Effective collider data analysis with Analysis Description Language and CutLang

Harrison Prosper (Florida State U.)
Sezen Sekmen (Kyungpook Nat. U.)
Gökhan Ünel (U. of California, Irvine) and the ADL/CutLang team

PHENO 2021
24-27 May 2021, Online
Welcome to the LHC physics analysis jungle

- Inclusive analyses with hundreds of selection regions
- Overlaps between different analyses?
- Multiple analyses exploring similar final states
- Is my control region your signal region???
- Many alternative definitions for analysis objects
- Many variables, ambiguous definitions

...time to get better organized to work more efficiently!
We perform collider analyses for experimental or phenomenological studies.

Analyses are traditionally performed using analysis software frameworks:

- Frameworks based on general purpose languages like C++ / Python.
- Physics content and technical operations are intertwined and handled together.
- Hard to read, maintain and communicate.

Could there be an alternative way?

- Allow more direct interaction with data and
- Decouple the physics information from purely technical tasks?
Analysis Description Language (ADL) is a domain specific and declarative language that describes the physics content of a collider analysis in a standard and unambiguous way.

- **Domain specific:** Customized to express analysis-specific concepts. Reflects conceptual reasoning of particle physicists.
- **Declarative:** Tells what to do but not how to do it.
- **Designed for everyone:** Experimentalists, phenomenologists, other enthusiasts…

ADL is a language, independent of software frameworks:

→ Any framework recognizing ADL can run the analysis.
  - Focus directly on physics, not on programming.
  - Communicate analyses easily between groups, exp & pheno, etc.
  - Adapt analyses easily between exp & pheno.

Earlier HEP physics formats/languages have been very useful:

- **SUSY Les Houches Accord** and **Les Houches Event Accord**
• **Event processing:** Priority focus.

  - **Input:** event content
  - **Simple and composite object definitions** (jets, muons, Ws, RPV stops, …)
  - **Event variable definitions** ($M_{T2}$, angular variables, BDTs…)
  - **Event selection definitions** (signal, control, validation regions, …)

  **Event processing…**

  - **Output:** event selection

• **Analysis results, i.e. counts and uncertainties:** Available

• **Histogramming:** Partially available.

• **Systematic uncertainties:** To be within the scope. Work in progress.

• **Operations with selected events, e.g. background estimation, scale factor derivation:** Very versatile. Not within the scope yet.
The ADL construct

ADL consists of

• a plain text ADL file describing the analysis algorithm using an easy-to-read DSL with clear syntax rules.

• a library of self-contained functions encapsulating variables that are non-trivial to express with the ADL syntax (e.g. MT2, ML algorithms). Internal or external (user) functions.

• ADL file consists of blocks separating object, variable and event selection definitions. Blocks have a keyword-expression structure.

• keywords specify analysis concepts and operations.

  blocktype blockname
  keyword1 value1
  keyword1 value2
  keyword3 value3 # comment

• Syntax includes mathematical and logical operations, comparison and optimization operators, reducers, 4-vector algebra and HEP-specific functions (dφ, dR, …). See backup.

ADL syntax: example

Objects

# AK8 jets
object AK8jets
take FatJet
select pt > 200
select abs(eta) < 2.4

# mass-tagged jets
object WjetsMassTag
take AK8jets
select msoftdrop [ ] 65 105

# W-tagged jets
object Wjets
take WjetsMassTag
select tau2 / tau1 <= 0.4

Variable definitions

define MR = fMR(megajets)
define Rsq = sqrt(fMTR(megajets, met) / MR)
define METI = MET + leptonsVeto[0]
define RsqI = sqrt(fMTR(megajets, METI) / MR)
define MT = fMT(leptonsVeto[0], MET)
define MII = fMI([leptonsTight[0], leptonsTight[1]])
define dphimegajets = dPhi(megajets[0], megajets[1])

Event selection regions

# preselection region
region preselection
select size(AK4jets) >= 3
select size(AK8jets) >= 1
select MR > 800
select Rsq > 0.08

# control region for tt+jets
region ttjetsCR
select preselection
select size(leptonsVeto) == 0
select size(Wjets) == 1
select dphimegajets < 2.8
select MT > 100
select size(bjetsLoose) >= 1
bin MR [ ] 800 1000 and Rsq [ ] 0.08 0.1
bin MR [ ] 800 1000 and Rsq [ ] 0.1 0.2
bin ...

From CMS SUSY razor analysis CMS-SUS-16-017
Full implementation link
Running analyses with ADL

Experimental / phenomenology analysis model with ADLs

- ADL file
- self-contained functions for complex variables
- events with any format

- Cutflows, counts
- Histograms
- Selected events
- Other results

- adl2tnm (transpiler)
- generic c++ analysis code
- CutLang (runtime interpreter)

transpiler / interpreter for any exp/pheno analysis framework (earlier prototypes exist for Rivet and Checkmate)
Running analyses with ADL: adl2tnm

- Python transpiler converting ADL to C++ code.
- C++ code executed within the generic ntuple analysis framework TNM (TheNtupleMaker). Only depends on ROOT.
- Can work with any simple ntuple format. Automatically incorporates the input event format into the C++ code:
  
  ADL + input ntuple ➔ adl2tnm.py ➔ C++ analysis code ➔ compile & run

- Assumes that a standard extensible type is available to model all analysis objects. Uses adapters to translate input to standard types.
- Can be used for experimental or phenomenological analyses.
- Currently moving from proof of principle to the use of formal grammar building and parsing.

adl2tnm Github: https://github.com/hbprosper/adl2tnm
CutLang runtime interpreter:

- **No compilation.** Directly runs the ADL file on events.
- CutLang itself is written in **C++,** works in any modern **Unix** environment.
- Based on **ROOT** classes for Lorentz vector operations and histograms.
- ADL parsing by **Lex & Yacc.**

CutLang framework: interpreter + tools

- Input events via **ROOT** files.
- **multiple input formats:** Delphes, CMS NanoAOD, ATLAS/CMS Open Data, LVL0, FCC. More can be easily added.
- All event types converted into **predefined particle object types.** —> can run the same ADL file on different input types.
- Includes **many internal functions.**
- Output in **ROOT** files: ADL file, cutflows, bins and histograms for each region in a separate **TDirectory.**
- Tools available for **parallel processing.**

CutLang Github: [https://github.com/unelg/CutLang](https://github.com/unelg/CutLang)
Physics with ADL

Designing new analyses:
• Experimental analyses:
  • 2 ATLAS EXO analyses ongoing with CutLang
• Phenomenology studies:
  • E6 isosinglet quarks at HL-LHC & FCC w/ CutLang (Eur Phys J C 81, 214 (2021))
• Analysis of LHC Open Data:
  • Demo at CMS Open Data workshop: [link](#)
  • An example analysis implemented: [link](#)
  • Analysis optimization via differentiable programming (under development).

Using existing analyses:
ADL analysis database with ~15 LHC analyses: [https://github.com/ADL4HEP/ADLLHCanalyses](https://github.com/ADL4HEP/ADLLHCanalyses) (more being implemented)
• Reinterpretation studies:
  • Integrating ADL into the SModelS framework
• Analysis queries, comparisons, combinations:
  • Automated tools under development
• Long term analysis preservation
Making this a community effort

  - Resulted in prototype language LHADA (Les Houches Analysis Description Accord)
- LHADA workshop, Grenoble, 25-26 Feb 2016
- LHADA workshop, CERN, 16-18 Nov 2016 (link)
- Discussions and activities within HSF data analysis WG and IRIS-HEP.
- 1st dedicated Analysis Description Languages for the LHC workshop (with experimentalists, phenomenologists, computing experts), Fermilab LPC, 6-8 May 2019 (link)
- ADL/CutLang used for training for beginner students with no programming experience:
  - 1st Data Analysis School with ADL+CutLang (3-7 Feb 2020), Istanbul (link, proceedings link)
  - 26th Vietnam School of Physics (VSOP) (Dec 2020) (link)
What is next?

ADL is a promising and feasible approach for practical collider analysis for experiment and phenomenology. ADL, CutLang and adl2tnm are under constant development!

The language

• Develop ADL into a domain-complete language.
• In particular, add systematic uncertainties.

Tools:

• Unify the transpiler and runtime interpreter through a modern set of tools.
• Further automatize incorporating external functions and new input types.
• Implement static analysis & differentiable programming.

Physics:

• Enlarge the ADL analysis database with new analyses.
• Test ADL & tools in many experimental & pheno studies and get feedback.
Extra slides
ADL core syntax: blocks, keywords, operators

<table>
<thead>
<tr>
<th>Block purpose</th>
<th>Block keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>object definition blocks</td>
<td>object</td>
</tr>
<tr>
<td>event selection blocks</td>
<td>region</td>
</tr>
<tr>
<td>analysis information</td>
<td>info</td>
</tr>
<tr>
<td>tables of results, etc.</td>
<td>table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keyword purpose</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>define variables, constants</td>
<td>define</td>
</tr>
<tr>
<td>select object or event</td>
<td>select</td>
</tr>
<tr>
<td>reject object or event</td>
<td>reject</td>
</tr>
<tr>
<td>define the mother object</td>
<td>take</td>
</tr>
<tr>
<td>define histograms</td>
<td>histo</td>
</tr>
<tr>
<td>applies object/event weights</td>
<td>weight</td>
</tr>
<tr>
<td>bins events in regions</td>
<td>bin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison operators</td>
<td>&gt; &lt; =&gt; =&lt; == !=</td>
</tr>
<tr>
<td></td>
<td>[] (include)</td>
</tr>
<tr>
<td>Mathematical operators</td>
<td>+ - * / ^</td>
</tr>
<tr>
<td>Logical operators</td>
<td>and or not</td>
</tr>
<tr>
<td>Ternary operator</td>
<td>condition ? truecase : falsecase</td>
</tr>
<tr>
<td>Optimization operators</td>
<td>~= (closest to)</td>
</tr>
<tr>
<td></td>
<td>~! (furthest from)</td>
</tr>
<tr>
<td>Lorentz vector addition</td>
<td>LV1 + LV2</td>
</tr>
<tr>
<td></td>
<td>LV1 LV2</td>
</tr>
</tbody>
</table>

ADL syntax rules: [https://twiki.cern.ch/twiki/bin/view/LHCPhysics/ADL](https://twiki.cern.ch/twiki/bin/view/LHCPhysics/ADL)
ADL core syntax: functions

Standard/internal functions: Sufficiently generic math and HEP operations would be a part of the language and any tool that interprets it.
- Math functions: abs(), sqrt(), sin(), cos(), tan(), log(), …
- Collection reducers: size(), sum(), min(), max(), any(), all(), …
- HEP-specific functions: dR(), dphi(), deta(), m(), …
- Object and collection handling: sort, comb(), union(), …

External/user functions: Variables that cannot be expressed using the available operators or standard functions would be encapsulated in self-contained functions that would be addressed from the ADL file.
- Variables with non-trivial algorithms: $M_{T2}$, aplanarity, razor variables, …
- Non-analytic variables: Object/trigger efficiencies, variables/efficiencies computed with ML, …

ADL syntax rules: https://twiki.cern.ch/twiki/bin/view/LHCPhysics/ADL