Создание специализированной БД для хранения данных физики частиц, записанных на формальном языке.

Реализация Particle Physics Data System ОС-независимыми средствами: C, yacc, lex

Герасимов А. С., МФТИ, группа № 324
Научный руководитель: Зенин О. В.

11 июля 2017

Постановка и обоснование задачи

В ИФВЭ с ∼1980 г. развиваются базы данных по физике частиц, PPDS (Particle Physics Data System)

- **DataGuide:** 1 запись – 1 экспериментальная статья (всего 33716 статей)
  - формальная запись об эксперименте (установка, реакции, энергия пучка(ов), измеряемые величины, изучавшиеся частицы и их параметры
  - Библиографическая информация: выходные данные статьи, список авторов, коллаборация, название статьи, абстракт, ссылки (в том числе на статьи с предварительными/окончательными результатами того же измерения).
- **ReactionData:** 1 запись – 1 экспериментальная статья (всего 9958 статей)
  - то же, что DataGuide, но меньше библиографических данных, и есть табличные данные по опубликованным измерениям (дальний родственник – HEPDATA).
- **CrossSections:** фактически ReactionData, но содержит только измерения полных сечений

Постановка и обоснование задачи

- **Vocabulary** – словарь PPDL (Particle Physics Data Language): **все** данные (кроме авторов, названий статей, абстрактов и комментариев) во всех базах закодированы на этом языке:
  - частицы: обозначения, синонимы, квантовые числа, pdgId
  - измеряемые величины: полные и дифференциальные сечения, асимметрии, массы, ширины ....
  - единицы измерения
  - ускорители, детекторы, эксперименты, институты

Словарь PPDL задает "слова" языка описания данных, но не его синтаксис.

Пример: синтаксис реакций (после синтаксического разбора реакций, программирование других конструкций языка трудностей не представляет):

`RE = PI- P → N F1(1285) < ETA < 2GAMMA > PI+ PI- > + N ETAPRIME(958);`

Работы, в которых изучалась записанная так реакция, должны находиться, например, по запросу `RE = P PI → PI+ PI- GAMMA GAMMA;`

Постановка и обоснование задачи

Система PPDS была реализована в ИФВЭ на базе Berkeley database management system (BDMS)
Последние версии работают на платформе DEC VAX/VMS (`telnet npt.ihep.su`):

- использование специфических для DEC VAX расширений языков C/Fortran
- использование специфических для DEC VAX библиотечных подпрограмм
- ядро базы данных и парсеры конструкций языка PPDL запрограммированы на DEC Fortran (кроме парсера реакций – lex/yacc/C, $\sim$1988)
- существующий код не адаптируется под GNU/Linux (вообще, под любую POSIX-совместимую ОС) за разумное время
- под эмулятором VAX/VMS (`telnet npt.ihep.su`) PPDS можно поддерживать неограниченно долго, но:
  - уходят авторы кода
  - под VMS нет простого сопряжения PPDS с веб-сервером $\Rightarrow$ базы данных потеряны для потенциальных пользователей $\Rightarrow$ нет обратной связи $\Rightarrow$ консервация 40-летнего труда

Постановка и обоснование задачи

Реализовать PPDS с нуля стандартными средствами:

- POSIX-совместимая ОС: Unix (GNU/Linux, POSIX-подсистема Windows, etc.)
- стандартный C, реализуемый распространенными компиляторами C (GCC 3.X/4.X, etc.)
- программирование парсеров языка не вручную, а стандартными генераторами кода: LEX, YACC $\Rightarrow$ меньше написанного вручную кода $\Rightarrow$ в случае недоступности авторов в программах легко разобраться по исходникам и минимальной документации

Первый подход к решению:

- Макет DataGuide: прочесть “дамп”, разобрать его на записи, записи разобрать в соответствии со структурой записи (сл. слайд)
- Правые части в `KEY = VALUE;` надо идентифицировать с соответствующими записями в словаре, прочитанном из “дампа” Vocabulary: частицы с их квантовыми числами; ускорители, детекторы, журналы, институты ...

Структура записи DataGuide (упрощенно)

```
<record>
|-SC(short code)
|  |-PDGSC
|-IRN(spires id)
|-R(eference)
| |-TY(pe of reference)
| |-D(ate)
|-AUTHORS
| |-A(uthor)
| | |-I(institution)
|-T(itle)
| |-TITLE-TEX
|-ABS(tract)
| |-ABS-TEX
|-EXPERIMENT
| |-AC(celerator)
| |-DE(tector)
| |-PR(oposal)
| |-COL(laboration)
| |-CE(comment)
| | |-CE-TEX
.....
....
| |-CD(comment on data)
| | |-CD-TEX
| |-REAC-DATA
| | |-RE(action)
| | |-B(eam energy)
| | |-DD(data description)
| |-PART-PROP
| | |-P(article)
| | |-PP(particle property)
.....
```

Пример записи DataGuide

```
SC = AMELIN 00;
IRN = 4392124;
R = NP A668, 83;
COD = NUPHA,A668,83;
TY = JOUR;
D = 2000;
AUTHORS;
A = Amelin, D.V.;
...
I = SERP;
...
T = Natural Parity Resonances in {ETA} {PI+} {PI-};
ABS = ... ;
EXPERIMENT;
AC = SERP; DE = VES; PR = SERP-E-164; COL = VES Collaboration;
REAC-DATA;
RE = PI- BE --> PI+ PI- 2GAMMA ANYTHING;
B = 37 GEV (PLAB);
DD = MASS;
----------------------------------------
----------------------------------------
REAC-DATA.;
RE = PI- P --> N ETA PI+ PI-;
B = 37 GEV (PLAB);
PLAB = 37.0000,37.0000;
DD = PWA;
PART-PROP;
P = F2(1565); P. = RHO3(1690)0; P. = F4(2300);
PP = MASS; PP. = W; PP. = QN;
PART-PROP.;
P = EXOTIC; PP = EX;
----------------------------------------
```

Та же запись в *хэшах*:

```
(null) {
 SC { v:62464478336d75474439775453596b71714b66355431 }
 IRN { v:337565537874365459 7a3569754d6a765558356a7331 }
 R { v:4a2f67524c6b2e6e7a30573939756a3542535a773531
  COD { v:5279767236573870434b354c48532f7341527269422e }
  TY  { v:4b6148526847614879664e556d3167356c71346c2f }
  D   { v:437a4859347032352e65714f6f6461526a3763795631 }
 }
 AUTHORS {
  A { v:477131383559574a4d4d4e754d46447634796451492e   }
  ...
 }
 T { v:6c7166675537613049324d4e6d5057636865426b6f2e
  TITLE-TEX { v:465a4f7539546a2f5637303033676145757152 33512f } |}
 }                                                              |}
 ABS { v:54387447583075344f4341657451326 34f3771494c2e  }
 ...
 EXPERIMENT {
  AC { v:6c706e6a537a5a41496c3030356e656d73336274442f }
  DE { v:386d766b3774353744527033546845773 06a2f4d2f2f }
  PR { v:4b386a303638424a725869325631346 8466e7a4e702f }
  COL {  v:3953522f726b35684e745158475a37683346796c4331 }
  REAC-DATA {
      RE { v:4863714f504955386c6c552e7175775248677867 2e30 }
      B {  v:546e6f3538422f45466b7048597830437a316f432f2e }
      DD { v:4b794c3659796d6956592f7234676f42515073616931 }
  }
  REAC-DATA {
      RE { v:4b5a7a496270494f64336b36314d435467656b496131 }
      B {  v:546e6f3538422f45466b7048597830437a316f432f2e }
      DD { v:75714362655662526d456a5a48574c6936524c49562f }
  }
  ..........
```

```
.....
| PART-PROP {
|  P { v:0000000000000000000000000000000000000000
|  P { v:0000000000000000000000000000000000000000
|  P { v:0000000000000000000000000000000000000000
|  PP { v:4b794c3659796d6956592f7234676f42515073
|  PP { v:79496f39304437736561527a70772e55644741
|  PP { v:647a34425a436842445647626966426359424e
| }
| PART-PROP {
|  P { v:0000000000000000000000000000000000000000
|  PP { v:3832772f525938356533795945 6d4b2e777242
| }
|}
|}
```

Определение структуры записи в программе (C):

Запись DataGuide разбирается на связанные в дерево элементы КЛЮЧ = ЗНАЧЕНИЕ. Правила обращения с парой КЛЮЧ = ЗНАЧЕНИЕ определяются следующей структурой:

```
struct record_key {
  char *s;                    /* название ключа. Может принимать значения
                              RE (реакция), DD (измеряемая величина),
                              P (частица) и др.*/
  enum keytype key, par;      /* id ключа и его родителя  */
  int  noval;                 /* значение ключа, если оно есть. Если значения
                              нет, то записываем ноль (Напр, для EXPERIMENT)*/
  void  (*vy_init)(void); /* адрес функции для чтения допустимых значений
                              ключа из словаря PPDL при запуске базы */
  void* (*s2t)(char*);        /* адрес функции, переводящей строку ЗНАЧЕНИЕ
                              в переменную типа struct type_КЛЮЧ  */
  char* (*t2s)(void*);        /* адрес функции, переводящей переменную типа
                              struct type_КЛЮЧ в строку. */
  char* (*t2h)(void*);        /* адрес функции, вычисляющей уникальный ‘‘хэш’’
                              по переменной типа struct type_КЛЮЧ */
  int   (*eq )(void*,void*); /* возвращает 1, если две переменные одного
                                типа struct type_KEY равны, 0, если не равны */
};
```

Определение структуры записи в программе (C):

```
struct record_key keys[] = {
{"SC"               ,_SC_                 ,_RECORD_     , 0, 0,0,0,0,0}, //....
{"IRN"              ,_IRN_                ,_RECORD_     , 0, 0,0,0,0,0}, //....
{"R"                ,_R_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"COD"              ,_COD_                ,_R_          , 0, 0,0,0,0,0},
{"TY"               ,_TY_                 ,_R_          , 0, 0,0,0,0,0},
{"D"                ,_D_                  ,_R_          , 0, 0,0,0,0,0},
....
{"AUTHORS"          ,_AUTHORS_            ,_RECORD_     , 1, 0,0,0,0,0},
{"A"                ,_A_                  ,_AUTHORS_    , 0, 0,0,0,0,0},
{"I"                ,_I_                  ,_AUTHORS_    , 0, 0,0,0,0,0}, //....
{"T"                ,_T_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"TITLE-TEX"        ,_TITLE_TEX_          ,_T_          , 0, 0,0,0,0,0}, //....
{"ABS"              ,_ABS_                ,_RECORD_     , 0, 0,0,0,0,0},
{"EXPERIMENT"       ,_EXPERIMENT_         ,_RECORD_     , 1, 0,0,0,0,0},
{"AC"               ,_AC_                 ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"DE"               ,_DE_                 ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"PR"               ,_PR_                 ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"COL"              ,_COL_                ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"RR"               ,_RR_                 ,_EXPERIMENT_, 0, 0,0,0,0,0}, //....
{"REAC-DATA"        ,_REAC_DATA_          ,_EXPERIMENT_, 1, 0,0,0,0,0},
{"RE"               ,_RE_                 ,_REAC_DATA_ , 0, 0,0,0,0,0}, //....
{"B"                ,_B_                  ,_REAC_DATA_ , 0, 0,0,0,0,0}, //....
{"DD"               ,_DD_                 ,_REAC_DATA_ , 0, 0,0,0,0,0}, //....
{"PART-PROP"        ,_PART_PROP_          ,_EXPERIMENT_, 1, 0,0,0,0,0},
{"P"                ,_P_                  ,_PART_PROP_ , 0, vy_init_P,s2t_P,t2s_P,t2h_P,eq_P },
{"PP"               ,_PP_                 ,_PART_PROP_ , 0, 0,0,0,0,0},
....
};
```

Далее пример из ↑единственной строки с непустыми vy_init_KEY, etc. ⇒

Операции с KEY = P (частица): чтение словаря

vy_init_P() – при старте базы читаем из файла определения частиц в формате:

```
id B L s c b t is_anti-particle is_heavy-nucleus class mass I3 Q
     orcode имя1 [ имя2 ... ]
```

Например, запись для K^{*+}:

```
41 0 0 1 0 0 0 1 0 0 0.891590 0.500000 1.000000 1041892476 K*(892)+ K*+
```

записывается в

```
struct type_P {
 int id; /* наш внутренний id частицы */
 int  bar,lep, S, C, B, T, anti, heavy_nuc, class;
 double m, I3, Q;
 char orcode[32]; /* legacy: длинное целое число,
                     по которому вычисляются квантовые числа */
 int nval;   /* сколько имен-синонимов? */
 char **val; /* синонимы */
 long val0sum;
};
```

Чтение пары *P = VALUE* из дампа и запись в базу

При чтении выражения *KEY = VALUE;* из дампа DataGuide:

- вызываем s2t_KEY(VALUE) – если VALUE нашлось в прочитанном при старте словаре, возвращает адрес стуктуры – внутреннего представления KEY, в случае KEY = P это struct type_P* p; p = NULL означает, что VALUE в словаре нет $\Rightarrow$ ОШИБКА.
- вычисляем по *p хэш: *hash* = t2h_P(p) и пишем P { v:*hash* } в base/records/текущая_запись.
- ищем в base/P/h строку с *hash*:
  - если нашлось, то добавляем *hash текущая_запись* в base/P/r
  - если не нашлось, то дописываем *hash VALUE* в конец base/P/v, а в base/P/h дописываем *hash (адрес этой записи в base/P/v)*, и добавляем *hash текущая_запись* в base/P/r

Эта последовательность работает для любых пар *KEY = VALUE;* с ф-циями из таблицы `struct record_key keys[]`

Чтение пары *RE = VALUE* из дампа и запись в базу

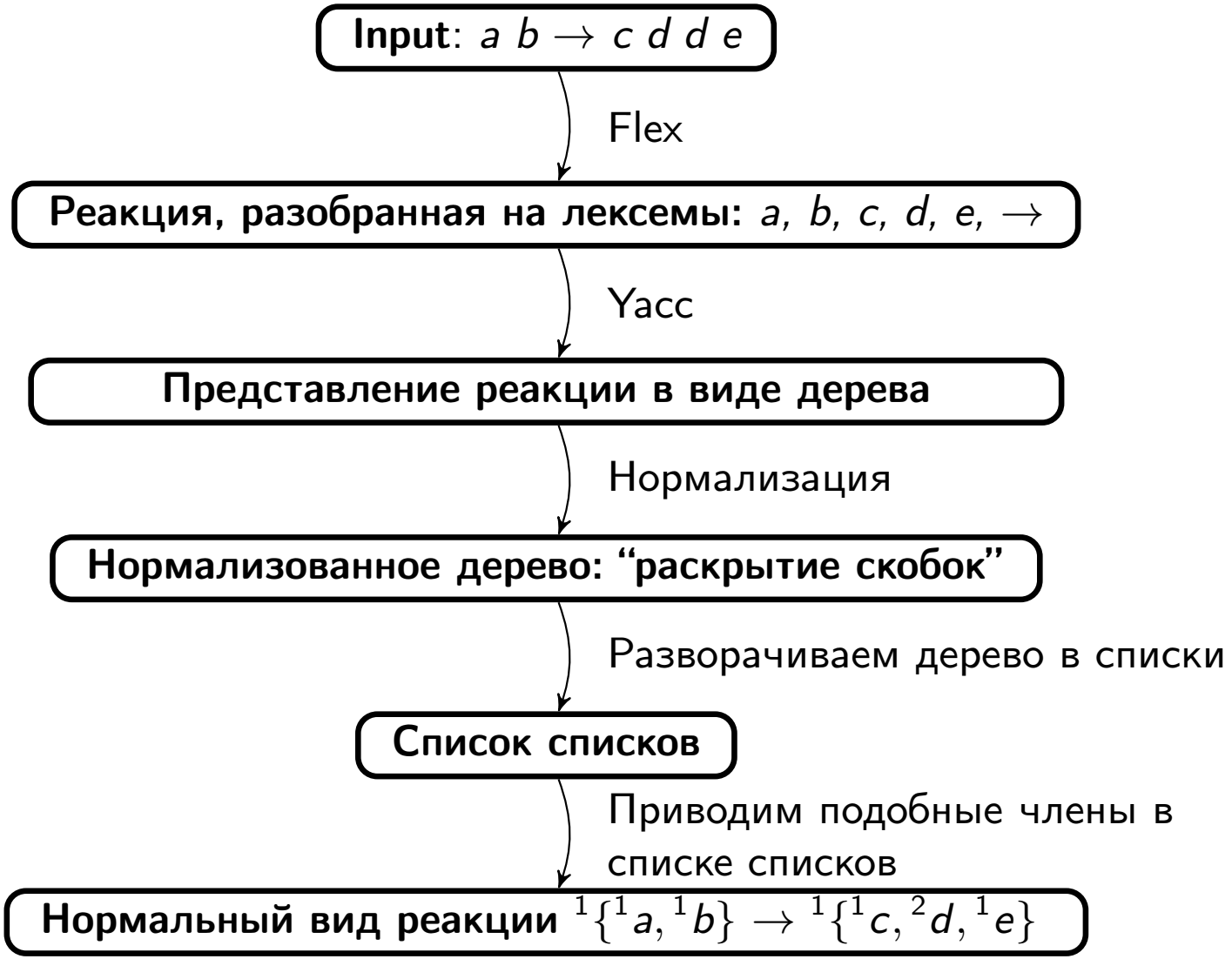
- **vy_init_RE == NULL**, из словаря читать ничего не надо.
- Запись реакции в виде строки:
  {начальное состояние} $\rightarrow$ {конечное состояние}

В качестве примера рассмотрим реакцию

```
RE = PI- P -->
N ( F1(1285) < ETA < GAMMA GAMMA > PI+ PI- > + ETAPRIME(958) ) ;
```

- Пучок/мишень: $\pi^- p$
- Изучались конечные состояния $n f_1(1285)$ и $n \eta'$
- Реконструкция $f_1(1285) \rightarrow \eta(\rightarrow 2\gamma)\pi^+\pi^-$

Алгоритм обработки введенной реакции



Разбор на лексемы средствами LEX

шаблон { операция; } :

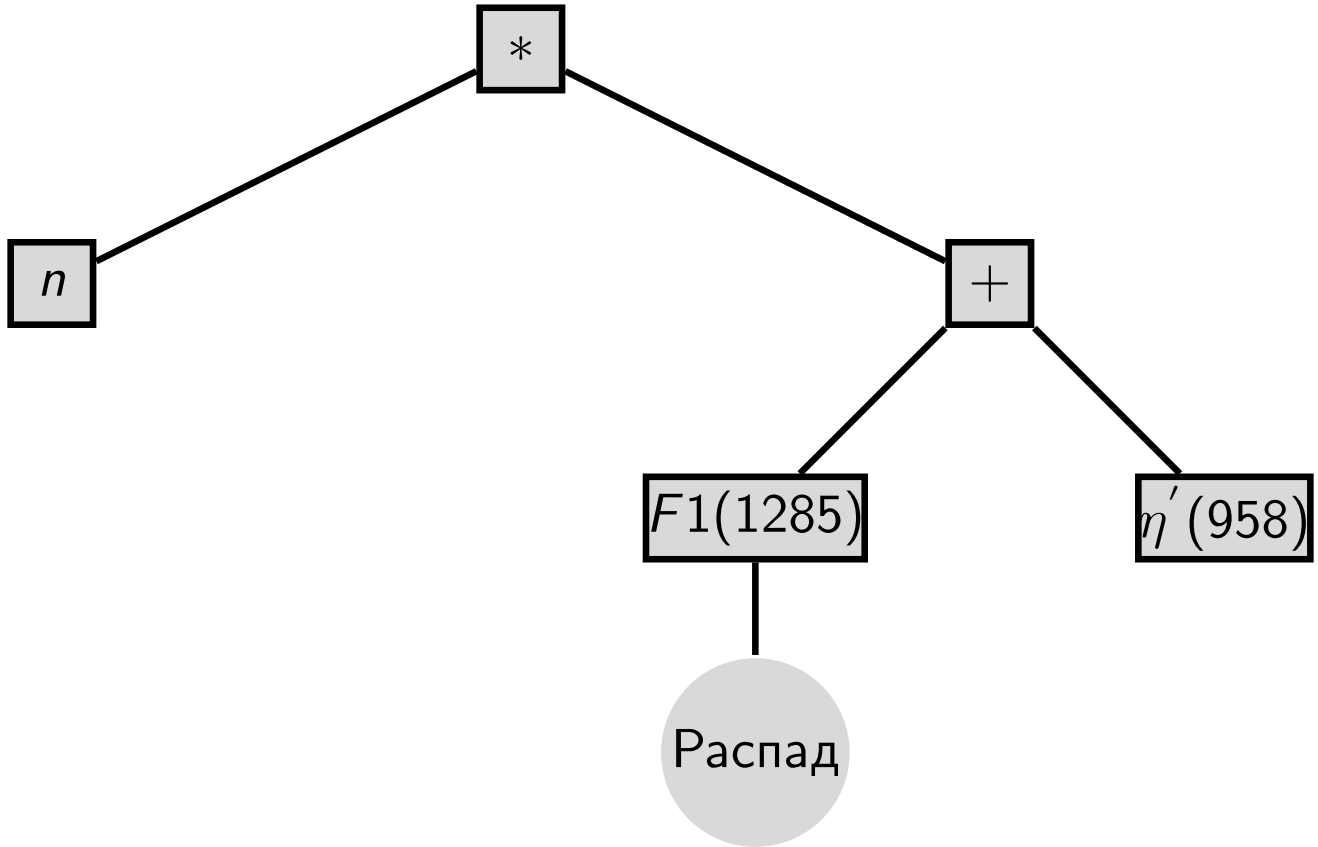
```
[ \t]+\-\-\>[ \t]+ { return ARROW; }
[A-Za-z][A-Za-z0-9_\*\+\-\(\)]*|\([A-Za-z][A-Za-z0-9_\*\+\-\(\)]*\)
{yylval = prt_id(yytext) );  return PRT;}
[0-9]+ { yylval  = atoi(yytext)); return NUM;  }
[ \t]*\;        { return ';'; }
[ \t]+\+[ \t]+ { return '+'; }
[ \t]+\-[ \t]+ { return '-'; }
\([ \t]+        { return '('; }
[ \t]+\)        { return ')'; }
[ \t]+\<[ \t]+ { return '<'; }
[ \t]+\>        { return '>'; }
[ \t]+\.[ \t]+ { return '.'; }
[ \t]+          { return '*'; }
.               { fprintf(stderr,"Invalid char:%c:\n",*yytext); }
```

Синтаксический разбор средствами YACC

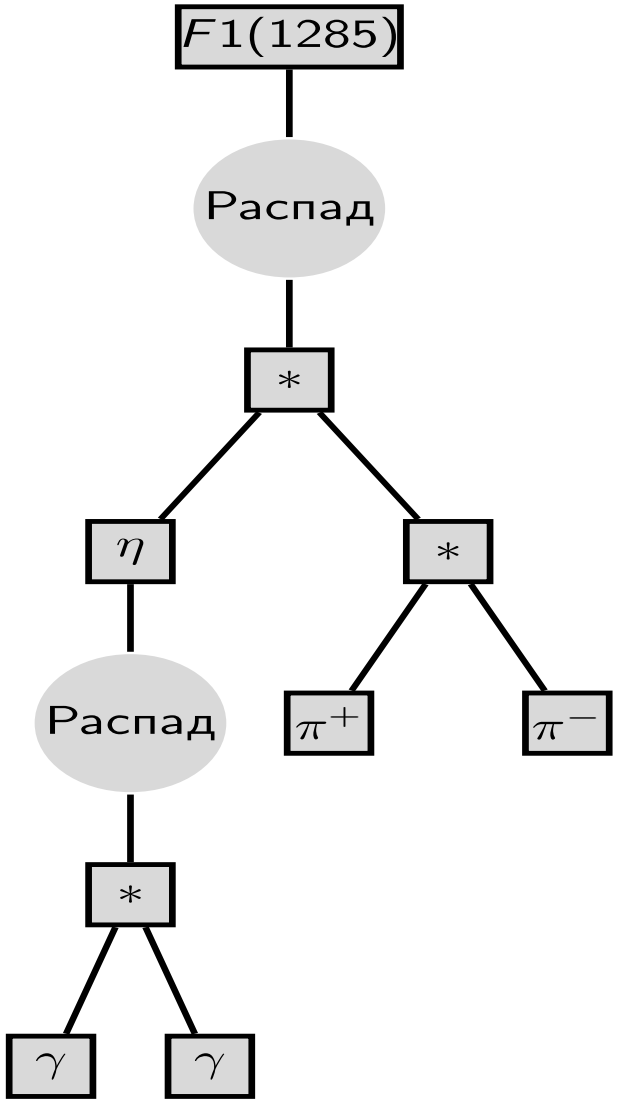
```
%token ARROW PRT NUM
%left '+' '-'
%left '*'
%left '.'
%%
input: input re ';' { }
       |
       ;
re:    expr ARROW expr { is = $<no>1; fs = $<no>3; }
     | expr            { is = $<no>1; fs = NULL;   }
     ;
expr:   PRT { $<no>$ = new_node(_TERM_, $<id>1,1, NULL, NULL,NULL); }
  | NUM PRT { $<no>$ = new_node(_TERM_, $<id>2,$<id>1, NULL,  NULL,NULL); }
  | PRT '<' expr '>' { $<no>$ = new_node(_TERM_, $<id>1,1, $<no>3, NULL,NULL); }
  | NUM PRT '<' expr '>' {
                  $<no>$ = new_node(_TERM_, $<id>2,$<id>1, $<no>4, NULL,NULL); }
  | expr '*' expr { $<no>$ = new_node(_MUL_,0,1, 0, $<no>1,$<no>3); }
  | expr '+' expr { $<no>$ = new_node(_PLUS_,0,1, 0, $<no>1,$<no>3); }
  | expr '-' expr { $<no>$ = new_node(_MINUS_,0,1, 0, $<no>1,$<no>3); }
  | '(' expr ')'  { $<no>$ = $<no>2; }
  ;
%%
```

Конечное состояние реакции в виде дерева

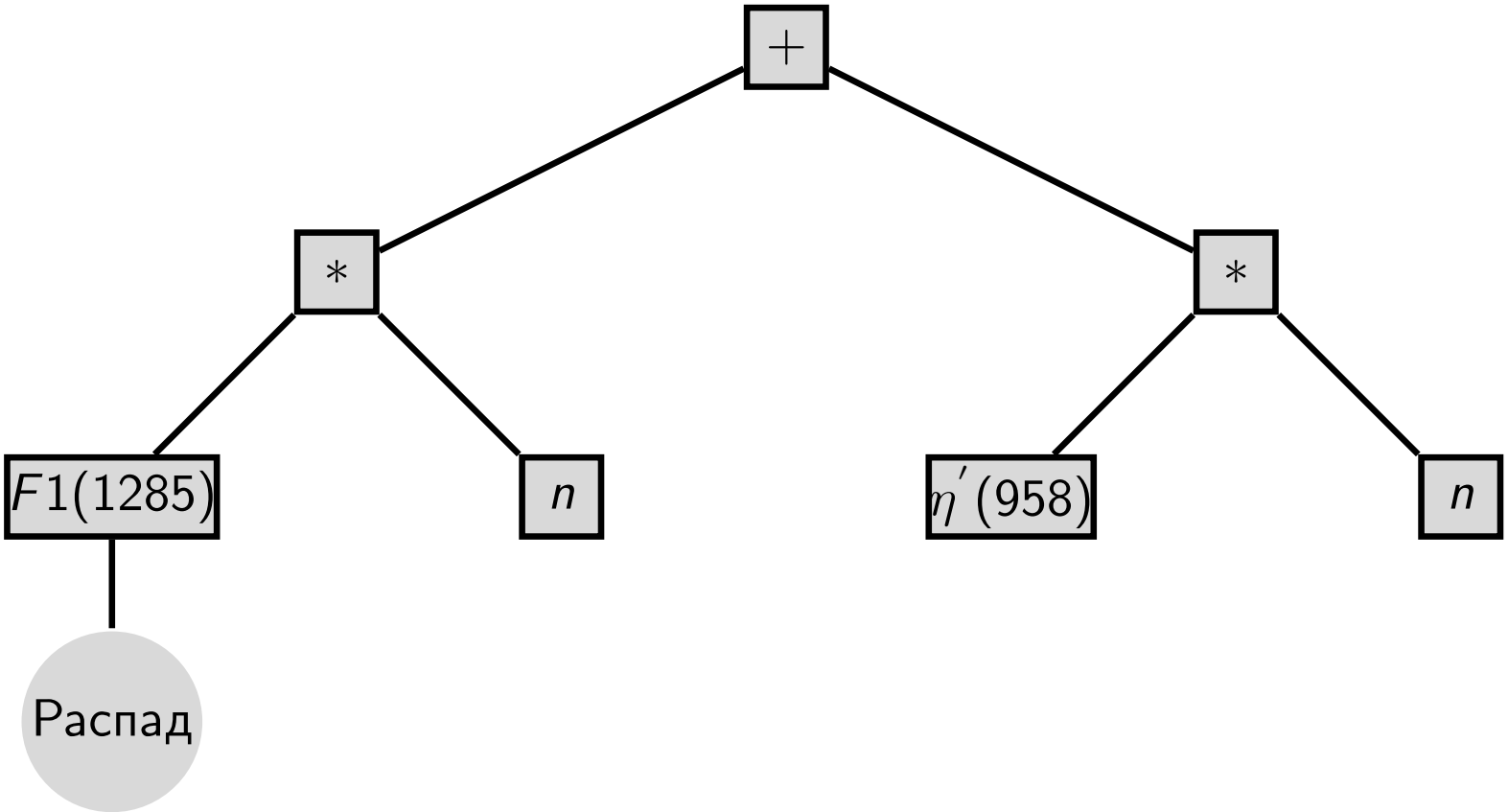
```
N ( F1(1285) < ETA < GAMMA GAMMA > PI+ PI- > + ETAPRIME(958) ) :
```



Конечное состояние реакции в виде дерева



- преобразуем дерево так, чтобы все узлы '*' были выше по дереву узлов '+', '-' (раскрытие скобок)



Дерево → список списков

- разворачиваем части дерева, содержащие только '*' и частицы, в списки
- сортируем эти списки по *id* частиц, приводим в них подобные члены
- разворачиваем дерево в список списков
- приводим подобные члены в списке списков

$$\left\{ {}^{1}\left\{ {}^{1}n, {}^{1}f_1(1285)\left\langle \left\{ {}^{1}\pi^-, {}^{1}\pi^+, {}^{1}\eta\left\langle {}^{2}\gamma \right\rangle \right\} \right\rangle \right\}, {}^{1}\left\{ {}^{1}n, {}^{1}\eta' \right\} \right\}$$

Эквивалентная запись конечного состояния

```
N F1(1285) < ETA < 2GAMMA > PI+ PI- > + N ETAPRIME(958);
```

дает тот же список:

$$\left\{ {}^{1}\left\{ {}^{1}n, {}^{1}f_1(1285)\left\langle \left\{ {}^{1}\pi^-, {}^{1}\pi^+, {}^{1}\eta\left\langle {}^{2}\gamma \right\rangle \right\} \right\rangle \right\}, {}^{1}\left\{ {}^{1}n, {}^{1}\eta' \right\} \right\}$$

Заключение

Поставленная в работе задача воспроизведения БД DataGuide “с нуля” стандартными современными средствами решена частично. Разработаны и отлажены программы, выполняющие следующие задачи:

- Разбор “дампа” эксплуатируемой на VAX/VMS БД Vocabulary на записи – определения частиц на языке PPDL (всего 4321 записей).
- Разбор “дампа” эксплуатируемой на VAX/VMS PPDS DataGuide на записи, соответствующие отдельным экспериментальным работам (всего 33716 экспериментальных работ).
- Синтаксический разбор записей DataGuide на организованные в деревья пары `КЛЮЧ = ЗНАЧЕНИЕ`.

- Разбор пар `P(article) = ЗНАЧЕНИЕ`: правая часть проверяется на соответствие зачитанному выше словарю PPDL.
- Разбор пар `RE(action) = ЗНАЧЕНИЕ`: синтаксический разбор записанной на PPDL реакции и ее приведение к однозначному нормальному виду, необходимому для реализации поиска по реакции и/или части реакции.
- Сохранение разобранного “дампа” DataGuide в виде набора файлов, пригодного для реализации поиска, аналогичного существующему в эксплуатируемой на VAX/VMS БД DataGuide.
- Решена наиболее сложная из необходимых для “спасения” системы PPDS задач – независимо воспроизведен процессор реакций, записанных на языке PPDL.
- Разработанный программный код беспрепятственно дорабатывается до полного функционального аналога эксплуатируемой в ИФВЭ БД PPDS DataGuide.

Спасибо за внимание!