LHADA-to-Rivet translator

Philippe Gras

CEA/IRFU - Saclay

May 06, 19

Introduction

The `lhada2rivet` translator is a code generator that produces a `Rivet` routine out of an analysis description written in LHADA

- It was developed as a proof-of-principle for LHADA-to-program-code automatic translation
- Rivet is a framework to describe analysis results to allow comparison with event generator predictions
  - Originally designed for measurements, whose detector effect has been unfolded
  - Extended to searches and non-unfolded results since release 2.5.2
  - Adopted standard of LHC experiment for the SM measurements.

The ADL language:

LHADA

- The LHADA language was defined in Les Houches Physic at Tev 2015 workshop. arXiv:1605.02684
- Loosely defined language and very flexible
- Calculations of quantities used in the analysis described with a link to a plain text documentation and a code example
  - The code can be written in any language the analysis description author considered as "commonly used".
  - No specification on the code interface like type of the parameters
  - No code dependency specification: in principle the code example could depend on any other code (e.g. jet algorithm).
- No formal description of the syntax and grammar

The ADL language (cont'd)

LHADA17

- LHADA specification was complement in arXiv:1803.10379 to defined LHADA17
  - "code examples" $\rightarrow$ code in c++-11 compilable, with interface (like argument types) and dependencies defined in the LHADA17
  - Defines two object types, FourMomentum and LHADAParticle
  - Defines a random generator (required for emulation of efficiencies and resolution)
  - Common object and code library provided in a common repository (stored on github, https://github.com/lhada-hep/lhada)
- Syntax and grammar defined in BNF (Backus-Naur form)

Common libraries and code generators

Code generated from a LHADA17 description does not need to use LHADA object type or LHADA code library

The code generated by `lhada2rivet` uses standard `Rivet` types for particles, jets and four momentum, including experimentally-defined object (like an ATLAS/CMS electron)

Lhada2rivet: what does it do?

- Takes an analysis description written in LHADA17
  - the description should start from objects defined in the LHADA object repository, currently includes generator-level defined particles/jets and common ATLAS/CMS reconstruction-level objects
  - event-count based analysis
- Produces a Rivet routine

Example

- search for dark matter in jet+MET final state (monojet), EPJ C76 (2016) 7 392, that was already implemented in Rivet
- implemented in `LHADA` and `Rivet` code produced with `lhada2rivet`
- Compared with analysis implementation from Rivet routine

Input example (1/4)

```
# arxiv:1605.03814 ATLAS JetMET

info analysis
# Details about experiment
  experiment ATLAS
  collider LHC
  id SUSY-2013-15
  inspireID 1458270
  title Search for squarks and gluinos in final states with jets and missing transverse...
  publication Eur. Phys. J. C(2016) 76: 392
  sqrtS 13.0
  lumi 3.2
  arXiv 1605.03814
  hepdata http://hepdata.cedar.ac.uk/view/ins1304456
  doi 10.1140/epjc/s10052-016-4184-8

function calc_Meff
  arg jets
  arg MET
  code ATLASSUSY1605.03814_functions.h


[...]
```

[...] = file truncated

Input example (2/4)

```
# OBJECT SELECTIONS
object jets
  take external JetAk04-AtlasRun2-00
  select pt > 20
  select |eta| < 2.8

object cleanjets
  take jets
  apply dRJetVeto(col2 = electrons, minDeltaR = 0.2)

object muons
# Muons
  take external Muon-AtlasRun2-00
  select pt > 10
  select |eta| < 2.7
  #select IsolationVarRhoCorr < 0.1
  #select isol(src=tracks, dR=0.4, reliso=true) < 0.1

[...]

# EVENT VARIABLES

variable Meff
  apply calc_Meff(jetsSR = jetsSR, MET = MET)

[...]
```

Input example (3/4)

```
# EVENT SELECTION

cut preselection
# Pre-selection cuts
  select MET.pt > 200
  reject cleanmuons.size > 0
  reject verycleanelectrons.size > 0
  select jetsSR.size >= 2

cut 2jl
  select preselection
  select jetsSR[0].pt > 200
  select jetsSR.size >= 2
  select dPhiMet3j > 0.8
  select jetsSR[1].pt > 200
  select METoversqrtHT > 15
  select Meff > 1200

[...]
```

Input example (3/4)

```
# Results
table results_events
  type events
  columns name obs bkg   dbkg
  entry   2jl  263 283   24
  entry   2jm  191 191   21
  entry   2jt   26  23    4
  entry   4jt    7   4.6  1.1
  entry   5j     7  13.2  2.2
  entry   6jm    4   6.9  1.5
  entry   6jt    3   4.2  1.2

#End of Lhada file
```

Ouput example (1/6)

```
// -*- C++ -*-
#include <iostream>
#include "Rivet/Analysis.hh"
#include "Rivet/AnalysisHandler.hh"
...

namespace Rivet {


  class ATLASSUSY1605_03814: public Analysis {
  public:

    ///Cut flow ids
    enum {k2jt, k6jt, k4jt, k6jm, k5j, k2jm, k2jl} CutFlowIds;
    /// Constructor
    ATLASSUSY1605_03814(): Analysis("ATLASSUSY1605_03814")
    {
        cutflows.addCutflow("2jt", {"MET.pt() > 200",
                                    "!(cleanmuons.size() > 0)",
                                    "!(verycleanelectrons.size() > 0)",
                                    "jetsSR.size() >= 2",
                                    "jetsSR[0].pt() > 200",
                                    "jetsSR.size() >= 2",
                                    "dPhiMet3j > 0.8",
                                    "jetsSR[1].pt() > 200",
                                    "METoversqrtHT > 20",
                                    "Meff > 2000"});

[...]
```

Ouput example (2/6)

```
/// Book histograms and initialise projections before the run
void init() {

  FinalState fs;
  VisibleFinalState visfs(fs);

  FinalState caloFS(Cuts::abseta < 4.8);
  FastJets jetAk04Eta48Proj(caloFS, FastJets::ANTIKT, 0.4);
  declare(jetAk04Eta48Proj, "jetAk04Eta48Proj");
  declare(SmearedJets(jetAk04Eta48Proj, JET_SMEAR_ATLAS_RUN2,
                      JET_BTAG_ATLAS_RUN2_MV2C20), "recoJetAk04");
  PromptFinalState TruthMuonFS(Cuts::abseta < 2.7
                               && Cuts::abspid == PID::MUON,
                               true, true);
  declare(TruthMuonFS, "TruthMuons");
  declare(SmearedParticles(TruthMuonFS, MUON_EFF_ATLAS_RUN2,
                           MUON_SMEAR_ATLAS_RUN2), "preMuons");
  PromptFinalState TruthElectronFS(Cuts::abseta < 2.5
                                   && Cuts::abspid == PID::ELECTRON,
                                   true, true);
  declare(TruthElectronFS, "TruthElectrons");
  declare(SmearedParticles(TruthElectronFS, ELECTRON_EFF_ATLAS_RUN2,
                           ELECTRON_SMEAR_ATLAS_RUN2), "preElectrons");
  FinalState caloFS1(Cuts::abseta < 4.8);
  MissingMomentum TruthMET(caloFS1);
  declare(TruthMET, "TruthMET");
  declare(SmearedMET(TruthMET, MET_SMEAR_ATLAS_RUN2), "MET");
}
[...]
```

Ouput example (3/6)

```
double  calc_aplanarity(const Jets& jets){
  double S12=0., S31=0., S23=0., S11=0., S22=0., S33=0., Stot=0.;
  for(const auto&j : jets)  {
    S11+=j.px()*j.px();
    S12+=j.px()*j.py();
    S22+=j.py()*j.py();
    S23+=j.py()*j.pz();
    S31+=j.pz()*j.px();
    S33+=j.pz()*j.pz();
    Stot+=j.p()*j.p();
  }

  S11=S11/Stot;
  S12=S12/Stot;
  S22=S22/Stot;
  S23=S23/Stot;
  S31=S31/Stot;
  S33=S33/Stot;

  [...]

  double a1 = (Sii-cth)/3.+sth;
  double a2 = (Sii-cth)/3.-sth;
  double a3 = (Sii-cth)/3.+cth;
  double lam3 = 1.5*std::min(std::min(a1,a2),a3);
  return lam3;
}

[...]
```

Ouput example (4/6)

```
bool cut_preselection(double w){
  std::vector<int> cfs = {k2jt, k6jt, k4jt, k6jm, k5j, k2jm, k2jl};
  bool r = true;

  r = r && fillCutFlows(cfs, 0,
                        MET.pt() > 200,
                        w);
  r = r && fillCutFlows(cfs, 1,
                        !(cleanmuons.size()> 0),
                        w);

  r = r && fillCutFlows(cfs, 2,
                        !(verycleanelectrons.size()> 0),
                        w);
  r = r && fillCutFlows(cfs, 3,
                        jetsSR.size() >= 2,
                        w);
  return r;
};

[...]
```

Ouput example (5/6)

```
    /// Perform the per-event analysis
    void analyze(const Event& event) {
      SmearedJets recoJetAk04 = applyProjection<SmearedJets>(event,
                                                              "recoJetAk04");
      Jets preJets = recoJetAk04.jetsByPt();
      jets.clear();
      for(const auto& p: preJets){
        if(p.pt() > 20
          && (p.abseta() < 2.8)){
          jets.push_back(p);
        }
      }

      cleanjets = dRJetVeto(jets, electrons, 0.2);
      ...
      Meff = calc_Meff(jetsSR, MET);

      dPhiMet3j = calc_dPhiMetJets(jetsSR, 3, MET);
      ...
      double w = event.weight();
      cutflows.fillinit(w);
      if(cut_preselection(w)){
        cut_2jl(w);
        cut_2jm(w);
        cut_2jt(w);
        [...]
      }
    }
[...]
```

Ouput example (6/6)

```
    /// Normalise histograms etc., after the run
    void finalize() {

      //Event count in the cut flows is normalized
      //such that in case of a sample whose only
      //the weight sign vary, events with a positive
      //weight are counted as +1 and events with a negative
      //weight as -1.
      double fact = sqrt(numEvents() / handler().sumW2());
      if(isnormal(fact)){
        for(auto& x: cutflows.cfs){
          x.scale(fact);
        }
      }

      std::cout << "Analsyis cut flow:\n"
                << "-----------------\n\n"
                << cutflows << "\n";
    }

[...]
  DECLARE_RIVET_PLUGIN(ATLASSUSY1605_03814);
}
```

Cut flow comparison

	Reference		Lhada+Rivet		
Description	#evt	tot.eff	#evt	tot.eff	Δ/σ
2jl cut-flow	31250.0	100%	31250.0	-	
Pre-sel+MET+pT1	28581.0	91%	28606.0	100%	0.10
Njet	28581.0	91%	28606.0	100%	0.10
Dphi_min(j,MET)	17279.0	55%	17277.0	60%	-0.01
pT2	17051.0	55%	17058.0	99%	0.04
MET/sqrtHT	8910.0	29%	8891.0	52%	-0.14
m_eff(incl)	8909.0	29%	8890.0	100%	-0.14
2jm cut-flow	31250.0	100%	31250.0	-	
Pre-sel+MET+pT1	28466.0	91%	28488.0	100%	0.09
Njet	28466.0	91%	28488.0	100%	0.09
Dphi_min(j,MET)	22900.0	73%	22950.0	81%	0.23
pT2	22900.0	73%	22950.0	100%	0.23
MET/sqrtHT	10728.0	34%	10724.0	47%	-0.03
m_eff(incl)	10621.0	34%	10629.0	99%	0.05
2jt cut-flow	31250.0	100%	31250.0	-	
Pre-sel+MET+pT1	28581.0	91%	28606.0	100%	0.10
Njet	28581.0	91%	28606.0	100%	0.10
Dphi_min(j,MET)	17279.0	55%	17277.0	60%	-0.01
pT2	17051.0	55%	17058.0	99%	0.04
MET/sqrtHT	5073.0	16%	5082.0	30%	0.09
Pass m_eff(incl)	4852.0	16%	4861.0	96%	0.09

Conclusions

- Proof-of-concept for automatic ADL interpretation and code generation was implemented
- Translator used on a concrete example

- It was demonstrated that automatic generation of recasting code from `LHADA17` can be done