phi\partial_\mu\partial_\nu\phi\partial^2\phi}{[1+(\partial\phi)^2]^{3/2}} + \frac{(\partial_\mu\phi\partial_\nu\phi\partial_\mu\partial_\nu\phi)^2}{[1+(\partial\phi)^2]^{5/2}} \right\}. \tag{A.2}$$

It also implements the substitution $v_\mu(x) = \bar{v}_\mu + \hat{v}_\mu(x)$, where $\bar{v}_\mu$ is a constant field and the expansion of the resulting expression in powers of $\hat{v}_\mu(x)$ and its

derivatives, up to second order. We adopt the conventions

phi(mu) $\equiv \partial_\mu \phi$	phi(mu,nu) $\equiv \partial_\mu \partial_\nu \phi, \ldots$
dd(mu) $\equiv \dfrac{\partial}{\partial \partial_\mu \phi}$	dd(mu,nu) $\equiv \dfrac{\partial}{\partial \partial_\mu \partial_\nu \phi}, \ldots$
vt(mu) $\equiv \hat{v}_\mu$	vt(mu,nu) $\equiv \partial_\nu \hat{v}_\mu, \ldots$
v0(mu) $\equiv \bar{v}_\mu$	[1+phi(mu).phi(mu)] $\equiv 1 + (\partial\phi)^2$.

```
*
*                    Full_Vertex_sum.form
*
* This program calculates the second functional derivative of the full
* Hamiltonian, and expands the result around a constant field.
*
Functions phi,partial,dd,Vs,V1,V2,V3,V4,[1+phi(mu).phi(mu)],b,p,vt;
Cfunctions v0,[1+v0^2];
Indices mu=2,nu=2,mu2=2,nu2=2,mu3=2,nu3=2,
        sigma=2,sigma1=2,sigma2=2,sigma3=2,sigma4=2,
        sigma5=2,sigma6=2,sigma7=2,
        sigma8=2,alpha=2,alphap=2, alphapp=2, alphappp=2,
        beta=2,betap=2,betapp=2,betappp=2,
        rho=2,rho2=2,rho3=2,rho4=2,rho5=2;
Symbols n,np,nf,nd,m,kappa;

* Calculation of the first functional derivative according to the
* Euler-Lagrange formula

Local F1 = - partial(sigma1) * dd(sigma1) * Vs * b +
 partial(sigma1)*partial(sigma2)*dd(sigma1,sigma2)*Vs*b;

id Vs =   m*V1 + (kappa/2)*V2 -kappa*V3 + (kappa/2)*V4;

id V1 = [1+phi(mu).phi(mu)]^(1/2);

id V2 = [1+phi(mu).phi(mu)]^(-1/2)*phi(rho,rho)*phi(rho2,rho2);

id V3 = [1+phi(mu).phi(mu)]^(-3/2)*phi(rho2)*phi(rho3)*phi(rho2,rho3)*
        phi(rho,rho);
```

```
id V4 = [1+phi(mu).phi(mu)]^(-5/2)*phi(rho2)*phi(rho3)*phi(rho2,rho3)*
        phi(rho4)*phi(rho5)*phi(rho4,rho5);

* Here the chain rule for the functional derivatives is implemented

repeat;
id dd(nu?)*[1+phi(mu).phi(mu)]^(n?)= 2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(nu)
                                     + [1+phi(mu).phi(mu)]^(n)*dd(nu);
id dd(nu?) * phi(mu?) =  d_(mu,nu)  + phi(mu)* dd(nu) ;
id dd(nu?) * phi(mu?,mu2?) = phi(mu,mu2)*dd(nu);
id dd(mu?,nu?)*[1+phi(mu).phi(mu)]^(n?)=[1+phi(mu).phi(mu)]^(n)*dd(mu,nu);
id dd(mu?,nu?)*phi(mu2?)=phi(mu2)*dd(mu,nu);
id dd(mu?,nu?)*phi(mu2?,nu2?)=d_(mu,mu2)*d_(nu,nu2)+phi(mu2,nu2)*dd(mu,nu);
endrepeat;

id dd(nu?) * b = 0;
id dd(mu?,nu?)*b =0;

* Here the chain rule for the partial derivatives is implemented. The first
* two steps must stay out of the loop in order to avoid ambiguities in the
* sum indices.

id  partial(mu?)*[1+phi(mu).phi(mu)]^(n?)=
       2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(alpha)*phi(mu,alpha)
       + [1+phi(mu).phi(mu)]^(n)*partial(mu);

id  partial(mu?)*[1+phi(mu).phi(mu)]^(n?)=
       2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(alphap)*phi(mu,alphap)
       + [1+phi(mu).phi(mu)]^(n)*partial(mu);

repeat;
id partial(mu?)*phi(nu?)=phi(nu,mu)+phi(nu)* partial(mu);
id partial(mu?)*phi(mu2?,nu2?)=phi(mu2,nu2,mu)+phi(mu2,nu2)*partial(mu);
id partial(mu?)*phi(nu?,mu2?,nu2?)=phi(nu,mu2,nu2,mu)
                                  +phi(nu,mu2,nu2)*partial(mu);
endrepeat;

id partial(mu?) * b =0;
```

```
* Now the first functional derivative is printed out.

print;
.sort

* Calculation of the second functional derivative

* Now the Euler-Lagrange formula is applied to the first
* functional derivative

drop F1;

Local F2 = - partial(sigma3) * dd(sigma3) * F1 +
partial(sigma3) * partial(sigma4) * dd(sigma3,sigma4)* F1 -
partial(sigma3)*partial(sigma4)*partial(sigma5)
  *dd(sigma3,sigma4,sigma5)*F1+
partial(sigma3)*partial(sigma4)*partial(sigma5)*partial(sigma6)
*dd(sigma3,sigma4,sigma5,sigma6)*F1;

* Here the chain rule for the functional derivatives is implemented

repeat;
id dd(nu?)*[1+phi(mu).phi(mu)]^(n?)=2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(nu)
                                    + [1+phi(mu).phi(mu)]^(n)*dd(nu);
id dd(nu?) * phi(mu?) = d_(mu,nu)+phi(mu)* dd(nu) ;
id dd(nu?) * phi(mu?,mu2?) = phi(mu,mu2)*dd(nu);
id dd(nu?) * phi(mu?,mu2?,mu3?) = phi(mu,mu2,mu3)*dd(nu);
id dd(nu?) * phi(mu?,mu2?,mu3?,nu3?) = phi(mu,mu2,mu3,nu3)*dd(nu);


id dd(mu?,nu?)*[1+phi(mu).phi(mu)]^(n?)=[1+phi(mu).phi(mu)]^(n)*dd(mu,nu);
id dd(mu?,nu?)*phi(mu2?)=phi(mu2)*dd(mu,nu);
id dd(mu?,nu?)*phi(mu2?,nu2?)= d_(mu,mu2)*d_(nu,nu2)
                               +phi(mu2,nu2)*dd(mu,nu);
id dd(mu?,nu?)*phi(mu2?,nu2?,nu3?) = phi(mu2,nu2,nu3)*dd(mu,nu);
id dd(mu?,nu?)*phi(mu2?,nu2?,nu3?,mu3?) = phi(mu2,nu2,nu3,mu3)*dd(mu,nu);
```

```
id dd(mu?,nu?,mu2?)*[1+phi(mu).phi(mu)]^(n?) =
                                    [1+phi(mu).phi(mu)]^(n)*dd(mu,nu,mu2);
id dd(mu?,nu?,nu2?)*phi(mu2?)=phi(mu2)*dd(mu,nu,nu2);
id dd(mu?,nu?,mu3?)*phi(mu2?,nu2?)=  phi(mu2,nu2)*dd(mu,nu,mu3);
id dd(mu?,nu?,mu3?)*phi(mu2?,nu2?,nu3?)= d_(mu,mu2)*d_(nu,nu2)*d_(mu3,nu3) +
                                         phi(mu2,nu2,nu3)*dd(mu,nu,mu3);
id dd(mu?,nu?,mu3?)*phi(mu2?,nu2?,nu3?,alpha?)=
                                    phi(mu2,nu2,nu3,alpha)*dd(mu,nu,mu3);



id dd(mu?,nu?,mu2?,nu2?)*[1+phi(mu).phi(mu)]^(n?) =
                                  [1+phi(mu).phi(mu)]^(n)*dd(mu,nu,mu2,nu2);
id dd(mu?,nu?,mu2?,nu2?)*phi(mu3?)=phi(mu3)*dd(mu,nu,mu2,nu2);
id dd(mu?,nu?,mu2?,nu2?)*phi(mu3?,nu3?)= phi(mu3,nu3)*dd(mu,nu,mu2,nu2);
id dd(mu?,nu?,mu2?,nu2?)*phi(mu3?,nu3?,alpha?)=
                                   phi(mu3,nu3,alpha)*dd(mu,nu,mu2,nu2);
id dd(mu?,nu?,mu2?,nu2?)*phi(mu3?,nu3?,alpha?,beta?)=
                           d_(mu,mu3)*d_(nu,nu3)*d_(mu2,alpha)*d_(nu2,beta)+
                           phi(mu3,nu3,alpha,beta)*dd(mu,nu,mu2,nu2);
endrepeat;

id dd(nu?) * b = 0;
id dd(nu?,mu?) * b = 0;
id dd(nu?,mu?,mu2?)*b = 0;
id dd(mu?,nu?,mu2?,nu2?)*b = 0;

* Here the chain rule for the partial derivatives is implemented. The first
* four steps must stay out of the loop in order to avoid ambiguities in the
* sum indices.

id partial(mu?)*[1+phi(mu).phi(mu)]^(n?)=
            2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(beta)*phi(mu,beta)
            + [1+phi(mu).phi(mu)]^(n)*partial(mu);

id partial(mu?)*[1+phi(mu).phi(mu)]^(n?)=
            2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(betap)*phi(mu,betap)
```

```
            + [1+phi(mu).phi(mu)]^(n)*partial(mu);

id partial(mu?)*[1+phi(mu).phi(mu)]^(n?)=
            2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(betapp)*phi(mu,betapp)
            + [1+phi(mu).phi(mu)]^(n)*partial(mu);

id partial(mu?)*[1+phi(mu).phi(mu)]^(n?)=
            2*n*[1+phi(mu).phi(mu)]^(n-1)*phi(betappp)*phi(mu,betappp)
            + [1+phi(mu).phi(mu)]^(n)*partial(mu);

repeat;
id partial(mu?)*phi(nu?)=phi(nu,mu)+phi(nu)*partial(mu);
id partial(mu?)*phi(mu2?,nu2?)=phi(mu2,nu2,mu)+phi(mu2,nu2)*partial(mu);
id partial(mu?)*phi(mu2?,nu2?,mu3?) = phi(mu2,nu2,mu3,mu)
                              + phi(mu2,nu2,mu3)*partial(mu);
id partial(mu?)*phi(mu2?,nu2?,mu3?,nu3?) = phi(mu2,nu2,mu3,nu3,mu)
                                  + phi(mu2,nu2,mu3,nu3)*partial(mu);
id partial(mu?)*phi(mu2?,nu2?,mu3?,nu3?,alpha?) =
                              phi(mu2,nu2,mu3,nu3,alpha,mu)
                              + phi(mu2,nu2,mu3,nu3,alpha)*partial(mu);
endrepeat;

id b = 1;

id partial(mu?) = i_  *p(mu);

symmetrize phi;

* Here the expansion around a constant field is implemented

id phi(mu?) = nf*vt(mu) + v0(mu);
id phi(mu?,nu?)= nf*nd*vt(mu,nu);
id phi(mu?,nu?,mu2?) = nf*nd^2*vt(mu,nu,mu2);
id phi(mu?,nu?,mu2?,nu2?) = nf*nd^3*vt(mu,nu,mu2,nu2);
id phi(mu?,nu?,mu2?,nu2?,mu3?) = nf*nd^4*vt(mu,nu,mu2,nu2,mu3);
id phi(mu?,nu?,mu2?,nu2?,mu3?,nu3?)= nf*nd^5*vt(mu,nu,mu2,nu2,mu3,nu3);

symmetrize vt;
```

```
* Only the terms containing up to 2 derivatives are kept

id nd^14 =0;
id nd^13 =0;
id nd^12 =0;
id nd^11 =0;
id nd^10 =0;
id nd^9 =0;
id nd^8 =0;
id nd^7 =0;
id nd^6 =0;
id nd^5 =0;
id nd^4 =0;
id nd^3 =0;

id [1+phi(mu).phi(mu)]^(n?)*[1+phi(mu).phi(mu)]^(np?)=
   [1+phi(mu).phi(mu)]^(n+np);

* Here the expansion of the denominator is implemented

id [1+phi(mu).phi(mu)]^(n?)=
[1+v0^2]^(n)*(1+ n*(2*v0(alphapp)*nf*vt(alphapp))/[1+v0^2]
+ (1/2)*n*(n-1)*(2*v0(alphapp)*nf*vt(alphapp))*
(2*v0(alphappp)*nf*vt(alphappp))/[1+v0^2]^2);

* Only the terms containing up to 2 fields are kept

id nf^14 =0;
id nf^13 =0;
id nf^12 =0;
id nf^11 =0;
id nf^10 =0;
id nf^9 =0;
id nf^8 =0;
id nf^7 =0;
id nf^6 =0;
id nf^5 =0;
```

```
id nf^4 =0;
id nf^3 =0;

* Here we sum over all indices

sum rho, rho2, rho3,rho4,rho5;
sum sigma1, sigma2, sigma3, sigma4, sigma5, sigma6;
sum alpha, alphap, alphapp, alphappp, beta, betap, betapp, betappp;
Bracket nf;
.sort

* The resulting terms are sorted in powers of the field

drop F2;

Local F3 = [1+v0^2]^(1/2)*F2[1];

Local F4 = [1+v0^2]^(1/2)*F2[nf];

Local F5 = [1+v0^2]^(1/2)*F2[nf^2];

id 1/[1+v0^2] = [1+v0^2]^(-1);
id 1/([1+v0^2]*[1+v0^2]) = [1+v0^2]^(-2);
id [1+v0^2]^(n?)*[1+v0^2]^(np?)=[1+v0^2]^(n+np);

Bracket nd;
print;
.end
```

A.2 Calculation of the functional trace

The matrix elements generated with the previous program are used to calculate the expansion of the functional trace in (4.3). In the first term of the expansion, no commutators are needed to shift the momentum operators to the left, so that the calculation amounts to evaluating integrals. The following FORM program calculates the first term in the expansion of the functional trace (4.15), and carries out the corresponding integrals.

```
*
*                                G2.form
*
* This program calculates the first term of the expansion of the Tr ln
*
Functions L2phi,vt,p;
Cfunctions G,k,v0,Z,Z1,Z2,Z3,v0,F2,G2,q;
Indices mu,nu,mu1,nu1,mu2,nu2,mu3,nu3,
        sigma,N1,N2,N3,N4,N5,N6,
        N7,N8,N9,M1,M2,M3,M4,M5,
        M6,M7,M8,M9,
        alpha,beta,alphap,betap,
        rho,rho1,rho2,rho3,rho4,rho5
        lambda, lambda1, lambda2
        llambda, llambda1, llambda2, gamma;
Symbols n,np,nf,nd,m,kappa,[k^2],[v0^2],Pi,prop,[1+v0^2],[q^2],[1-q^2];


Local GL1 = (1/2)*G*L2phi;


* G corresponds to the propagator

id G = prop;

* The matrix elements quadratic in vt are stored in the file L2phi.input
* They are now included in the program

#include L2phi.input

* Since there are no commutators involved at this step, we may write

id p(alpha?) = k(alpha);

id i_ =0;

* The relations (B.1) are implemented here
```

```
id k(mu?)*k(nu?)*prop = F2*d_(mu,nu) + G2*q(mu)*q(nu)/[q^2];

* The numerical value of the integrals goes here

id F2 = (1/(4*Pi*kappa))*[1-q^2]^(-1/2);
id G2 = (1/(4*Pi*kappa))*[q^2]*[1-q^2]^(-3/2);

id q(mu?) = v0(mu)*[1+v0^2]^(-1/2);
id v0(mu?)*v0(mu?) = [v0^2];
id [1-q^2]^(n?) = [1+v0^2]^(-n);
id [q^2] = [v0^2]*[1+v0^2]^(-1);
id [v0^2] = [1+v0^2] - 1;
id [1+v0^2]^(n?)*[1+v0^2]^(np?)=[1+v0^2]^(n+np);
id [1+v0^2]^(n?)*[1+v0^2]^np?=[1+v0^2]^(n+np);

* Partial integrations are implemented here

id vt(mu?,nu?)*vt(alpha?,beta?) = Z(mu,nu,alpha,beta);
id vt(mu?)*vt(nu?,alpha?,beta?) = -Z(mu,nu,alpha,beta);
id vt(nu?,alpha?,beta?)*vt(mu?) = -Z(mu,nu,alpha,beta);

* Contraction of indices puts the results in the form in which
* they should appear in the effective energy

id Z(mu?,mu?,nu?,nu?) = Z1;
id Z(mu?,nu?,mu?,nu?) = Z1;
id Z(nu?,mu?,mu?,nu?) = Z1;

id v0(mu?)*v0(nu?)*Z(mu?,nu?,alpha?,alpha?) = Z2;
id v0(mu?)*v0(nu?)*Z(mu?,alpha?,nu?,alpha?) = Z2;
id v0(mu?)*v0(nu?)*Z(mu?,alpha?,alpha?,nu?) = Z2;
id v0(mu?)*v0(nu?)*Z(alpha?,alpha?,mu?,nu?) = Z2;
id v0(mu?)*v0(nu?)*Z(alpha?,mu?,alpha?,nu?) = Z2;
id v0(mu?)*v0(nu?)*Z(alpha?,mu?,nu?,alpha?) = Z2;


id v0(mu?)*v0(nu?)*v0(alpha?)*v0(beta?)*Z(mu?,nu?,alpha?,beta?) = Z3;
```

```
id nd=1;
Bracket Z1,Z2,Z3;
print;
.end
```

The second term in the expansion (4.15) is calculated here. Since now the commutation relation (4.16) has to be implemented, resulting in too many terms for a single calculation, the integrals are carried out later in a separate program.

```
*
*                               G11.form
*
* This program calculates the second term of the expansion of the Tr ln
*
Functions L1phi,vt,p,Gp,partial,b;
Cfunctions G,k,v0,Z,Z1,Z2,Z3,v0,F2,G2,q,L1phik;
Indices mu,nu,mu1,nu1,mu2,nu2,mu3,nu3,mu4,nu4,
        sigma,N1,N2,N3,N4,N5,N6,
        N7,N8,N9,M1,M2,M3,M4,M5,
        M6,M7,M8,M9,
        alpha,beta,alphap,betap,
        rho,rho1,rho2,rho3,rho4,rho5
        lambda, lambda1, lambda2
        llambda, llambda1, llambda2, gamma;
Symbols n,np,nf,nd,m,kappa,[k^2],[v0^2],Pi,prop,[1+v0^2],[q^2],[1-q^2];

Local GL1 = -(1/4)*G*L1phik*Gp*L1phi*b;

* G is the propagator on the left side
* Gp is the propagator where the commutation rule has been applied

id G = prop;

id Gp = prop

 - prop^2* ( + nd^2 * ( - partial(mu1)*partial(mu1)*m -
   partial(mu1)*partial(mu1)* k(mu2)*k(mu2)*kappa +
```

```
partial(mu1)*partial(mu1)*k(mu2)*k(mu3)*v0(mu2)
*v0(mu3)*kappa*[1+v0^2]^-1 -
4*partial(mu1)*partial(mu2)*k(mu1)*k(mu2 )*kappa +
2*partial(mu1)*partial(mu2)*k(mu1)*k(mu3)*v0(mu2)*v0(mu3)*
kappa*[1+v0^2]^-1 +
partial(mu1)*partial(mu2)*k(mu3)*k(mu3)*v0(mu1)*
v0(mu2)*kappa*[1+v0^2]^-1 -
partial(mu1)*partial(mu2)*k(mu3)*k(mu4)*
v0(mu1)*v0(mu2)*v0(mu3)*v0(mu4)*kappa*[1+v0^2]^-2 + partial(mu1)*
partial(mu2)*v0(mu1)*v0(mu2)*m*[1+v0^2]^-1 +
2*partial(mu1)*partial(
mu3)*k(mu1)*k(mu2)*v0(mu2)*v0(mu3)*kappa*[1+v0^2]^-1 +
2*partial(mu1)
*partial(mu3)*k(mu2)*k(mu3)*v0(mu1)*v0(mu2)*kappa*[1+v0^2]^-1 -
partial(mu1)*partial(mu3)*k(mu2)*k(mu4)*v0(mu1)*v0(mu2)*v0(mu3)*v0(
mu4)*kappa*[1+v0^2]^-2 -
partial(mu1)*partial(mu4)*k(mu2)*k(mu3)*v0(
mu1)*v0(mu2)*v0(mu3)*v0(mu4)*kappa*[1+v0^2]^-2 - partial(mu2)*
partial(mu2)*k(mu1)*k(mu1)*kappa +
partial(mu2)*partial(mu3)*k(mu1)*
k(mu1)*v0(mu2)*v0(mu3)*kappa*[1+v0^2]^-1 +
2*partial(mu2)*partial(mu3
)*k(mu1)*k(mu3)*v0(mu1)*v0(mu2)*kappa*[1+v0^2]^-1 - partial(mu2)*
partial(mu3)*k(mu1)*k(mu4)*v0(mu1)*v0(mu2)*v0(mu3)*v0(mu4)*kappa*
[1+v0^2]^-2 -
partial(mu2)*partial(mu4)*k(mu1)*k(mu3)*v0(mu1)*v0(mu2)
*v0(mu3)*v0(mu4)*kappa*[1+v0^2]^-2 +
partial(mu3)*partial(mu3)*k(mu1)
*k(mu2)*v0(mu1)*v0(mu2)*kappa*[1+v0^2]^-1 -
partial(mu3)*partial(mu4)
*k(mu1)*k(mu2)*v0(mu1)*v0(mu2)*v0(mu3)*v0(mu4)*kappa*[1+v0^2]^-2 )

+ nd * ( 2*partial(mu1)*k(mu1)*i_*m +
  2*partial(mu1)*k(mu1)*k(mu2)*k( mu2)*i_*kappa -
  2*partial(mu1)*k(mu1)*k(mu2)*k(mu3)*v0(mu2)*v0(mu3)*
  i_*kappa*[1+v0^2]^-1 -
  partial(mu1)*k(mu2)*k(mu3)*k(mu3)*v0(mu1)*v0(
  mu2)*i_*kappa*[1+v0^2]^-1 +
```

```
     partial(mu1)*k(mu2)*k(mu3)*k(mu4)*v0(mu1)
     *v0(mu2)*v0(mu3)*v0(mu4)*i_*kappa*[1+v0^2]^-2 -
     partial(mu1)*k(mu2)* v0(mu1)*v0(mu2)*i_*m*[1+v0^2]^-1 +
     2*partial(mu2)*k(mu1)*k(mu1)*k(mu2 )*i_*kappa -
     partial(mu2)*k(mu1)*k(mu1)*k(mu3)*v0(mu2)*v0(mu3)*i_*
     kappa*[1+v0^2]^-1 -
     partial(mu2)*k(mu1)*k(mu3)*k(mu3)*v0(mu1)*v0(mu2)
     *i_*kappa*[1+v0^2]^-1 +
     partial(mu2)*k(mu1)*k(mu3)*k(mu4)*v0(mu1)*v0(
     mu2)*v0(mu3)*v0(mu4)*i_*kappa*[1+v0^2]^-2 -
     partial(mu2)*k(mu1)*v0( mu1)*v0(mu2)*i_*m*[1+v0^2]^-1 -
     partial(mu3)*k(mu1)*k(mu1)*k(mu2)*v0(
     mu2)*v0(mu3)*i_*kappa*[1+v0^2]^-1 -
     2*partial(mu3)*k(mu1)*k(mu2)*k(
     mu3)*v0(mu1)*v0(mu2)*i_*kappa*[1+v0^2]^-1 +
     partial(mu3)*k(mu1)*k(mu2
     )*k(mu4)*v0(mu1)*v0(mu2)*v0(mu3)*v0(mu4)*i_*kappa*[1+v0^2]^-2 +
     partial(mu4)*k(mu1)*k(mu2)*k(mu3)*v0(mu1)*v0(mu2)*v0(mu3)*v0(mu4)*i_*
     kappa*[1+v0^2]^-2 ))

+ prop^3*
  nd * ( 2*partial(mu1)*k(mu1)*i_*m +
  2*partial(mu1)*k(mu1)*k(mu2)*k( mu2)*i_*kappa -
  2*partial(mu1)*k(mu1)*k(mu2)*k(mu3)*v0(mu2)*v0(mu3)*
  i_*kappa*[1+v0^2]^-1 - partial(mu1)*k(mu2)*k(mu3)*k(mu3)*v0(mu1)*v0(
  mu2)*i_*kappa*[1+v0^2]^-1 +
  partial(mu1)*k(mu2)*k(mu3)*k(mu4)*v0(mu1)
  *v0(mu2)*v0(mu3)*v0(mu4)*i_*kappa*[1+v0^2]^-2 - partial(mu1)*k(mu2)*
  v0(mu1)*v0(mu2)*i_*m*[1+v0^2]^-1 +
  2*partial(mu2)*k(mu1)*k(mu1)*k(mu2 )*i_*kappa -
  partial(mu2)*k(mu1)*k(mu1)*k(mu3)*v0(mu2)*v0(mu3)*i_*
  kappa*[1+v0^2]^-1 -
  partial(mu2)*k(mu1)*k(mu3)*k(mu3)*v0(mu1)*v0(mu2)
  *i_*kappa*[1+v0^2]^-1 +
  partial(mu2)*k(mu1)*k(mu3)*k(mu4)*v0(mu1)*v0(
  mu2)*v0(mu3)*v0(mu4)*i_*kappa*[1+v0^2]^-2 - partial(mu2)*k(mu1)*v0(
  mu1)*v0(mu2)*i_*m*[1+v0^2]^-1 -
  partial(mu3)*k(mu1)*k(mu1)*k(mu2)*v0(
```

```
  mu2)*v0(mu3)*i_*kappa*[1+v0^2]^-1 - 2*partial(mu3)*k(mu1)*k(mu2)*k(
  mu3)*v0(mu1)*v0(mu2)*i_*kappa*[1+v0^2]^-1 +
  partial(mu3)*k(mu1)*k(mu2
  )*k(mu4)*v0(mu1)*v0(mu2)*v0(mu3)*v0(mu4)*i_*kappa*[1+v0^2]^-2 +
  partial(mu4)*k(mu1)*k(mu2)*k(mu3)*v0(mu1)*v0(mu2)*v0(mu3)*v0(mu4)*i_*
  kappa*[1+v0^2]^-2 )*

  nd * ( 2*partial(nu1)*k(nu1)*i_*m + 2*partial(nu1)*k(nu1)*k(nu2)*k(
    nu2)*i_*kappa -
    2*partial(nu1)*k(nu1)*k(nu2)*k(nu3)*v0(nu2)*v0(nu3)*
    i_*kappa*[1+v0^2]^-1 -
    partial(nu1)*k(nu2)*k(nu3)*k(nu3)*v0(nu1)*v0(
    nu2)*i_*kappa*[1+v0^2]^-1 +
    partial(nu1)*k(nu2)*k(nu3)*k(nu4)*v0(nu1)
    *v0(nu2)*v0(nu3)*v0(nu4)*i_*kappa*[1+v0^2]^-2 -
    partial(nu1)*k(nu2)* v0(nu1)*v0(nu2)*i_*m*[1+v0^2]^-1 +
    2*partial(nu2)*k(nu1)*k(nu1)*k(nu2 )*i_*kappa -
    partial(nu2)*k(nu1)*k(nu1)*k(nu3)*v0(nu2)*v0(nu3)*i_*
    kappa*[1+v0^2]^-1 -
    partial(nu2)*k(nu1)*k(nu3)*k(nu3)*v0(nu1)*v0(nu2)
    *i_*kappa*[1+v0^2]^-1 +
    partial(nu2)*k(nu1)*k(nu3)*k(nu4)*v0(nu1)*v0(
    nu2)*v0(nu3)*v0(nu4)*i_*kappa*[1+v0^2]^-2 -
    partial(nu2)*k(nu1)*v0( nu1)*v0(nu2)*i_*m*[1+v0^2]^-1 -
    partial(nu3)*k(nu1)*k(nu1)*k(nu2)*v0(
    nu2)*v0(nu3)*i_*kappa*[1+v0^2]^-1 -
    2*partial(nu3)*k(nu1)*k(nu2)*k(
    nu3)*v0(nu1)*v0(nu2)*i_*kappa*[1+v0^2]^-1 +
    partial(nu3)*k(nu1)*k(nu2
    )*k(nu4)*v0(nu1)*v0(nu2)*v0(nu3)*v0(nu4)*i_*kappa*[1+v0^2]^-2 +
    partial(nu4)*k(nu1)*k(nu2)*k(nu3)*v0(nu1)*v0(nu2)*v0(nu3)*v0(nu4)*i_*
    kappa*[1+v0^2]^-2 );


* The matrix elements quadratic in vt are stored in the files
* L1phi.input and L1phik.input
* They are now included in the program
```

```
#include L1phik.input

#include L1phi.input

* Now the commutation rule is implemented
* nd counts the number of derivatives of each term

id p(mu?) = k(mu) + i_*nd*partial(mu);

id nd^6=0;
id nd^5=0;
id nd^4=0;
id nd^3=0;

id i_ = 0;

* Partial integrations are implemented here

id partial(mu?)*vt(nu?) = vt(nu,mu) + vt(nu)*partial(mu);

id partial(mu?)*vt(nu?,alpha?) = vt(nu,alpha,mu) + vt(nu,alpha)*partial(mu);

id partial(mu?)*b =0;

id partial(mu?)*vt(nu?) = vt(nu,mu) + vt(nu)*partial(mu);

id partial(mu?)*vt(nu?,alpha?) = vt(nu,alpha,mu) + vt(nu,alpha)*partial(mu);

id partial(mu?)*b =0;

id b=1;

* Now indices are contracted

id k(mu?)*k(mu?) = [k^2];

id k(mu?) = np*k(mu);
```

```
Bracket nd;

.sort

drop GL1;

* Only terms containing 2 derivatives of vt are kept.

Local GL2 = GL1[nd^2];

Bracket np;

.sort

* Terms are sorted according to the powers of the propagator

Local I0 = GL2[1];

Local I2 = GL2[np^2];

Local I4 = GL2[np^4];

Local I6 = GL2[np^6];

Local I8 = GL2[np^8];

Local I10 = GL2[np^10];

Local I12 = GL2[np^12];

Local I14 = GL2[np^14];

sum  mu,nu,mu1,nu1,mu2,nu2,mu3,nu3,mu4,nu4,
        sigma,N1,N2,N3,N4,N5,N6,
        N7,N8,N9,M1,M2,M3,M4,M5,
        M6,M7,M8,M9,
        alpha,beta,alphap,betap,
        rho,rho1,rho2,rho3,rho4,rho5
```

```
        lambda, lambda1, lambda2
        llambda, llambda1, llambda2, gamma;

*symmetrize vt;

Bracket np,[k^2],prop;

print  ;

.end
```

The calculation of the integrals containing up to 14 powers of momentum is carried out separately for each power of the propagator, since the number of terms involved in each one is exceedingly large. Here we list a program for the calculation of the first integrals. All other integrals are performed similarly, using the results of Appendix B, so we do not list all programs explicitly here.

```
*
*                               I0.form
*
* This program calculates integrals containing the propagator^1
*
CFunctions L1phi,vt,p,Gp,partial,b,I0;
Cfunctions G,k,v0,Z,Z1,Z2,Z3,v0,F2,G2,q,L1phik;
Indices mu,nu,mu1,nu1,mu2,nu2,mu3,nu3,mu4,nu4,
        sigma,N1,N2,N3,N4,N5,N6,
        N7,N8,N9,M1,M2,M3,M4,M5,
        M6,M7,M8,M9,
        alpha,beta,alphap,betap,
        rho,rho1,rho2,rho3,rho4,rho5
        lambda, lambda1, lambda2
        llambda, llambda1, llambda2, gamma;
Symbols n,np,nf,nd,m,kappa,[k^2],[v0^2],Pi,prop,[1+v0^2],[q^2],[1-q^2],e,er;

Local F = I0;

* Integrals containing only the first power of the propagator are
```

```
* stored in the file int0.input
* They are now included in the program

#include int0.input

* Here the value of the integrals is calculated, as in Appendix 2.

id [k^2]^5*prop^3 = (1/(512*Pi*kappa^3))*(256 - 640*[q^2] +
960*[q^2]^2 - 800*[q^2]^3 + 350*[q^2]^4 -
63*[q^2]^5)*[1-q^2]^(-11/2)*(1/e);

id [k^2]^4*prop^3 = 0;

id [k^2]^3*prop^3 = 0;

id [k^2]^3*prop^2 = (1/(32*Pi*kappa^2))*(16 - 24*[q^2] + 18*[q^2]^2 -
5*[q^2]^3)*[1-q^2]^(-7/2)*(1/e);

id [k^2]^2*prop^3 = (1/(16*Pi*m^3))*(8 - 8*[q^2] +
3*[q^2]^2)*[1-q^2]^(-5/2)*(1/er);

id [k^2]^2*prop^2 = 0;

id [k^2]*prop^2 =  (1/(4*Pi*m^2))*(2 - [q^2])*[1-q^2]^(-3/2)*(1/er);


id q(mu?) = v0(mu)*[1+v0^2]^(-1/2);

id v0(mu?)*v0(mu?) = [v0^2];

id [1-q^2]^(n?) = [1+v0^2]^(-n);

id [q^2] = [v0^2]*[1+v0^2]^(-1);

id [v0^2] = [1+v0^2] - 1;

id [1+v0^2]^(n?)*[1+v0^2]^(np?)=[1+v0^2]^(n+np);
```

```
id [1+v0^2]^(n?)*[1+v0^2]^np?=[1+v0^2]^(n+np);

* Here the Z terms are identified

id vt(mu?,nu?)*vt(alpha?,beta?) = Z(mu,nu,alpha,beta);

id vt(mu?)*vt(nu?,alpha?,beta?) = -Z(mu,nu,alpha,beta);

id vt(nu?,alpha?,beta?)*vt(mu?) = -Z(mu,nu,alpha,beta);

id Z(mu?,mu?,nu?,nu?) = Z1;
id Z(mu?,nu?,mu?,nu?) = Z1;
id Z(nu?,mu?,mu?,nu?) = Z1;

id v0(mu?)*v0(nu?)*Z(mu?,nu?,alpha?,alpha?) = Z2;
id v0(mu?)*v0(nu?)*Z(mu?,alpha?,nu?,alpha?) = Z2;
id v0(mu?)*v0(nu?)*Z(mu?,alpha?,alpha?,nu?) = Z2;
id v0(mu?)*v0(nu?)*Z(alpha?,alpha?,mu?,nu?) = Z2;
id v0(mu?)*v0(nu?)*Z(alpha?,mu?,alpha?,nu?) = Z2;
id v0(mu?)*v0(nu?)*Z(alpha?,mu?,nu?,alpha?) = Z2;

id v0(mu?)*v0(nu?)*v0(alpha?)*v0(beta?)*Z(mu?,nu?,alpha?,beta?) = Z3;

id nd=1;

Bracket Z1,Z2,Z3,e,er;

print;

.end
```