



O'Mega (age 10)

Thorsten Ohl

Institute for Theoretical Physics and Astrophysics, Würzburg University

WHIZARD - Event Generation for LHC, ILC, CLIC
November 21-23, 2011
DESY Hamburg

- ▶ A computer program implementing the function

$$(\mathcal{L}, \{\text{incoming}\}, \{\text{outgoing}\}) \mapsto \mathcal{M}(\alpha_1, \dots; p_1, \dots; s_1, \dots)$$

where

- ▶ \mathcal{L} : Lagrangian (or Feynman rules) of a model (SM, MSSM, ...)
- ▶ $\mathcal{M}(\alpha_1, \dots; p_1, \dots; s_1, \dots)$: a function

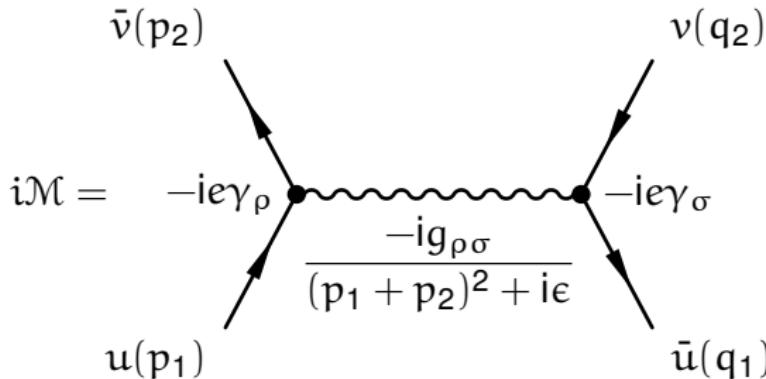
$$\underbrace{\mathbf{R} \times \dots \times \mathbf{R}}_{\text{masses, couplings, ...}} \times \underbrace{\mathbf{V}^+ \times \dots \times \mathbf{V}^+}_{\text{4-momenta (forward light cones)}} \times \underbrace{\mathbf{Z} \times \dots \times \mathbf{Z}}_{\text{helicities, colors}} \rightarrow \underbrace{\mathbf{C}}_{\text{amplitude}}$$

in a form that can be evaluated numerically, typically as C, C++ or Fortran code in that can be compiled and linked to Monte Carlo phase space integrators and generators

- ▶ NB: in some cases only $\mathcal{L} \rightarrow \sum |\mathcal{M}(\alpha_1, \dots; p_1, \dots; s_1, \dots)|^2$ is required. It is often better defined (infrared/collinear cancellations) and sometimes more compact (spin/polarization sums).
- ▶ first robust and usable examples in the early 1990s: CompHEP, FeynArts, Grace, MadGraph, ...



- ▶ for simplicity: $e^+e^- \rightarrow \mu^+\mu^-$ at PETRA (i. e. QED, mostly)
- ▶ just one Feynman diagram



- ▶ analytical expression

$$\begin{aligned} i\mathcal{M} &= \bar{v}(p_2)(-ie\gamma^\rho)u(p_1) \frac{-ig_{\rho\sigma}}{(p_1 + p_2)^2 + i\epsilon} \bar{u}(q_1)(-ie\gamma^\sigma)v(q_2) \\ &= ie^2 \frac{1}{s} [\bar{v}(p_2)\gamma_\rho u(p_1)] [\bar{u}(q_1)\gamma^\rho v(q_2)] \end{aligned}$$

- e.g. command line for O'Mega to compute $e^- e^+ \rightarrow \mu^- \mu^+$ in QED

```
$ omega_QED -scatter "e- e+ -> m- m+"
```

- resulting Fortran95 code (as of O'Mega 0.x)

```
pure function eleposmuamu (k, s) result (amp)
  real(kind=omega_prec), dimension(0::), intent(in) :: k ! 4-momenta k_i
  integer, dimension(:), intent(in) :: s
  complex(kind=omega_prec) :: amp
  type(momentum) :: p1, p2, p3, p4
  type(spinor) :: muo_4, ele_1
  type(conjspinor) :: amu_3, pos_2
  type(vector) :: gam_12
  type(momentum) :: p12
  p1 = - k(:,1) ! incoming e-
  p2 = - k(:,2) ! incoming e+
  p3 = k(:,3) ! outgoing m-
  p4 = k(:,4) ! outgoing m+
  ele_1 = u (mass(11), - p1, s(1)) ! u_{s_1}(k_1)
  pos_2 = vbar (mass(11), - p2, s(2)) ! \bar{v}_{s_2}(k_2)
  amu_3 =ubar (mass(13), p3, s(3)) ! \bar{u}_{s_3}(k_3)
  muo_4 = v (mass(13), p4, s(4)) ! v_{s_4}(k_4)
  p12 = p1 + p2
  gam_12 = pr_feynman(p12, + v_ff(qlep, pos_2, ele_1)) ! (1/s) e\bar{v}(k_2)\gamma_\mu u(k_1)
  amp = 0
  amp = amp + gam_12*( + v_ff(qlep, amu_3, muo_4)) ! (1/s) e\bar{v}(k_2)\gamma_\mu u(k_1) e\bar{u}(k_3)\gamma^\mu v(k_4)
  amp = - amp ! 2 vertices, 1 propagators
end function eleposmuamu
```

(some additional interface routines suppressed)

full disclosure: code generated by O'Mega 2.x:

```
pure subroutine calculate_amplitudes (amp, k)
  complex(kind=default), dimension(:,:,:,:), intent(out) :: amp
  real(kind=default), dimension(0:3,*), intent(in) :: k
  integer, dimension(n_prt) :: s
  integer :: h
  p1 = - k(:,1) ! incoming
  p2 = - k(:,2) ! incoming
  p3 =   k(:,3) ! outgoing
  p4 =   k(:,4) ! outgoing
  p12 = p1 + p2
  amp = 0
  do h = 1, n_hel
    s = table_spin_states(:,h)
    owf_pos_1 = vbar (mass(11), - p1, s(1))
    owf_ele_2 = u (mass(11), - p2, s(2))
    owf_muo_3 = v (mass(13), p3, s(3))
    owf_amu_4 = ubar (mass(13), p4, s(4))
    call compute_fusions_0001 ()           ! help compiler by breaking code into smaller chunks
    call compute_brakets_0001 ()
    amp(1,h,1) = oks_poseleamumuo        ! compute several amplitudes at once
  end do
end subroutine calculate_amplitudes
subroutine compute_fusions_0001 ()
  owf_gam_12 = pr_feynman(p12, + v_ff(qlep,owf_pos_1,owf_ele_2))
end subroutine compute_fusions_0001
subroutine compute_brakets_0001 ()
  oks_poseleamumuo = 0
  oks_poseleamumuo = oks_poseleamumuo + owf_gam_12*( + v_ff(qlep,owf_amu_4,owf_muo_3))
  oks_poseleamumuo = - oks_poseleamumuo ! 2 vertices, 1 propagators
end subroutine compute_brakets_0001
```



The number of tree Feynman diagrams w/ n legs grows like a **factorial**,
 e.g. in ϕ^3 -theory: $F(n) = (2n - 5)!! = (2n - 5) \cdot (2n - 7) \cdot \dots \cdot 3 \cdot 1$

n	$F(n)$	$P(n)$	
4	3	3	computational costs grow beyond all reasonable limits
5	15	10	
6	105	25	gauge cancellations cause loss of precision
7	945	56	
8	10 395	119	Number of possible momenta in tree diagrams grows only exponentially
9	135 135	246	
10	2 027 025	501	
11	34 459 425	1 012	
12	654 729 075	2 035	
13	13 749 310 575	4 082	
14	316 234 143 225	8 177	
15	7 905 853 580 625	16 368	

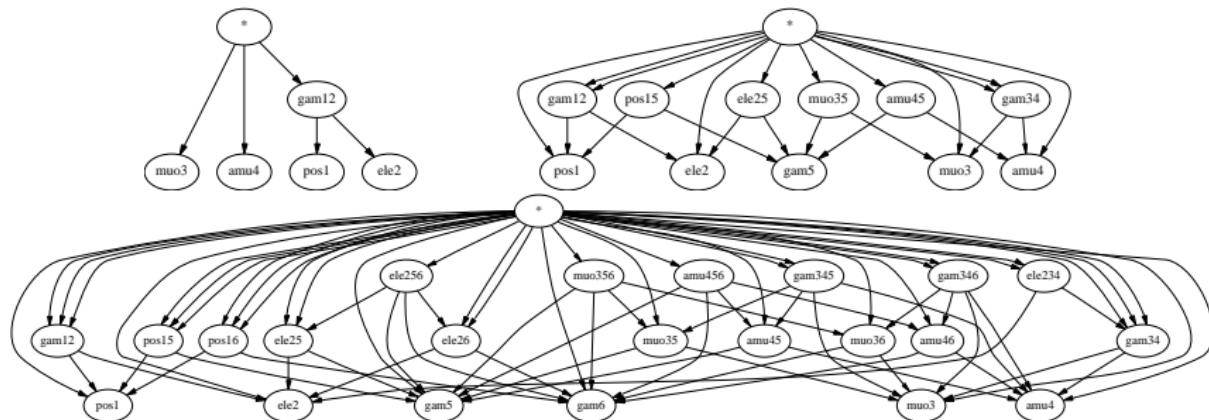
$$P(n) = \frac{2^n - 2}{2} - n = 2^{n-1} - n - 1$$

\therefore Feynman diagrams **redundant** for many external particles!

∴ Replace the forest of tree diagrams by the **Directed Acyclical Graph (DAG)** of the algebraic expression.

$$ab(ab + c) = \begin{array}{c} \times \\ / \quad \backslash \\ a \quad b \end{array} + \begin{array}{c} \times \\ / \quad \backslash \\ a \quad b \end{array} c = \begin{array}{c} \times \\ / \quad \backslash \\ a \quad b \end{array} + \begin{array}{c} \times \\ / \quad \backslash \\ a \quad b \end{array} c$$

- simplest examples: $e^+e^- \rightarrow \mu^+\mu^-$, $e^+e^- \rightarrow \mu^+\mu^-\gamma$ and $e^+e^- \rightarrow \mu^+\mu^-\gamma\gamma$ (only QED)



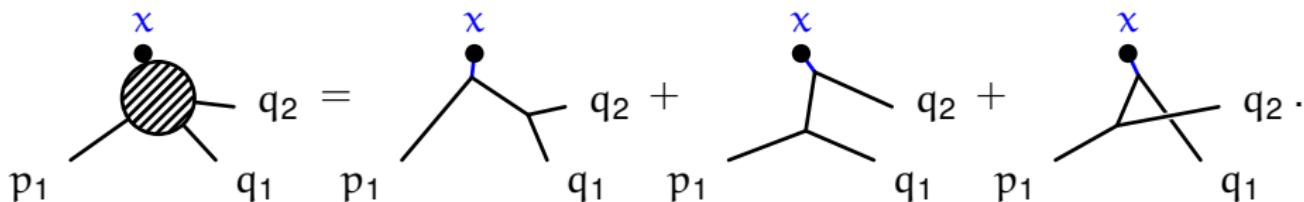
Efficient tree amplitudes

- ▶ Berends-Giele Recursion Relations [Berends, Giele]
 - ▶ manual calculations
- ▶ HELAS [Hagiwara et al.],
 - ▶ manual partial common subexpression elimination
- ▶ Madgraph [Stelzer et al.], AMEGIC++, COMIX [Krauss et al.]:
 - ▶ partial common subexpression elimination
 - ∴ partial elimination of redundancy
- ▶ ALPHA [Caravaglios & Moretti]:
 - ▶ tree level scattering amplitude is Legendre transform of Lagragian
 - ⌚ can be performed numerically, using only $P^*(n)$ independent variables
- ▶ HELAC [Papadopoulos et al.]:
 - ▶ ALPHA algorithm can be reformulated as recursive numerical solution of Schwinger-Dyson equations
- ▶ O'Mega [TO et al.]:
 - ▶ systematic elimination of all redundancies
 - ▶ symbolic, generation of compilable code

One particle off-shell wave functions (**1POWs**) are obtained from by applying the LSZ reduction formula to all but one line:

$$W(x; p_1, \dots, p_n; q_1, \dots, q_m) = \langle \phi(q_1), \dots, \phi(q_m); \text{out} | \Phi(x) | \phi(p_1), \dots, \phi(p_n); \text{in} \rangle .$$

E.g. $\langle \phi(q_1), \phi(q_2); \text{out} | \Phi(x) | \phi(p_1); \text{in} \rangle$ in ϕ^3 -theory at tree level



set of **all** 1POWs at tree level grows **exponentially**, each 1POW can be constructed recursively from other 1POWs at tree level.

There exists a well defined set of **keystones K** that allow to express the sum of Feynman diagrams through **1POWs**:

$$T = \sum_{i=1}^{F(n)} D_i = \sum_{k,l,m=1}^{P(n)} K_{f_k f_l f_m}^3(p_k, p_l, p_m) W_{f_k}(p_k) W_{f_l}(p_l) W_{f_m}(p_m)$$

- ▶ Even for vector particles, the 1POWs are ‘almost’ physical objects and satisfy simple **Ward Identities** in unbroken gauge theories

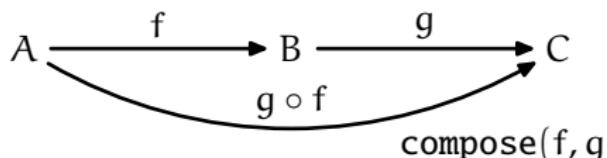
$$\frac{\partial}{\partial x_\mu} \langle \text{out} | A_\mu(x) | \text{in} \rangle_{\text{amp.}} = 0$$

- ▶ and spontaneously gauge theories (R_ξ -gauge)

$$\frac{\partial}{\partial x_\mu} \langle \text{out} | W_\mu(x) | \text{in} \rangle_{\text{amp.}} = \xi_W m_W \langle \text{out} | \phi_W(x) | \text{in} \rangle_{\text{amp.}} .$$

- ⌚ code for matrix elements can optionally be instrumented to check these Ward identities, testing the **consistency** a particular model and the **numerical stability** of expressions (gauge cancellations happen **within** 1POWs).
- ▶ Amplitudes can be continued off-shell:
 - ▶ **Slavnov-Taylor Identities** can be checked numerically by adding **operator insertions** implementing BRS transformations.

- ▶ Modern functional programming languages allow high level construction of code
- ▶ just like in modern mathematics, **functions** are (almost¹) **first class citizens**, can be **arguments** of other functions
- ▶ simplest example: given any pair of functions f and g with $\text{dom}(g) = \text{cod}(f)$, we can form the **composition** $g \circ f$



∴ for all sets A, B, C , “composition” is itself a **function**

$$\circ : (A \rightarrow B) \times (B \rightarrow C) \rightarrow (A \rightarrow C)$$

- ▶ e. g. in O'Caml

```
let compose f g = fun x -> f (g x)
```

defines `compose` as \circ

¹up to undecidability of termination ...

- ▶ State-of-the-art implementations of these concepts offer
 - ▶ type systems that automatically verify that $\text{dom}(g) = \text{cod}(f)$ in all occurrences $g \circ f$
 - ∴ $\text{dom}(f)$ must be defined precisely for f
 - ∴ programs can not crash (except for hardware limitations and bugs in the compiler or runtime system)
 - ▶ nevertheless competitive execution speed (except for problems that can solved by matrix multiplications)
- ▶ CAVEAT: all problems can be mapped to matrix multiplication and subsequently be implemented in low level languages
 - ∴ tedious, no compile time verification, complex memory management
 - ∴ error prone, difficult to extend after a short time
- ▶ recursion more naturally represented by functions $f : A \rightarrow A$

$f, f \circ f, f \circ f \circ f, \dots, f \circ f \circ \dots \circ f, \dots$

and persistent data structures



Slightly simplified Model.T signature that **all** models must implement:

```
module type Model.T =  
  sig  
    type flavor (* all quantum numbers *)  
    val flavor_symbol : flavor -> string  
    val conjugate : flavor -> flavor (* antiparticles *)  
    val lorentz : flavor -> Coupling.lorentz (* spin *)  
    val fermion : flavor -> int (* fermion, boson, antifermion *)  
    val width : flavor -> Coupling.width (* scheme, not value! *)  
    type gauge (* parametrized gauges *)  
    val gauge_symbol : gauge -> string  
    val propagator : flavor -> gauge Coupling.propagator  
    type constant (* coupling constants *)  
    val constant_symbol : constant -> string  
    val fuse2 : flavor -> flavor ->  
      (flavor * constant Coupling.t) list (*  $A_\mu(p_{12}) \leftarrow g\bar{\psi}(p_1)\gamma_\mu\psi(p_2)$  *)  
    val fuse3 : flavor -> flavor -> flavor ->  
      (flavor * constant Coupling.t) list (*  $\Phi(p_{123}) \leftarrow g\phi(p_1)\phi(p_2)\phi(p_3)$  *)  
    val fuse : flavor list -> (flavor * constant Coupling.t) list  
  end
```

- ▶ process specific code with unrolled loops compiled by optimizing compilers can **in principle** run faster than purely **numerical** approaches
 - ಠ **in practice**, optimizing compilers can run out of **memory** and **time** for complicated LHC processes (\rightarrow switch off optimizer)
 - ಠ **in practice**, compiled unrolled code can **itself** become very large and cause many **cache misses**
- ▶ **subsets of diagrams** can be selected based on intermediate states, e. g.
 - ▶ $u\bar{u} \rightarrow Z \rightarrow W^+W^- \rightarrow u\bar{d}d\bar{u}$
`$ omega_SM -scatter "u ubar -> u dbar d ubar" \
-cascade "3+4+5+6~Z && 3+4~W+ && 5+6~W-"`
 - ▶ $u\bar{u} \rightarrow u\bar{u}$ with only t-channel photons
`$ omega_SM -scatter "u ubar -> u ubar" -cascade "1+2~A"`
- ▶ **automatized special effects**: charged particle width, K-matrix, ...
- ▶ **human-readable amplitudes** can be **edited by hand** for special effects: **nonlocal interactions**, **loops**, ...



- :(no concept of unbroken global symmetries
- :(:(no concept of unbroken gauge symmetries
 - ∴ no builtin color ...
- :(architecture sufficiently flexible **and** robust for WHIZARD to add color from the outside by augmenting the model
 1. add a **color flow** label as a new quantum number
 2. add color flow selection rules to the vertices
 3. generate amplitudes for all color flows
 4. add the amplitudes in quadrature
- ▶ made WHIZARD usable for LHC processes
 - :(matrix element generation could take a **veeeery** long time
 - ∴ bad scaling w/ **#particle species** became relevant
 - :(maintenance of **hand-colored models** tedious and error prone



- ▶ construct another model with flavor extended by color flow quantum numbers and fuse2 and fuse3 implementing colored Feynman rules (unique for standard dim4 interactions)

Colorize : functor ($M : \text{Model} \rightarrow \text{Model}$)

- ▶ e. g. coupling of three gluons in the adjoint representation

$$= gf_{a_1 a_2 a_3} C^{\mu_1 \mu_2 \mu_3}(k_1, k_2, k_3)$$

- ▶ same vertex in the color flow basis after applying Colorize:

$$= ig (\delta^{i_1 j_3} \delta^{i_2 j_1} \delta^{i_3 j_2} - \delta^{i_1 j_2} \delta^{i_2 j_3} \delta^{i_3 j_1}) \times C^{\mu_1 \mu_2 \mu_3}(k_1, k_2, k_3)$$

- ∴ model descriptions for automatic matrix element generators / event generators usually ignore approximate flavor symmetries
- ∴ amplitudes for, e. g., d and s quarks are computed separately, though identical at LHC energies, f. a. p. p.
- ⌚ wasteful power law increase of required computing resources for multi-jet and hadron-collider cross sections
- ∴ most elegant approach: describe particles as flavor multiplets with energy dependent flavor symmetry breaking
 - ▶ SM:

$$\{u, d, s, c, b, t\} \rightarrow \left\{ \begin{pmatrix} u \\ c \\ t \end{pmatrix}, \begin{pmatrix} d \\ s \\ b \end{pmatrix} \right\}$$
$$\xrightarrow{E \gg m_t} \left\{ \begin{pmatrix} u \\ c \end{pmatrix}, \begin{pmatrix} d \\ s \\ b \end{pmatrix}, t \right\} \xrightarrow{E \gg m_b} \left\{ \begin{pmatrix} u \\ c \end{pmatrix}, \begin{pmatrix} d \\ s \end{pmatrix}, b, t \right\}$$

- ▶ even more drastic when flavor and gauge group factorize
 - ▶ QCD:

$$\{u, d, s, c, b, t\} \rightarrow \left\{ \begin{pmatrix} u \\ d \\ s \\ c \\ b \\ t \end{pmatrix} \right\} \xrightarrow{E \gg m_t} \left\{ \begin{pmatrix} u \\ d \\ s \\ c \\ b \end{pmatrix}, t \right\} \xrightarrow{E \gg m_b} \left\{ \begin{pmatrix} u \\ d \\ s \\ c \end{pmatrix}, b, t \right\}$$

- ▶ complication: **discrete** property **flavor group** depends on **continuous** parameter(s): **masses, energies, cuts**
- ⌚ proliferation of model files
- ⌚ **functors** to the rescue:
 - ▶ `Restore_Flavor_Symmetry`: `model × multiplets → model'`
 - ▶ `Break_Flavor_Symmetry`: `model × splitting → model'`

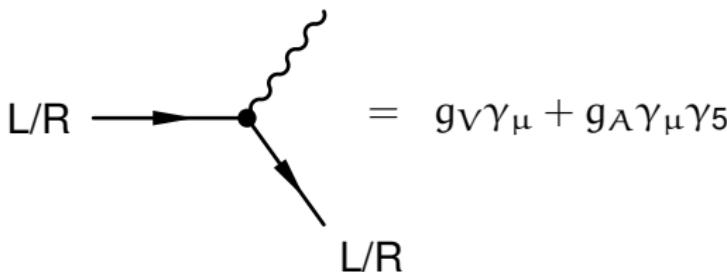
- ▶ $V_{CKM} \neq \mathbf{1}$ leads to a substantial increase of CC interactions

$$3 \times \bar{u}_i W d_i \rightarrow 9 \times \bar{u}_i W d_j$$

(also in SUSY)

- :(explosion of the number of contributions to multi particle matrix elements
- :(additional source of flavor symmetry breaking, spoiling the previous slide
- ▶ way out:
 1. compute scattering amplitudes in the flavor basis, i. e. before CKM mixing
 2. apply CKM mixing to the external states only
- : stay tuned ...

- ∴ helicity of massless fermions **conserved** across **vector** and **axial-vector** couplings, as in the SM



- ∴ approximate helicity selection rules for light quarks
- ∴ again a **continuous** parameter (m_f) controls a **discrete** property
- ⌚ possible proliferation of model files
 - ▶ currently: **numerical heuristic** similar to old MadGraph version
- ⌚ future: functorized switching of fermion masses and analytic helicity selection rules

- ▶ current limitation: **predefined** set of **vertex structures** mapping **Feynman rules** to Fortran subroutines
 - ☺ complete for the SM
 - ☹ extensions require surgery on O'Mega itself (cf. Fabian's Talk)
∴ only a limit set of higher dimensional operators available
 - ☺ infrastructure available, but coding and testing required
- ▶ mid term project: implement **vertex description language close to physics notation** that O'Mega can compile to Fortran
- ▶ another limitation: color flows only implemented for
 - ▶ $3 \otimes 8 \otimes \bar{3}$ (quark/anti-quark/glue)
 - ▶ $8 \otimes 8 \otimes 1$ (glue/glue/higgs)
- ▶ not sufficient for more exotic BSM scenarios
 - ▶ color-sextets, ...: **more than one arrow per line**
 - ▶ $3 \otimes 3 \otimes 3$ couplings: **clashing arrows**
- ☺ theory paper draft exists, implementation forthcoming