Declarative interfaces for HEP data analysis: FuncADL and ADL/CutLang

FuncADL:

Mason Proffitt *(University of Washington)*
Gordon Watts *(University of Washington)*

ADL/CutLang:

Changgi Huh *(Kyungpook National University)*
Harry Prosper *(Florida State University)*
Sezen Sekmen *(Kyungpook National University)*
Burak Şen *(Middle East Technical University)*
Gokhan Unel *(University of California Irvine)*

2021-12-01
ACAT 2021

Analysis Description Languages (ADLs)

- What is an ADL?
  - A domain-specific language (DSL) sufficient to completely specify a collider physics analysis
- Why are they useful?
  - Unambiguous: not subject to the imprecision of natural language as used in papers
  - Declarative: emphasis is on the physics content, not algorithm design
    - Abstracts away details that are not essential to the analysis
    - Easier to read, write, review, and reinterpret
    - Optimization can be decoupled from the analysis description
- How can they be implemented?
  - DSLs can be divided into two categories:
    - Internal (or embedded): exists within a host language, basically a highly focused API
    - External: an independent language with its own interpreter or compiler

FuncADL

This work was supported by the National Science Foundation under Cooperative Agreement OAC-1836650.

Motivation

- Query languages:
  - Database management systems help to address, among other issues[1]:
    - data redundancy
    - data independence
  - A key aspect of database management is query languages, such as SQL
- Functional languages:
  - Functional programming offers several desirable features for physics analyses:
    - Declarative
    - Stateless
    - Lazy
- Both of these concepts lead to more modular code:
  - Insulate analysis code from data storage location and file format
  - Insulate each section of code from other parts of the code

[1] https://opentextbc.ca/dbdesign01/chapter/chapter-3-characteristics-and-benefits-of-a-database/

Interface (front end)

- FuncADL is:
    - a functional query interface
    - modeled after Language INtegrated Query (LINQ[2], part of C#)
    - using Python as a host language (embedded DSL)
- Queries are built from a set of basic operators like Select, Where, Count, etc.
- Example:
    - To retrieve E_T^{miss} in all events with at least two jets with p_T > 40 GeV:

```
EventDataset(dataset_identifier)\

    .Where(lambda event: event.Jet_pt.Where(lambda pt: pt > 40).Count() >= 2)\

    .Select(lambda event: event.MET_pt)
```

[2] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

Interface (front end)

```
EventDataset(dataset_identifier)\

      .Where(

            lambda event: event.Jet_pt.Where(

                  lambda pt: pt > 40

            )

            .Count() >= 2

      )\

      .Select(

            lambda event: event.MET_pt

      )
```

`EventDataset()` yields a sequence of events

`Where()` applies a filter function to each sequence element

`Jet_pt` is a sequence within each event

`Count()` reduces a sequence to an integer (its length)

`Select()` applies a transformation to each sequence element

`MET_pt` is a single value in each event

Execution (back end)

- A back end implementation translates the FuncADL query into appropriate code for execution on the underlying file format
- Code generation is done by traversing the Python abstract syntax tree of the FuncADL query and forming a native representation of each tree node
- Currently three implementations:
  - Uproot back end
    - Generates Python code utilizing Uproot
    - Can operate on any flat ROOT ntuple
      - For example: CMS NanoAOD
  - xAOD (ATLAS) back end
    - Generates C++ code that utilizes AnalysisBase
  - CMS Run 1 AOD back end
    - Generates C++ code that utilizes CMSSW
  - More to come!

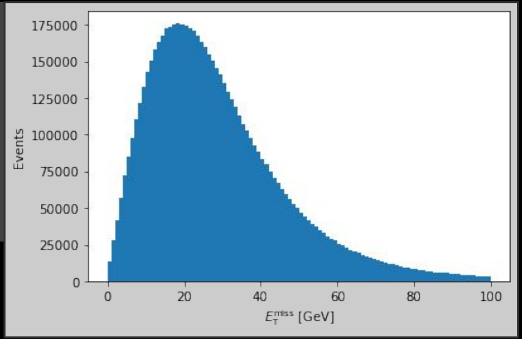
Full standalone examples

Several example queries are explained in [this notebook] using CMS open data:

```
>>> from func_adl_uproot import UprootDataset
>>> ds = UprootDataset('root://eospublic.cern.ch//eos/root-eos/benchmark/Run2012B_SingleMu.root')
>>> filtered_missing_ET = ds\
    .Where(lambda event: event.Jet_pt.Where(lambda pT: pT > 40).Count() >= 2)\
    .Select(lambda event: event.MET_pt)
>>> filtered_missing_ET.value()
<Array [15, 44.7, 30.5, ... 123, 30.3, 20.4] type='6665702 * float32'>
```

After the queried data is returned, you can continue your analysis in Python from there:

```
>>> import matplotlib.pyplot as plt
>>> plt.hist(filtered_missing_ET, bins=100, range=(0, 100))
>>> plt.xlabel(r'$E_\mathrm{T}^\mathrm{miss}$ [GeV]')
>>> plt.ylabel('Events')
>>> plt.show()
```



ServiceX

- The primary use case for FuncADL so far is with ServiceX:
  - A high-performance data delivery service
  - Provides a centralized and highly scalable platform to run FuncADL queries
  - Can be used to efficiently query large LHC Grid datasets
- ServiceX instances hosted at Nebraska and Chicago
- Can be run on any PC or cluster via Kubernetes
- See ACAT poster by Gordon Watts:
  - https://indico.cern.ch/event/855454/contributions/4596745/

ADL/CutLang

This work is supported in part by the **National Research Foundation of Korea** under grant no 2021R1I1A3048138.

ADL definition and scope

cern.ch/adl

ADL is a fully domain specific and declarative language that describes the physics content of a collider analysis in a standard and unambiguous way, independent of software frameworks.

Puts the focus on physics, allows to communicate analyses easily between different groups, exp, pheno, students, public, …

ADL domain scope:

- Event processing: Priority focus.

Event processing…

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, …) → output: event selection

- Analysis results, i.e. counts and uncertainties: Available
- Histogramming: Partially available.
- Systematic uncertainties: To be within the scope. Work in progress.

The ADL construct

cern.ch/adl

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, …).

LHADA (Les Houches Analysis Description Accord): Les Houches 2015 new physics WG report (arXiv:1605.02684, sec 17)

CutLang: Comput.Phys.Commun. 233 (2018) 215-236 (arXiv:1801.05727) ACAT 2019 proceedings (arXiv:1909.10621) arXiv:2101.09031

Simple analysis example with ADL

```
# OBJECTS
object goodJet
  take jet
  select pT(jet) > 30
  select abs(eta(jet)) < 2.4

object goodMuon
  take Muon
  select pT(Muon) > 30
  select abs(eta(Muon)) < 2.4

object goodEle
  take Ele
  select pT(Ele) > 30
  select abs(eta(Ele)) < 2.5

object goodLep
  take union(goodEle, goodMuo)
```

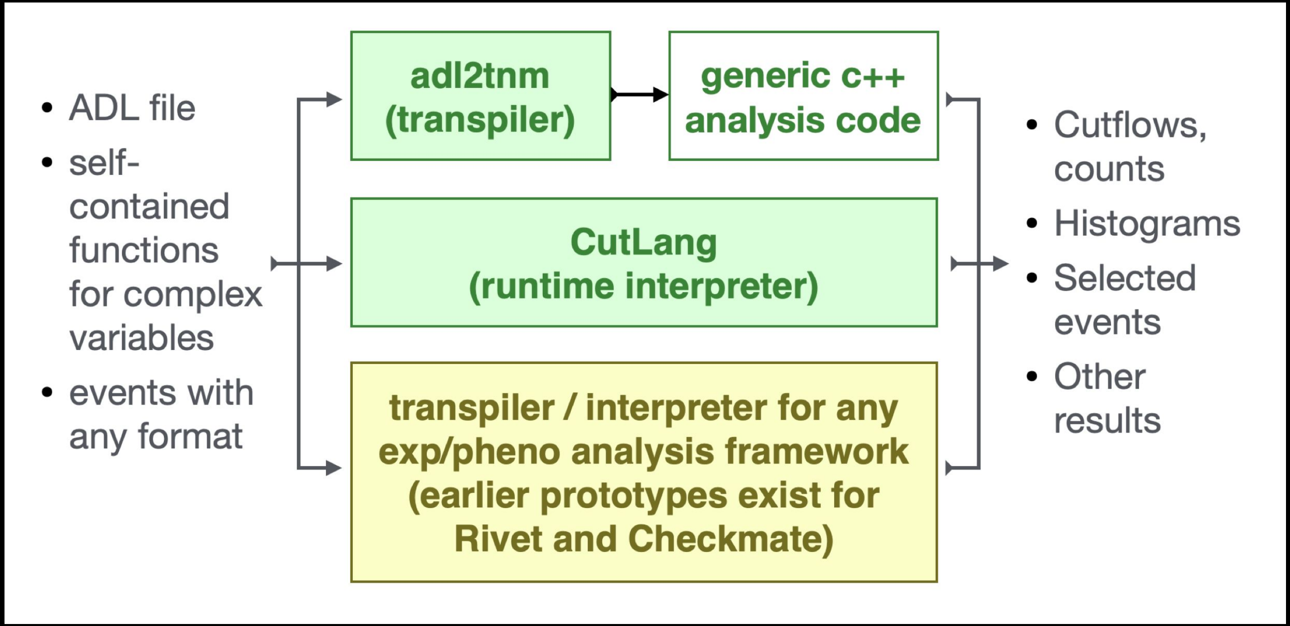
```
# EVENT VARIABLES
define HT = fHT(jets)
define MTl = Sqrt( 2*pT(goodLep[0]) * MET*(1-cos(phi(METLV[0]) - phi(goodLep[0]) )))

# EVENT SELECTION
region SR
  select size(jets) >= 2
  select HT > 200
  select MET > 200
  select MET / HT <= 1
  select Size(goodEle) == 0
  select Size(goodMuon) == 0
  select dphi(METLV[0], jets[0]) > 0.5
  select dphi(METLV[0], jets[1]) > 0.5
  select size(jets) >= 3 ? dphi(METLV[0], jets[2]) > 0.3 : ALL
  select size(jets) >= 4 ? dphi(METLV[0], jets[3]) > 0.3 : ALL
  histo hMET , "met (GeV)", 40, 200, 1200, MET
  histo hHT , "HT (GeV)", 40, 200, 1600, HT
```

Implementations of published LHC analyses in ADL analysis database : https://github.com/ADL4HEP/ADLLHCanalyses

Experimental / pheno analysis model with ADL

ADL is not bound to a specific framework.

It can be run on events by any infrastructure capable of parsing and executing it.



Running ADL analyses with CutLang

CutLang runtime interpreter:

- No compilation. Directly runs 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 Github: https://github.com/unelg/CutLang

CutLang publications: arXiv:1801.05727, arXiv:1909.10621 arXiv:2101.09031

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, histograms for each region in “TDirectory”s.
- Available in Docker, Conda, Jupyter (via Conda or binder).

CutLang core team: G. Ünel, B. Göktürk, B. Örgen, A. Paul[1], N. Ravel[1], S. Sekmen, J. Setpal, B. Şen, A. M. Toon[1], A. Adiguzel et.al.

[1]CERN summer students.

Physics with ADL/CutLang

Designing new analyses:

- Experimental analyses:
  - 2 ATLAS EXO analyses ongoing
- Phenomenology studies:
  - E6 isosinglet quarks at HL-LHC & FCC w/ CutLang (Eur Phys J C 81, 214 (2021)).
- Analysis of LHC Open Data:
  - Training exercises implemented: link
  - Demo at CMS Open Data workshop: link
  - CMS OD Summer Student Workshop: link
- Analysis optimization via differentiable programming (under development).

Using existing analyses:

ADL analysis database with ~15 LHC analyses: https://github.com/ADL4HEP/ADLLHCanalyses (more being implemented).
Validation of these analyses in progress.

- Reinterpretation studies:
  - Integrating ADL into the SModelS framework
- Analysis queries, comparisons, combinations:
  - Automated tools under development
- Long term analysis preservation in collaboration with CERN Analysis Preservation Group.

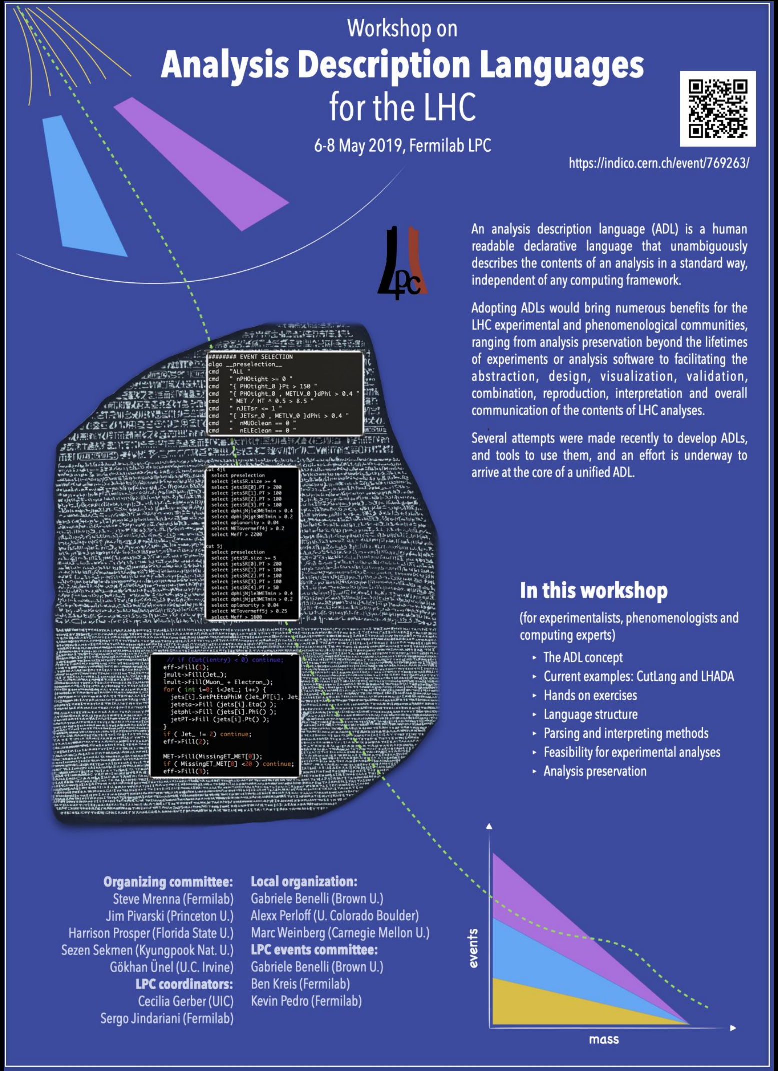
ADL/CutLang used for training for beginner students with no programming experience:

- 1st Data Analysis School with ADL+CutLang (3-7 Feb 2020), Istanbul (link, proceedings link)
- 26th Vietnam School of Physics (VSOP) (Dec 2020) (link)

Summary

- Analysis description languages (ADLs) are domain-specific languages (DSLs) that describe an analysis in a completely unambiguous way
- ADLs provide a declarative interface to specify the physics content of an analysis without the details of its execution
- FuncADL is an embedded DSL within Python that was inspired by functional programming and query languages
- ADL is an external DSL with its own runtime interpreter (CutLang) and transpiler (adl2tnm)
- Declarative interfaces like these make analyses easier to write and understand, and they simplify analysis preservation
- With increasing data set sizes as we move towards the high-luminosity LHC (HL-LHC), these declarative interfaces will be more important than ever

Backup



Recent dedicated workshop for a community-wide expert discussion.
Participation by experimentalists, phenomenologists, computer scientists.

- Overview of existing ADL efforts
- Language making tools

Extensive discussions on

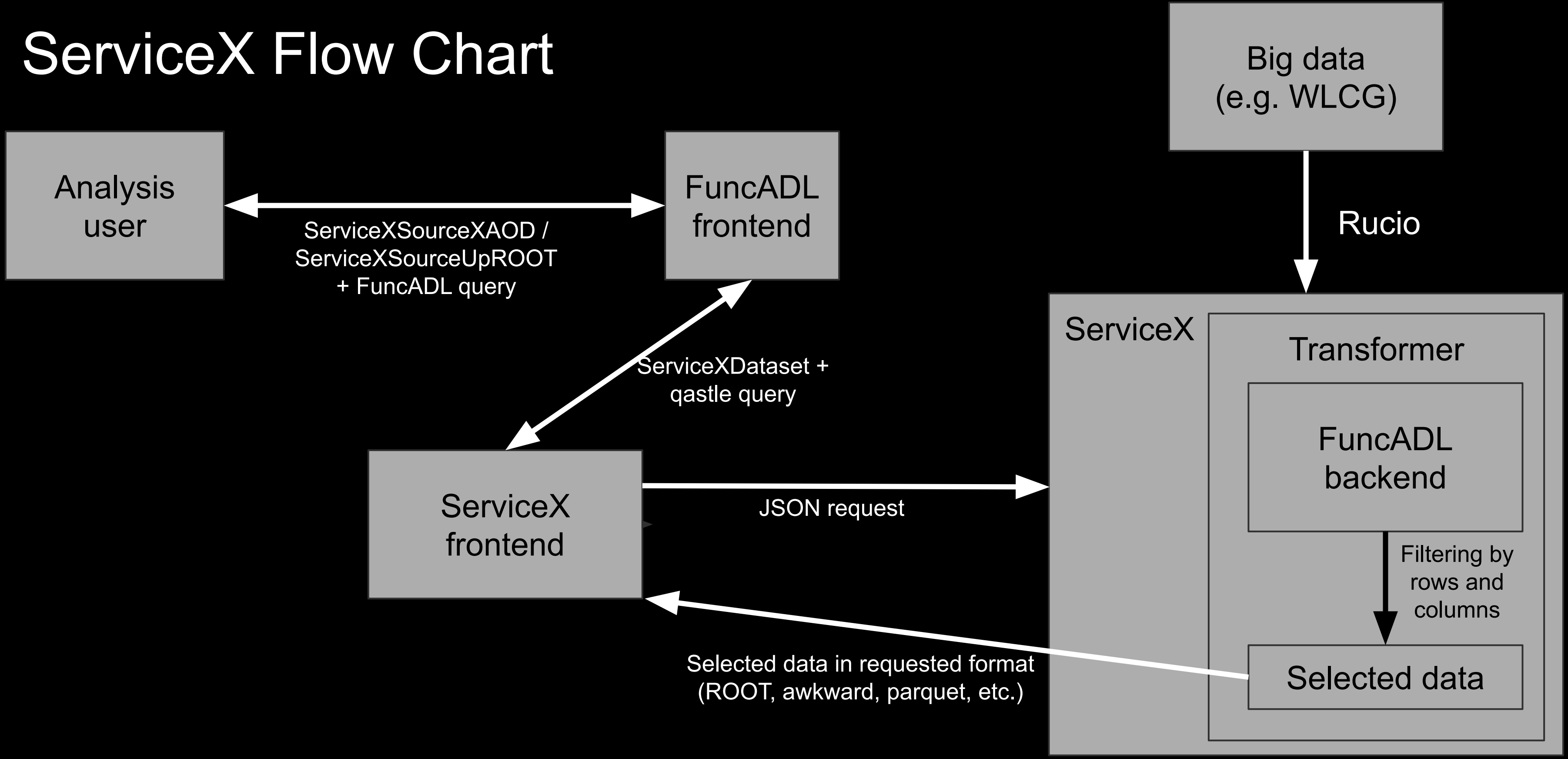
- Why/where do we need an ADL?
- ADL physics scope and content
- ADL users' requirements
- What kind of ADL syntax we need?
- Parsing / interpreting methods
- ADLs for analysis preservation

Detailed information on indico:
https://indico.cern.ch/event/769263/

FuncADL links

- FuncADL GitHub repositories:
  - https://github.com/iris-hep/func_adl
  - https://github.com/iris-hep/func_adl_servicex
  - https://github.com/iris-hep/func_adl_uproot
  - https://github.com/iris-hep/func_adl_xAOD
- ServiceX documentation, which includes FuncADL examples:
  - https://servicex.readthedocs.io/en/latest/user/getting-started/

ServiceX Flow Chart



ADL syntax: blocks, keywords, operators

Block purpose	Block keyword
object definition blocks	object
event selection blocks	region
analysis information	info
tables of results, etc.	table

Keyword purpose	Keyword
define variables, constants	define
select object or event	select
reject object or event	reject
define the mother object	take
define histograms	histo
applies object/event weights	weight
bins events in regions	bin

Operation	Operator
Comparison operators	> < => =< == != [] (include)][(exclude)
Mathematical operators	+ - * / ^
Logical operators	and or not
Ternary operator	condition ? truecase : falsecase
Optimization operators	~= (closest to) ~! (furthest from)
Lorentz vector addition	LV1 + LV2 LV1 LV2

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

ADL 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: MT2, 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

Running analyses with ADL: adl2tnm

H. B. Prosper,
S. Sekmen

- 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 ref: Les Houches 2017 new physics WG report (arXiv:1803.10379, sec 23)

adl2tnm Github: https://github.com/hbprosper/adl2tnm