NPointFunctions

Uladzimir Khasianevich

Institut für Kern- und Teilchenphysik, TU Dresden

KUTS@DESY 2024

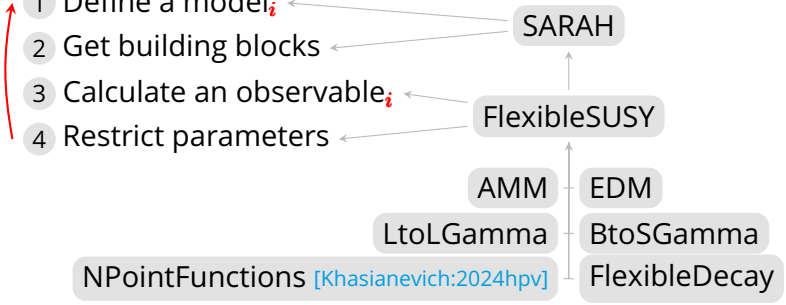
Overview

Structure

1. New essential features in **Part I**
2. Application examples in **Part II**
3. Plans in **Part III**

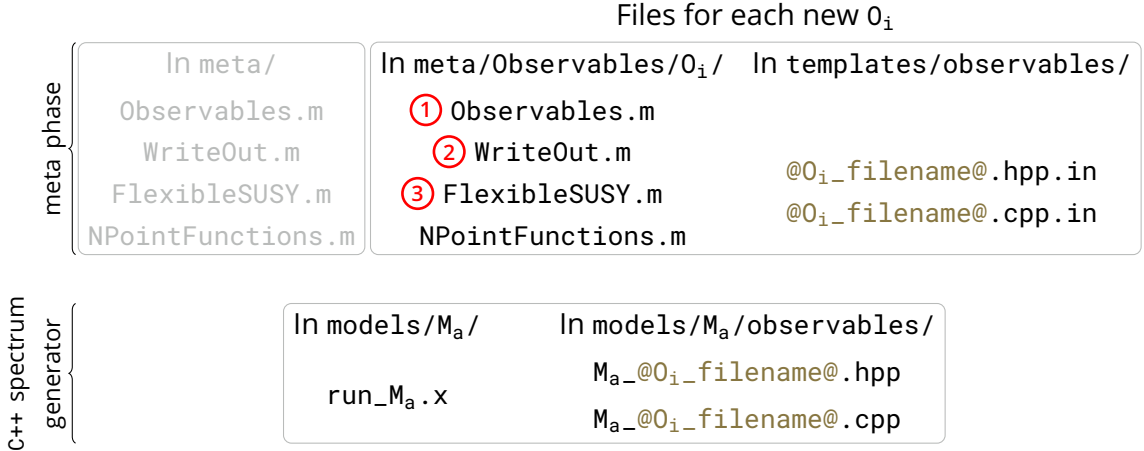
Workflow example (of a phenomenologist)

1. Define a model$_i$
2. Get building blocks
3. Calculate an observable$_i$
4. Restrict parameters



Part I

Creating new observable



Toy example: output two lepton masses

meta/Observables/Mass/Observables.m

```
Observables`DefineObservable[
   FlexibleSUSYObservable`Mass[fermion_[gen_]],
   GetObservableType      -> {2},
   GetObservablePrototype -> "fermion_mass(int gen, auto model, auto qedqcd)"
];
```

meta/Observables/Mass/WriteOut.m

```
WriteOut`WriteObservable[
   "FlexibleSUSYLowEnergy", obs:FlexibleSUSYObservable`Mass[_]
] := "Re(observables." <> Observables`GetObservableName[obs] <> "(0))";

WriteOut`WriteObservable[
   "ExampleLeptonMass", obs:FlexibleSUSYObservable`Mass[_]
] := "Re(observables." <> Observables`GetObservableName[obs] <> "(1))";
```

Toy example: output two lepton masses

meta/Observables/Mass/FlexibleSUSY.m

```
(* Task 1: generate function definition by replacing tokens by concrete C++ code *)
definitions = TextFormatting`ReplaceCXXTokens["
   @type@ @prototype@ {
      return forge<@type@, fields::@fermion@>(gen, model, qedqcd);
   }",
   {"@prototype@" -> Observables`GetObservablePrototype[#], ...}
] &/@ observables;
```

templates/mass.cpp.in

```
template <typename RTYPE, typename FIELD>
auto forge(int idx, ...) { ...
   auto context_mass = context.mass<FIELD>({idx});
   switch (idx) {case 0:  lepton_mass = qedqcd.displayPoleMel(); ...}
   RTYPE res {context_mass, lepton_mass};
   return res;
}
```

Content of new files

meta/Observables/O_i/Observables.m

```
Observables`DefineObservable[
   FlexibleSUSYObservable`Oi[parA_, parB_, ...],
   GetObservableType        -> {1},                          (* @Oi_output_type@ *)
   GetObservablePrototype   -> "calc_parA(int parB, ...)",

   (* Optional *)
   GetObservableName        -> "name_with_parAparB...",      (* @Oi_name@ *)
   CalculateObservable      -> "calc_parA(parB+1, ...)",
   GetObservableNamespace   -> "observable_namespace",       (* @Oi_namespace@ *)
   GetObservableFileName    -> "observable_file",            (* @Oi_filename@ *)
   GetObservableDescription -> "observable: parA, parB"
];
```

Content of new files

meta/Observables/O_i/WriteOut.m

```
WriteOut`WriteObservable["FlexibleSUSYLowEnergy",
   obs:FlexibleSUSYObservable`O_i[parA_, parB_, ...]
] := "Re(observables." <> Observables`GetObservableName[obs] <> "(0))";

WriteOut`WriteObservable["FWCOEF",
   obs:FlexibleSUSYObservable`O_i[parA_, parB_, ...]
] := StringReplace[
   {
      "fermions_1, operator_1, Oalpha_1, Oalphas_1, contributions_1, num_value, \"comment_1\"",
      ...
   },
   {
      "num_value" -> "Re(observables." <> Observables`GetObservableName[obs] <> ")",
      ...
   }
];
```

Content of new files

meta/Observables/O_i/FlexibleSUSY.m

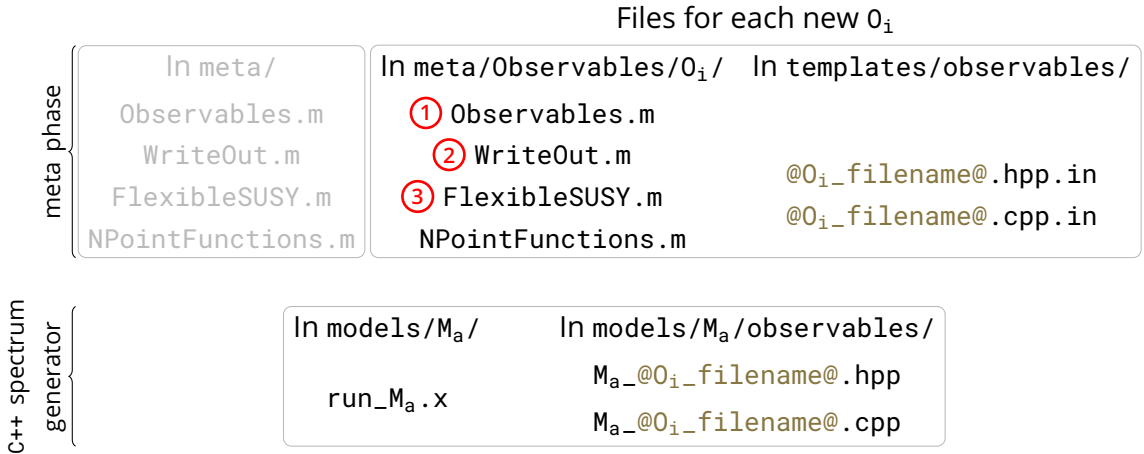
```
FlexibleSUSY`WriteClass[obs:FlexibleSUSYObservable`O_i, allObs_, files_] :=
Module[{ ... },
   If[observables =!= {},
      (* Task 1: filling function definitions *)
   ];

   (* Task 2: filling templates and moving them into models/M_a/observables/ *)

   (* Task 3: returning something to the outside world *)
   {
      "for_outside_usage_i" -> ...,
      ...
   }
];
```

Content of new files

templates/observables/@O$_i$_filename@.cpp.in

```
// Required and auxiliary #include directives

namespace flexiblesusy {
namespace @ModelName@_cxx_diagrams::npointfunctions {
   @npf_definitions@
}

using namespace @ModelName@_cxx_diagrams;
namespace @namespace@ {
   // Auxiliary expressions for the observable
   // Definition of the function that calculates the observable:
   @calculate_definitions@
}
}
```

Creating new observable



Enabling NPointFunctions

meta/Observables/O_i/FlexibleSUSY.m

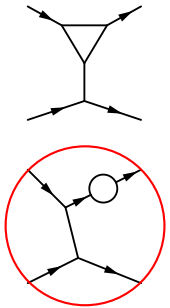
```
(* Task 1: generate function definition by replacing tokens by concrete C++ code *)
npf = NPointFunctions`NPointFunction[
   {field}, {field}, (* Incoming and outgoing particles *)
   ...
   NPointFunctions`Observable -> obs[]
];

AppendTo[definitions,
   TextFormatting`ReplaceCXXTokens["
      @type@ @prototype@ {
         const auto npf = npointfunctions::@name@(model, {gen, gen}, {});
         return {npf[0], npf[1]};
      }",
      {"@type@" -> ...}
   ]
];
```

Settings for Feynman diagrams

Settings

```
topologies[LOOPS]
diagrams[LOOPS, TYPE]
amplitudes[LOOPS, TYPE]
order[]
chains[LOOPS]
...
```

Feynman diagrams

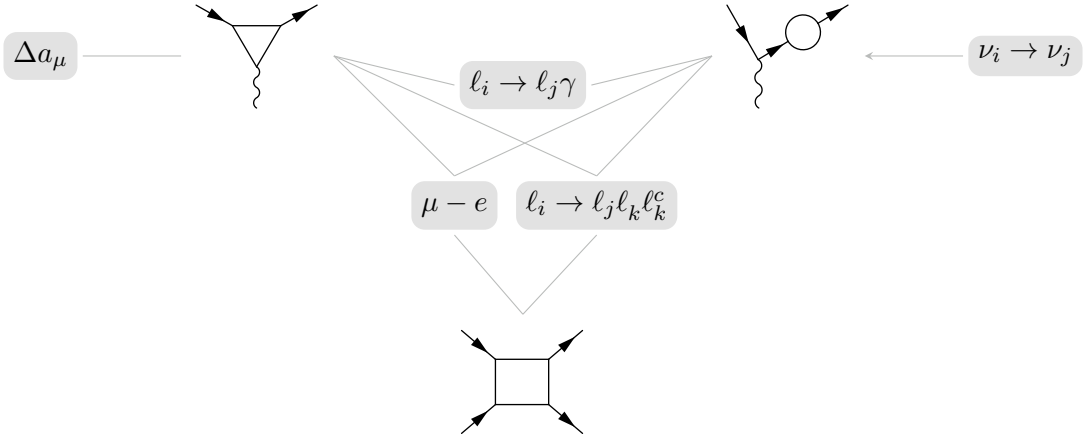


meta/Observables/O_i/NPointFunctions.m

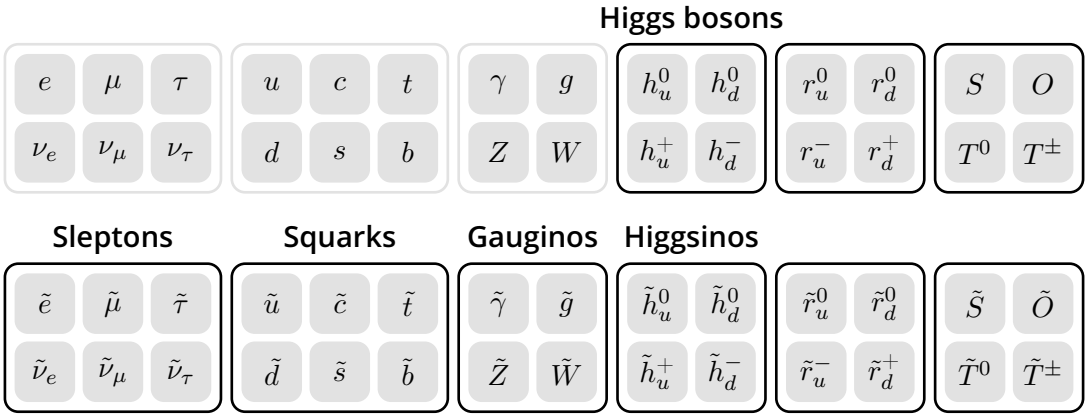
```
topologies[1] = {
   Scalars -> triangleT, Vectors -> outSelfT
};
```

Part II

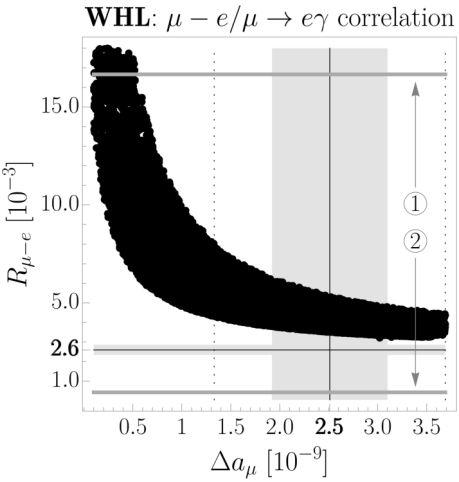
CLFV: Motivation

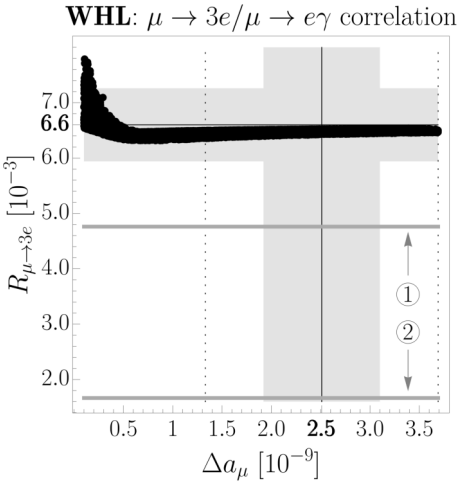


MRSSM: Content

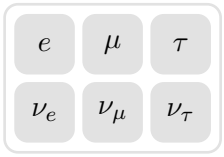


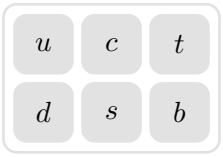
MRSSM: Study example





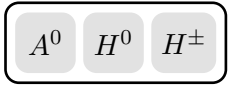
GNM: Content



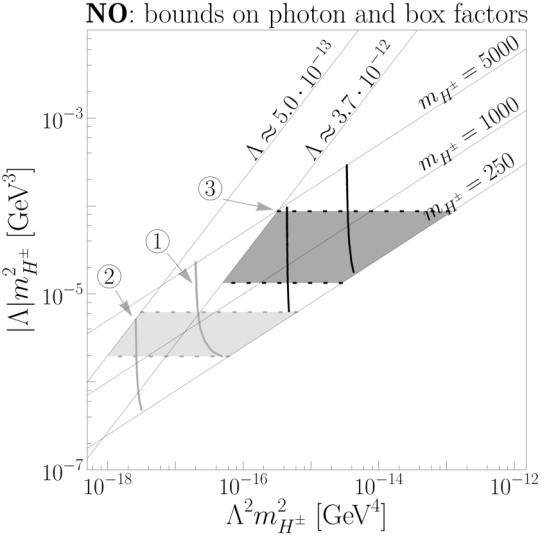


Heavy neutrino

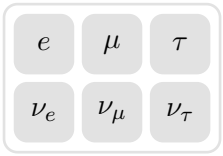
2nd Higgs doublet

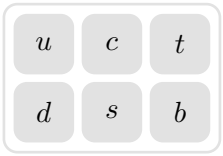


GNM: Study example [Dudenas:2022von, Dudenas:2022xnq]



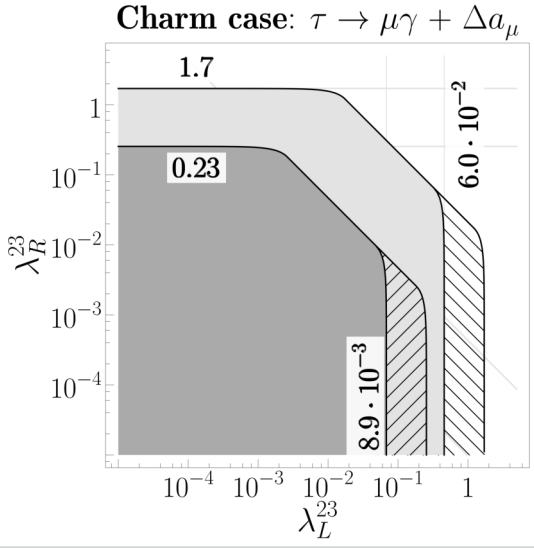
LQ: Content





Scalar leptoquark

LQ: Study example [Khasianevich:2023duu]



Part III

Post-processing

O_i/FlexibleSUSY.m

```
npf = NPointFunctions`ApplySubexpressions[npf];
npf = npf /. {
   SARAH`sum[__, FormCalc`ec[2, l_] FormCalc`ec[3, l_] SARAH`g[__]] :>          "e2e3",
   ...
};
npf = WilsonCoeffs`InterfaceToMatching[npf, {"eps", "e2e3", "e2m3" "e3m2"}];

...

AppendTo[npfDefinitions,
   NPointFunctions`NPFDefinitions[npf, "cpp_name", SARAH`Delta, {"eps", "e2e3", ...}]
];
```

Flavio output

templates/observables/@O$_i$_filename@.cpp.in

```
#include "json.hpp" // nlohmann
// Definition of C9_bsmumu and other Wilson coefficients
nlohmann::json j;
j["eft"]    = "WET";
j["basis"]  = "flavio";
j["scale"]  =
   dynamic_cast<@ModelName@_mass_eigenstates const&>(context.model).get_scale();
j["values"] = {
   {"C9_bsmumu",   {{"Re", Re(C9_bsmumu)},   {"Im", Im(C9_bsmumu)}}},
   // Other Wilson coefficients
};
std::ofstream wc_json("WET_bsmumu.json");
wc_json << std::setw(4) << j << std::endl;
wc_json.close();
```

Conclusions

- A simpler way to add new observables to FlexibleSUSY
- A way to generate C++ code for Feynman diagrams
- Based on [Khasianevich:2024hpv]

Thank you for attention!