LHADA, from an analysis description to the result reinterpretation

Philippe Gras

CEA/IRFU - Saclay

May 13, 18

Introduction

- Lhada was introduced in Les Houches (LH) 2015 Physics at TeV workshop as a possible standard to describe LHC analysis
- Latest developments focused on automatic code generation
- Question we will address in this talk:

> Can reinterpretation code can reliably* be generated from an analysis description in a simplified language like Lhada?

(*) Reliably means:

- the code generator will understand any valid input file;
- the generated code compiles and does what is expected.

Lhada

- Lhada aims to define a standard to describe analyses
- It stands for Les Houches Analysis Description Accord. It consists of:
  - A description of the requirements of the accord: Contrib. 16 of LH 2015, arXiv:1605.02684
  - A proposal for the accord, described in Contrib. 17 of LH 2015

$\rightarrow$a work in progress

- Lhada is not limited to result reinterpretation: five use cases are listed in Contrib. 16
- In the following "Lhada" will refer to the Contrib. 17 proposal
  - Principle: cuts are described in a simplified language, more complex algorithms are described in programming language.

Machine interpretation of Lhada language

- Lhada is loosely defined, which makes it very flexible
  - functions defined with a reference to a paper or document and an example code in a "commonly-used programming language", where commonly-used is not defined.
  - *external* object, like reconstructed object, are defined by a reference to a paper or documentation
- It is impossible to write a machine-interpreter that supports this flexibility

Lhada is well suited for human reading

- Lhada 2017 introduced in LH 2017 (arXiv:1803.10379)
  - Result-interpretation oriented
  - Based on Lhada: Lhada with extra rules
  - **Designed to be unambiguous and be machine interpretable**

Lhada17 is designed for human and **machine** reading

What makes Lhada 2017 specific?

- Defines the programming language (c++) and code provided with the Lhada are not just example codes, but code which can used by the interpreter, thanks to few simple rules
- *externals* objects come from a common library of object definitions
- Library of common functions provided for convenience (limited, but intended be extended)
- Syntax of Lhada 2017 rigorously defined using computing standards (BNF)

Philosophy: a well-defined language that tools will fully support

Analysis description example: SUSY search with jets+MET

- Analysis: Search for squarks and gluinos in final states with jets and missing transverse momentum at $\sqrt{s} = 13\,\text{TeV}$ with the ATLAS detector, doi:10.1140/epjc/s10052-016-4184-8
- Preselection common to all signal regions (SR)
  - MET $> 200\,\text{GeV}$
  - Veto on muons and electrons
  - At least two jets
- Remaining selection in the following, in Lhada format

Analysis description example: selection description

Lhada file written by S. Sekmen and P. Gras available in http://cern.ch/go/Sj6V

```
cut preselection
# Pre-selection cuts
  select MET.pt > 200
  reject cleanmuons.size > 0
  reject verycleanelectrons.size > 0
  select jetsSR.size >= 2
   select jetsSR[1].pt > 50
cut 2jl
# Signal region 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
```

```
cut 2jm
#Signal region 2jm
  select preselection
  select jetsSR[0].pt > 300
  select jetsSR.size >= 2
  select dPhiMet3j > 0.4

  select METoversqrtHT > 15
  select Meff > 1600

cut 2jt
# Signal region 2jt
  select preselection
  select jetsSR[0].pt > 200
  select jetsSR.size >= 2
  select dPhiMet3j > 0.8
  select jetsSR[1].pt > 200
  select METoversqrtHT > 20
  select Meff > 2000
```

Analysis description example: selection description (con't)

```
cut 4jt
#Signal region 4jt
  select preselection
  select jetsSR[0].pt > 200
  select jetsSR.size >= 4
  select dPhiMet3j > 0.4
  select dPhiMetAllJets > 0.2

cut 5j
#Signal region 5j
  select preselection
  select jetsSR[0].pt > 200
  select jetsSR.size >= 5
  select dPhiMet3j > 0.4
  select dPhiMetAllJets > 0.2
  select jetsSR[4].pt > 50
  select aplanarity > 0.04
  select METoverMeff5j > 0.25
  select Meff > 1600
  select aplanarity > 0.04
  select METoverMeff6j > 0.2
  select Meff > 2000
```

```
cut 6jm
  select preselection
  select jetsSR[0].pt > 200
  select jetsSR.size >= 6
  select dPhiMet3j > 0.4
  select dPhiMetAllJets > 0.2
  select jetsSR[5].pt > 50
  select aplanarity > 0.04
  select METoverMeff6j > 0.25
  select Meff > 1600

cut 6jt
  select preselection
  select jetsSR[0].pt > 200
  select jetsSR.size >= 6
  select dPhiMet3j > 0.4
  select dPhiMetAllJets > 0.2
  select jetsSR[3].pt > 100
  select jetsSR[5].pt > 50
```

Analysis description example: object description

```
object jets
  take external JetAk04-AtlasRun2-00      ← Lhada 2017 specific
  select pt > 20
  select |eta| < 2.8

object electrons
  take external Electron-AtlasRun2-00
  select pt > 10
  select |eta| < 2.47

object cleanjets
  take jets
  apply dRJetVeto(col2 = electrons,     ← use of a c++ function
                  minDeltaR = 0.2)
object jetsSR
  take cleanjets
  select pt > 50

object MET
  take external Met-AtlasRun2-00
```

Analysis description example: object description (con't)

```
object muons
  take external Muon-AtlasRun2-00
  select pt > 10
  select |eta| < 2.7

object cleanmuons
  take muons
  apply dRPartVeto(col2 = cleanjets, minDeltaR = 0.4)

object cleanelectrons
   ctake electrons
  apply dRPartVeto(col2 = cleanjets, minDeltaR = 0.4)

object verycleanelectrons
  take cleanelectrons
  apply unravelEl()
```

Functions in Lhada 2017

- Algorithm that goes beyond a cut are described with a function implemented in c++ like in previous `cleanjets` example
- Lhada 2017 defines the three variable types LhadaParticle, LhadaJet, and FourMomentum together with std::vector of these types
  - note: a code generator that reads Lhada can use different types for particle and four-momentum in the generated code
- Functions are provided in an accompanying file which must be compilable
- The code can depend only on a restricted set of libraries (currently c++ and the Lhada tool library)

Function example

In the Lhada file:

```
function unravelEl
  # Filter an electron collection by requiring a mininum of a 0.05 distance
  # in the \eta,\phi plane between the leptons. In case of multiple lepton
  # within this distance the first in the collection (pt ordered)
  # is kept and others are dropped.
  arg electrons #collection to filter
  code ATLASSUSY1605.03814_functions.h
```

In the `ATLASSUSY1605.03814_functions.h` file:

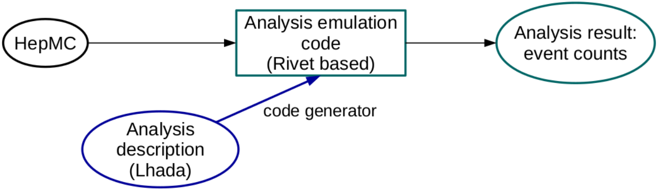
```
std::vector<LhadaParticle> unravelEl(const std::vector<LhadaParticle
    >& el){
  const int n = el.size();
  const double minDeltaR = 0.05;
  std::vector<LhadaParticle> r;
  r.reserve(n);
  for(int i = n - 1; i >= 0; --i){
    bool veto = false;
    for(int j = i - 1; j >= 0; --j){
      veto = veto || (deltaR(el[j], el[i]) < minDeltaR);
    }
    if(!veto){
      r.push_back(el[i]);
    }
  }
  return r;
}
```

Analysis results

Event counts in SR (and optionally control regions) are provided in the Lhada file

```
# 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
```

Generation of Rivet code



lhada2rivet is a tool written in python that generates a Rivet analysis from an analysis description written in Lhada 2017

- the accompanying c++-code is validated before the code generation
  - ease debugging by isolating problem in user's code from the possible ones in the generated code
- Particles and jets are implemented using Rivet specific objects
- At prototype level: not sufficiently tested yet to be smoothly used by non-developers

Validation

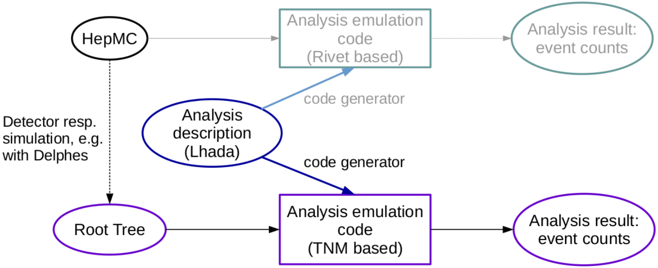
- Cutflow compared with the official Rivet routine from the release 2.6.0

	Reference		Lhada+Rivet		
Description	#evt	tot.eff	#evt	tot.eff	$\Delta/\sqrt{N}$
2jl cut-flow	31250	100%	31250	-	
Pre-sel+MET+pT1	28581	91%	28606	92%	0.10
Njet	28581	91%	28606	92%	0.10
Dphi_min(j,MET)	17279	55%	17277	55%	-0.01
pT2	17051	55%	17058	55%	0.04
MET/sqrtHT	8910	29%	8891	28%	-0.14
m_eff(incl)	8909	29%	8890	28%	-0.14
2jm cut-flow	31250	100%	31250	-	
Pre-sel+MET+pT1	28466	91%	28488	91%	0.09
Njet	28466	91%	28488	91%	0.09
Dphi_min(j,MET)	22900	73%	22950	73%	0.23
pT2	22900	73%	22950	73%	0.23
MET/sqrtHT	10728	34%	10724	34%	-0.03
m_eff(incl)	10621	34%	10629	34%	0.05
2jt cut-flow	31250	100%	31250	-	
Pre-sel+MET+pT1	28581	91%	28606	92%	0.10
Njet	28581	91%	28606	92%	0.10
Dphi_min(j,MET)	17279	55%	17277	55%	-0.01
pT2	17051	55%	17058	55%	0.04
MET/sqrtHT	5073	16%	5082	16%	0.09
Pass m_eff(incl)	4852	16%	4861	16%	0.09
4jt cut-flow	31250	100%	31250	-	
Pre-sel+MET+pT1	28581	91%	28606	92%	0.10
Njet	27317	87%	27359	88%	0.18
Dphi_min(j,MET)	18911	61%	18936	61%	0.13
pT2	18904	60%	18932	61%	0.14
pT4	16731	54%	16755	54%	0.13
Aplanarity	11866	38%	11897	38%	0.20
MET/m_eff(Nj)	8381	27%	8400	27%	0.15
m_eff(incl)	7231	23%	7224	23%	-0.06

	Refernce		Lhada+Rivet		
Description	#evt	tot.eff	#evt	tot.eff	$\Delta/\sqrt{N}$
5j cut-flow	31250	100%	31250	-	
Pre-sel+MET+pT1	28581	91%	28606	92%	0.10
Njet	21253	68%	21270	68%	0.08
Dphi_min(j,MET)	14296	46%	14299	46%	0.02
pT2	14291	46%	14296	46%	0.03
pT4	13346	43%	13344	43%	-0.01
Aplanarity	9864	32%	9865	32%	0.01
MET/m_eff(Nj)	4699	15%	4715	15%	0.16
m_eff(incl)	4657	15%	4673	15%	0.17
6jm cut-flow	31250	100%	31250	-	
Pre-sel+MET+pT1	28581	91%	28606	92%	0.10
Njet	13349	43%	13331	43%	-0.11
Dphi_min(j,MET)	8614	28%	8594	28%	-0.15
pT2	8613	28%	8593	27%	-0.15
pT4	8313	27%	8282	27%	-0.24
Aplanarity	6470	21%	6465	21%	-0.04
MET/m_eff(Nj)	2781	9%	2751	9%	-0.40
m_eff(incl)	2764	9%	2738	9%	-0.35
6jt cut-flow	31250	100%	31250	-	
Pre-sel+MET+pT1	28581	91%	28606	92%	0.10
Njet	13349	43%	13331	43%	-0.11
Dphi_min(j,MET)	8614	28%	8594	28%	-0.15
pT2	8613	28%	8593	27%	-0.15
pT4	8313	27%	8282	27%	-0.24
Aplanarity	6470	21%	6465	21%	-0.04
MET/m_eff(Nj)	4018	13%	4002	13%	-0.18
m_eff(incl)	3805	12%	3805	12%	0

Another code generator: Lhada2tnm

Sezen Sekmen, Harrison Prosper



- Based on genuine Lhada,
  - However, it makes assumptions similar to the rules introduced in Lhada2017: code in c++, define the c++ class to be used for particles and four-momentum (TLorentzVector).
- Produces code for the TheNupleMaker (TNM) framework which is based on ROOT ntuples.
- Use a plugin mechanism to read the input ntuple. Delphes ntuple reader provided.

Another code generator: Lhada2tnm (cont'd)

Sezen Sekmen, Harrison Prosper

Usage

- Can be used for any experimental or phenomenological analysis which uses simple ROOT ntuples.

Lhada reader properties

- does not require specific order of the definition blocks
- handle automatically name collisions the Lhada file can contain

Another code generator: Lhada2tnm (cont'd)

Sezen Sekmen, Harrison Prosper

Status and plans

- Being tested with the 2 analyses used in the LH 2017 Contrib 21 exercise
  - Validation result will be available very soon.
  - `https://github.com/lhada-hep/lhada/tree/master/lhada2tnm`
- Next steps:
  - Further diagnostic tools (work in progress)
  - More complete analyses to further test and improve lhada2tnm
  - Test the usage in experiment analysis development (other usage than reinterpretation)

Comparison with other approaches

Experiments provides the information, the recasters implement the reinterpretion code

- In this approach, the genuine Lhada is useful to describe unambiguously the analysis
- Pros: easier for the experiments
- Cons: difficult validation. Mode difficult for rescasters

Experiment describe their analyses in a Lhada 2017 like language, used to generate code

- Pros: analysis emulation can be validated by the experiments. Easier for the recasters. Code can potentially be generated for the different reinterpretation frameworks.
- Disadvantage: heavier for analysis authors

Experiment implement the analysis emulations in one of the available frameworks

- Pros: validated by analysis authors. Easier for the recasters.
- Cons: heavier for analysis authors.

Conclusions

Answer to the introduction question:

- Reinterpretation code can reliabl be generated from an analysis description in a simplified language like Lhada.

Next steps:

- Understand if, for the analysis authors, the generated code approach is preferred to direct code writing
  - Continuation of work on lhada2rivet to go from a prototype to a production tool will depend on this

Note for lhada2rivet

- The prototype was developed to test the Lhada concept
- Opened to use a different language
- Can easily be extended to other frameworks (MadAnalysis, CheckMate,...)

Backup: Lhada analysis use cases

- Analysis preservation
- Analysis design
- Analysis review and communication
- Interpretation studies and analysis reimplementation
- Comparison of analyses