Министерство образования и науки Российской Федерации

Федеральное государственное автономное
образовательное учреждение высшего образования
«Московский физико-технический институт
(государственный университет)»

Факультет общей и прикладной физики
Кафедра физики высоких энергий НИЦ "Курчатовский Институт" – ИФВЭ

Создание специализированной БД для хранения данных физики частиц, записанных на формальном языке. Реализация Particle Physics Data System ОС-независимыми средствами: C, yacc, lex

Выпускная квалификационная работа бакалавра

Выполнил:
студент 324п группы
Герасимов А.С.

Научный руководитель:
канд. физ.-мат. наук Зенин О.В.

Москва 2017

Содержание

Введение

Физика высоких энергий решает обширный комплекс проблем и задач, одной из которых является систематическое сравнение совокупности мировых экспериментальных данных с теоретическими утверждениями и феноменологическими моделями для выяснения текущего уровня согласия теоретического и эмпирического знания.

Проблема верификации эмпирического и теоретического знания приобретает особую значимость в реальных исследовательских ситуациях. Часто исследователю требуется обращение к большим массивам информации, постоянно обновляющимся и имеющим нелинейную структуру, что создаёт определённые трудности в принятии целого ряда решений:что же мы на самом деле наблюдаем, насколько обоснованно введение того или иного термина, как верифицировать предполагаемое существование гипотетического объекта, какова внутритеоеретическая взаимосвязь между данными терминами, какую следует дать эмпирическую интерпретацию конкретным теоретическим сущностям и т.п.

Решение данной задачи невозможно без создания единой базы данных физики частиц, одной из форм реализации которой является разработанная и эксплуатируемая в ИФВЭ система Particle Physics Data System (PPDS).

Создание специализированной БД для хранения данных физики частиц, записанных на формальном языке с использованием ОС-независимых средств (C, yacc, lex) позволяет решить проблему эффекивности обработки данных, упростить пользователям работу с ними.

Глава 1

Постановка задачи

Наборы данных по наблюдаемым в ФВЭ частицам накапливаются и сопровождаются в системе PPDS ИФВЭ в рамках программы КОМПАС. В качестве источников данных используются открытые публикации в реферируемых научных журналах. Сопровождение данных ведется по единой методике, включающей следующий цикл, повторяемый вместе с публикацией новых экспериментальных данных:

1. Контроль полноты данных по тематикам.
2. Перенос табличных данных из публикаций в БД.
3. Выделение предварительных данных.
4. Выделение модельно-зависимых данных.
5. Проверка соответствия содержимого БД оригинальным публикациям.

По некоторым тематикам (например, полные адронные сечения в столкновениях *адрон адрон*, γ *адрон*, e^+e^-) проводится периодическая проверка соответствия накопленных в PPDS данных современным теоретическим моделям.

Поисковая часть системы PPDS была реализована в ИФВЭ на ЭВМ DEC/VAX и Alpha, работающих под управлением ОС VMS. Базы данных используют иерархическую СУБД BDMS/4 – Berkeley Database Management System, первоначально разработанную в рамках Particle Data Group (LBNL, США) и существенно развитую в ИФВЭ [1].

Использование существующей системы PPDS в настоящее время затруднено по следующим причинам:

- необходимость использования эмулятора DEC/VAX в отсутствие физической машины
- программный код имеет объем в десятки тысяч строк и практически неподдерживаем в отсутствие его авторов

- под VAX/VMS затруднено сопряжение PPDS с веб-сервером, что ограничивает доступ к базам данных потенциальных пользователей, и, таким образом, приводит к отсутствию “обратной связи”

Адаптация существующего программного кода для POSIX-совместимых ОС невозможна в разумные сроки по следующим причинам:

- использование специфических для DEC VAX расширений языков C/Fortran, затрудняющих перенос системы на распространенные POSIX-совместимые ОС с использованием распространенных компиляторов, например gcc и gfortran

- использование специфических для DEC VAX библиотечных подпрограмм, которые должны быть независимо воспроизведены в POSIX-совместимой ОС

- поисковое ядро базы данных и парсеры конструкций языка Particle Physics Data Language (PPDL) [2] запрограммированы на DEC Fortran с агрессивным использованием битовых операций для упаковки данных

Вышеперечисленные причины приводят, по меньшей мере, к консервации 40-летнего труда по созданию и заполнению баз данных PPDS.

Исходя из сказанного выше, в данной работе была поставлена следующая задача – разработать “с нуля” функциональный аналог одной из баз PPDS, DataGuide, удовлетворяющий следующим требованиям:

- работа в современных POSIX-совместимых ОС, например, в GNU/Linux

- использование распространенных программных инструментов:
  - компилятор GNU C (gcc) [3]
  - лексический анализатор LEX [4]
  - генератор парсеров YACC [5]

Использование FLEX и YACC сокращает в десятки раз количество написанного вручную кода для разбора конструкций языка PPDL, и делает код обозримым и легко модифицируемым без участия его авторов.

Глава 2

Современное состояние PPDS

Эксплуатируемая в настоящее время в ИФВЭ система PPDS включает следующие БД:

DG (DataGuide) – содержит информацию, извлеченную из экспериментальных публикаций (но не сами экспериментальные данные).

Поиск данных возможен по номеру эксперимента, авторам публикации, ускорителю, детектору, реакции, частице пучка, импульсу пучка, частице мишени, частице конечного состояния реакции, наблюдаемой величине, более ранним публикациям эксперимента и другим атрибутам эксперимента и публикации.

Интервал поиска: 1895 → современность.

RD (ReactionData) – содержит табличные экспериментальные данные, извлеченные из экспериментальных публикаций (дифференциальные и полные сечения реакций, структурные функции, поляризационные наблюдаемые и другие измеряемые величины физики частиц).

Поиск данных возможен по номеру эксперимента, первому автору публикации, детектору, реакции, частице пучка, импульсу пучка, частице мишени, частице конечного состояния реакции, наблюдаемой величине и другим атрибутам эксперимента и публикации.

Интервал поиска: 1952 → современность.

CS (CrossSections) – содержит формально оцененные данные по интегральным сечениям и средним множественностям адронов в различных столкновениях. Включает данные из компиляций CERN—HERA и из старых компиляций UCLRL и LBL. Все данные проверены повторно в сравнении с оригинальными публикациями. Предварительные данные помечены. Модельно зависимые данные помечены. Записи дополнены данными о систематических погрешностях из оригинальных публикаций.

Поиск данных возможен по первому автору публикации, реакции, частице пучка, импульсу пучка, частице мишени, частице конечного состояния реакции, наблюдаемой величине и другим атрибутам эксперимента и публикации. Пополняется регулярно данными из архивной базы RD после повторной сверки с оригинальными публикациями.

Интервал поиска: 1950 → современность.

PP (ParticleProperties) – содержит оцененные данные по свойствам частиц из Review of Particle Properties Summary Tables группы PDG.

Поиск данных возможен по имени частицы, квантовым числам, модам распада, частице конечного состояния в распаде и по другим атрибутам частицы.

На данный момент не актуальна, последняя синхронизация с RPP в 1993 г.

VY (VocabularY) – содержит определения терминов, имена и синонимы частиц, ускорителей, детекторов и многие другие лексические единицы формального языка описания данных физики частиц – PPDL, используемого во всех базах данных PPDS.

EX (EXperiments) – содержит данные, описывающие одобренные предложения экспериментов большинства исследовательских лабораторий по физике частиц.

Поиск данных возможен по номеру эксперимента, авторам предложения, ускорителю, детектору, реакции, частице пучка, импульсу пучка, частице мишени, частице конечного состояния реакции, по библиографии публикаций эксперимента и другим атрибутам эксперимента.

Интервал поиска: 1961 → современность.

DY (DYscovery) – содержит описательную информацию о ключевых научных публикациях в ФВЭ с 1895 г. по настоящее время. В записях DY приводятся аннотации (выдержки) и формулировки “описательных формул” открытий. Используется также для подготовки публикации аннотированной Хронологии физики частиц. Первое издание Хронологии опубликовано как совместное издание ИФВЭ и LBNL (США) к столетию физики частиц в 1996 году .

2.1 Устройство БД DataGuide

Единицей хранения в БД DataGuide является *запись*, имеющая древовидную структуру. Упрощенно структура записи DG показана на Рис. 2.1а. “Листьями” дерева являются пары `КЛЮЧ = ЗНАЧЕНИЕ`, например, `R = NP A668, 83; RE =`

```
<record>                              SC = AMELIN 00;
|-SC(short code)                      IRN = 4392124;
| |-PDGSC                             R = NP A668, 83;
|-IRN(spires id)                      COD = NUPHA,A668,83;
|-R(eference)                         TY = JOUR;
| |-TY(pe of reference)               D = 2000;
| |-D(ate)                            AUTHORS;
|-AUTHORS                             A = Amelin, D.V.;
| |-A(uthor)                          ...
| | |-I(institution)                  I = SERP;
|-T(itle)                             ...
| |-TITLE-TEX                         T = Natural Parity Resonances in {ETA} {PI+} {PI-};
|-ABS(tract)                          ABS = ... ;
| |-ABS-TEX                           EXPERIMENT;
|-EXPERIMENT                          AC = SERP; DE = VES; PR = SERP-E-164; COL = VES Collaboration;
| |-AC(celerator)                     REAC-DATA;
| |-DE(tector)                        RE = PI- BE --> PI+ PI- 2GAMMA ANYTHING;
| |-PR(oposal)                        B = 37 GEV (PLAB);
| |-COL(laboration)                   DD = MASS;
| |-CE(comment)                       REAC-DATA.;
| | |-CE-TEX                          RE = PI- P --> N ETA PI+ PI-;
.....                                 B = 37 GEV (PLAB);
| |-CD(comment on data)               PLAB = 37.0000,37.0000;
| | |-CD-TEX                          DD = PWA;
| |-REAC-DATA                         PART-PROP;
| | |-RE(action)                      P = F2(1565); P. = RHO3(1690)0; P. = F4(2300);
| | |-B(eam energy)                   PP = MASS; PP. = W; PP. = QN;
| | |-DD(data description)            PART-PROP.;
| |-PART-PROP                         P = EXOTIC; PP = EX;
| | |-P(article)                      *E
| | |-PP(particle property)
```

(а) (б)

Рис. 2.1: Структура (а) и пример (б) записи DG

`PI- BE` $\rightarrow$ `PI+ PI- 2GAMMA ANYTHING`. Для каждого ключа определен единственно возможный ключ-родитель. Например, ключ `R(eference)` является родителем для ключей `TY(pe)` и `D(ate)`, и вместе они определяют ссылку на публикацию, закодированную в данной записи. Некоторые ключи (`AUTHORS`, `EXPERIMENT`, `REAC-DATA` и т.д.) не имеют своих собственных значений, и служат только родителями других ключей. Например, ключ `EXPERIMENT` логически объединяет ключи `AC(celerator)`, `DE(tector)`, `COL(laboration)`, ..., `REAC-DATA`, ключ `REAC-DATA` объединяет ключи `RE(action)`, `B(eam energy)`, `D(ata)D(escriptor)`. Значение ключа в общем случае не может быть произвольным. Например, значениями ключа `P(article)` могут быть только обозначения частиц, присутствующие в словаре Vocabulary. Значением ключа `RE(action)` может быть только синтаксически правильная запись реакции, в которой участвуют только присутствующие в Vocabulary частицы.

Пример записи в формате "дампа" DataGuide показан на Рис. 2.1б.

Полное определение структуры записи DataGuide [1] приведено в Приложе-

[1] File Definition Table (FDT), в терминах BDMS.

нии А.

Эксплуатируемая под эмулятором VAX/VMS БД DataGuide позволяет полностью выгрузить содержимое базы, то есть все имеющиеся в ней на данный момент 33716 записей, в текстовом формате, показанном на Рис. 2.1б. Словарь языка PPDL, содержащий 7349 входов, также полностью выгружается в сходном текстовом формате из БД Vocabulary. Эти так называемые "дампы" баз DataGuide и Vocabulary были приняты в качестве исходной точки для работы по независимому воспроизведению БД DataGuide.

Глава 3

Воспроизведение БД DataGuide

Решение задачи по воспроизведению БД DataGuide разбивается на две хорошо изолированных части:

- Считывание “дампа” базы и запись данных в постоянное хранилище в некотором внутреннем представлении.
- Реализация поиска записей в хранилище, выдача результатов поиска в человеко-читаемом формате.

Первая часть, в свою очередь, разбивается на следующие логические блоки:

1. Разбор “дампа” на отдельные записи.
2. Разбор дерева каждой записи на пары `КЛЮЧ = ЗНАЧЕНИЕ`.
3. Синтаксический разбор правых частей пар `КЛЮЧ = ЗНАЧЕНИЕ` и перевод их в некоторое внутреннее представление.
4. Запись разобранного дерева в хранилище.

Наибольшую сложность в этом списке представляет пункт 3 для ключа `RE(action)`, то есть синтаксический разбор записанной на языке PPDL реакции и приведение ее к виду, сводящему задачу сравнения двух реакций к сравнению списков. В данной работе эта часть задачи была успешно решена.[1] Программная реализация пунктов 1-4 кратко описана ниже.

Задача поиска записей в хранилище в данной работе не решалась.

[1] Отсутствие процессора реакций обесценивало предпринятые ранее попытки воспроизвести БД DataGuide с нуля.

3.1 Разбор записи DataGuide

Запись DataGuide разбирается на связанные в дерево элементы КЛЮЧ = ЗНАЧЕНИЕ приведенной в Приложении Е.3 программой. Правила обращения с парой КЛЮЧ = ЗНАЧЕНИЕ определяются следующей структурой:

```
struct record_key {
char *s;
enum keytype key, par;
int noval;
void (*vy_init)(void);
void* (*s2t)(char*);
char* (*t2s)(void*);
char* (*t2h)(void*);
int (*eq )(void*,void*);
};
```

Назначение полей структуры описано ниже:

- `char *s` – название ключа. Эта переменная может принимать значения `RE` (реакция), `DD` (измеряемая величина), `P` (частица) и др.
- `enum keytype key, par` – `id` (идентификаторы) ключа и его родителя
- `int noval` – эта переменная равна 0 для ключей `EXPERIMENT`, `REACTION-DATA` и др., и равна 1 для ключей, имеющих значение: `P(article)`, `RE(action)` и т.д.
- `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. Например, обозначение частицы π^-, `PI-`, переводится в переменную типа `struct type_P`.
- `char* (*t2s)(void*)` – адрес функции, переводящей переменную типа `struct type_`КЛЮЧ в строку. Например, внутреннее представление частицы π^- в виде `struct type_P` преобразуется в строку `PI-`.
- `char* (*t2h)(void*)` – адрес функции, вычисляющей уникальный "хэш" по переменной типа `struct type_`КЛЮЧ. Именно эти "хэши", а не текстовые ЗНАЧЕНИЯ записываются в хранилище.
- `int (*eq )(void*,void*)` – функция, возвращающая 1, если две переменные одного типа `struct type_KEY` равны, или 0, если они не равны.

Правила для ключей DataGuide заданы следующим образом (полный вариант правил приведен в Приложении Е.4):

```
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, NULL     ,s2t_RE,t2s_RE,t2h_RE,eq_RE},
{"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},
}
```

На данный момент реализованы правила для ключей `P` (частица) и `RE` (реакция). Реакции составлены из содержащихся в словаре PPDL частиц, но сами не содержатся в словаре, и поэтому указатель `vy_init` для ключа `RE` нулевой. Функция же `s2t_RE` принимает на вход адрес строки с текстовой записью реакции и преобразует ее во внутреннее представление, как будет описано ниже.

3.1.1 Правила работы с парами КЛЮЧ = ЗНАЧЕНИЕ

Порядок обработки пары `КЛЮЧ = ЗНАЧЕНИЕ` разобран ниже на примере ключа `P(article)`.

Правила обращения с ключом `P(article)` реализованы следующим образом. [2]

`vy_init_P()` – при старте базы определения частиц зачитываются из внешнего файла в следующем формате:

`id B L s c b t is_anti-particle is_heavy-nucleus class mass` I_3 `Q` *orcode* имя$_1$ [имя$_2$ ...],

[2] См. https://gitlab.ihep.su/zenin_o/reac/blob/scratch/dg/src/P/f.c или код в Приложении Е.8

где `id` – внутренний идентификатор частицы, `B ...t` – ее квантовые числа, *orcode* – целое число, по которому в оригинальной PPDS вычислялись квантовые числа частицы. Обозначения остальных полей самоочевидны. В качестве примера приведем словарную запись для K^{*+}:

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

Прочитанное определение частицы записывается в структуру следующего типа:

```
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;
};
```

1. Вызывается функция `s2t_КЛЮЧ(ЗНАЧЕНИЕ)`. Если словарь для `КЛЮЧА` определен, а это для ключа `P(article)` действительно так, и в нем присутствует `ЗНАЧЕНИЕ`, то функция возвращает `p` – адрес стуктуры типа `struct type_P`. В случае, если `ЗНАЧЕНИЕ` отсутствует в словаре, то возвращается нулевой адрес, и программа завершается с сообщением об ошибке чтения дампа.

2. Вычисляем "хэш" для `*p`: `hash = t2h_P(p)` и записываем строку `P { v:значение_hash }` (см. пример записи в Приложении Б) в файл в `base/records/текущая_запись`.

3. Ищем в файле `base/P/h` строку с `значение_hash`:

   - если такая строка нашлась, то есть такая частица уже встречалась при чтении дампа, то добавляем в файл `base/P/r` строку `hash текущая_запись`;
   - если строка не нашлась, то в конец файла `base/P/v` добавляем строку `значение_hash ЗНАЧЕНИЕ`, в конец файла `base/P/h` дописываем строку `значение_hash {адрес этой записи в файле base/P/v}` и добавляем строку `значение_hash текущая_запись` в файл `base/P/r`.

Эта последовательность работает для любых пар `КЛЮЧ = ЗНАЧЕНИЕ` с функциями из приведенной выше таблицы `struct record_key keys[]`. В результате чтения записи из дампа изменяется следующий набор файлов в хранилище:

- `base/records/текущая_запись` – содержит текущую запись в формате, приведенном в Приложении Б

- `base/КЛЮЧ/v` – содержит пары `hash ЗНАЧЕНИЕ` для всех пар `КЛЮЧ = ЗНАЧЕНИЕ`, содержащихся в данной записи
- `base/КЛЮЧ/h` – для каждого `hash` содержит адрес строки `hash ЗНАЧЕНИЕ` в файле `base/КЛЮЧ/v`
- `base/КЛЮЧ/r` – файл индекса, содержит строки `hash текущая_запись` для всех hash, вычисленных для пар `КЛЮЧ = ЗНАЧЕНИЕ` из текущей записи.

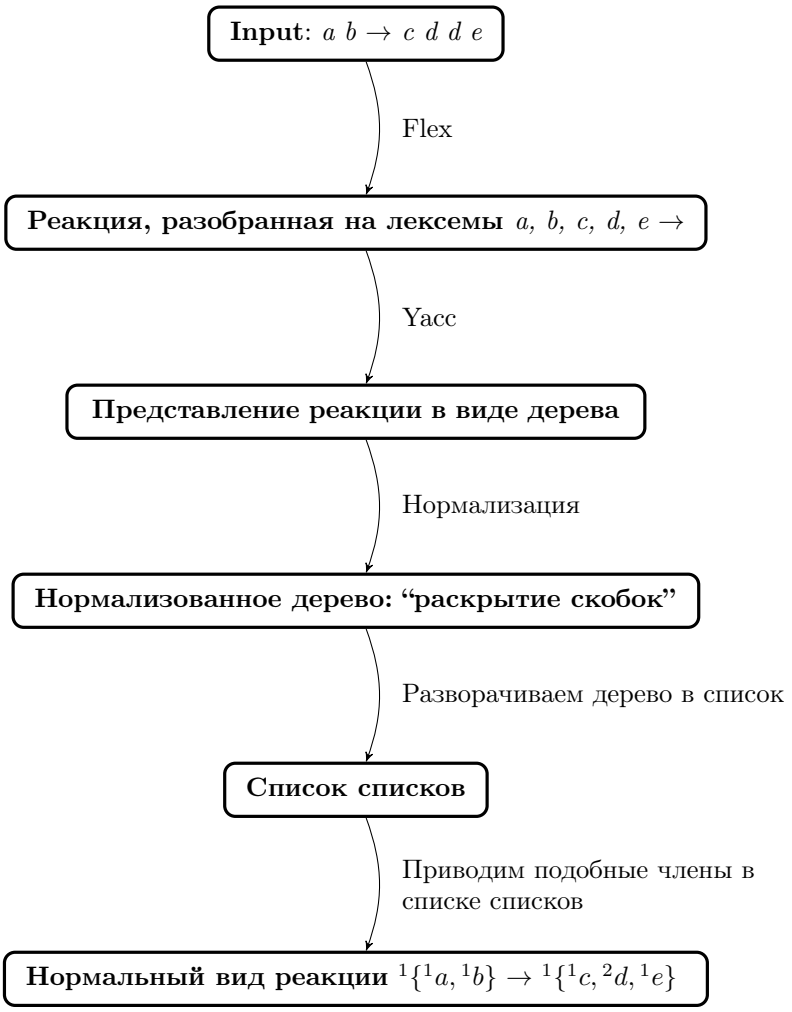

Рис. 3.1: Упрощенная последовательность обработки реакции.

3.1.2 Разбор записанных на языке PPDL реакций

Порядок обработки пары `RE = ЗНАЧЕНИЕ` отличается от разобранного выше случая `P = ЗНАЧЕНИЕ` тем, что реакция (`ЗНАЧЕНИЕ`) подвергается синтаксическому разбору, а не ищется в словаре. Порядок обработки реакции (Рис. **??**) кратко разобран ниже.

Реакция считывается из дампа в текстовом формате вида `начальное_состояние` $\rightarrow$ `конечное_состояние`. Например, если в эксперименте изучались конечные состояния $nf_1(1285)$ и $n\eta'$ рожденные в $\pi^- p$ столкновениях, то запись реакции на языке PPDL выглядит следующим образом:

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

В результате разбора записи реакции на *лексемы* – частицы и обозначения логических операций (см. Приложение Д.1) средствами лексического анализатора LEX, и последующего синтаксического разбора по грамматическим правилам, записанным на языке YACC (см. Приложение Д.2), начальное и конечное состояния реакции преобразуются в *деревья*. Исходное дерево для конечного состояния данной реакции показано на Рис. 3.2.

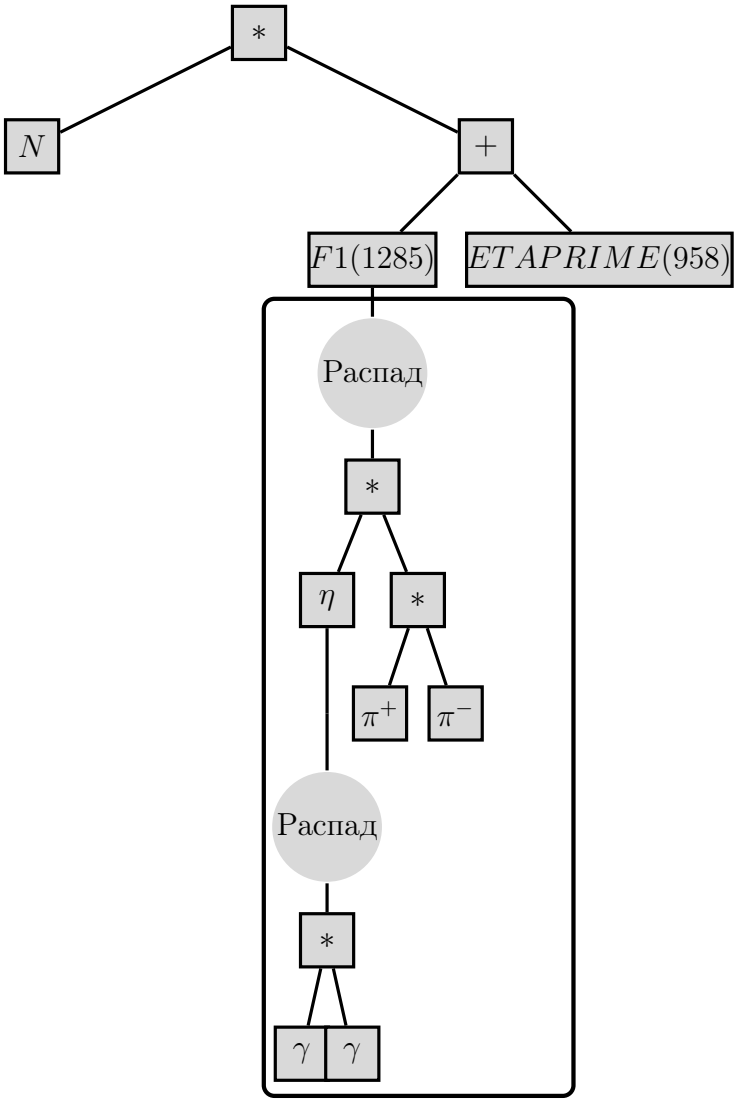

Рис. 3.2: Исходное дерево для конечного состояния $n(f_1(1285)(\rightarrow \eta(\rightarrow 2\gamma)\pi^+\pi^-)+\eta')$

Далее дерево преобразуется по правилам раскрытия скобок (см. функцию `norm_tree()` в Приложении Ж.8), и принимает вид, показанный на Рис. 3.3.

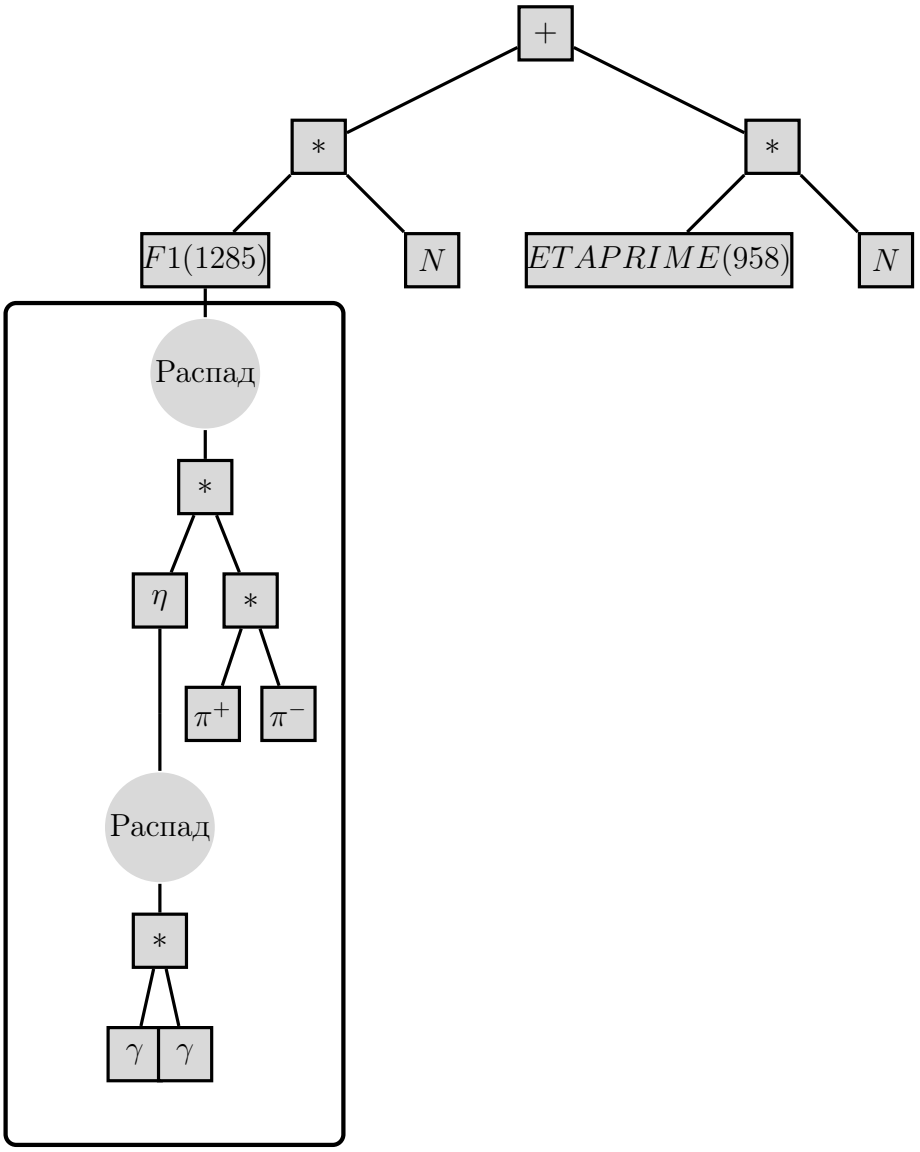

Рис. 3.3: Дерево конечного состояния $n(f_1(1285)(\to \eta(\to 2\gamma)\pi^+\pi^-) + \eta')$ после раскрытия скобок.

После раскрытия скобок '*'-узлы дерева не могут находиться выше узлов '+' и '-'. [3] Следовательно, такое *развернутое* дерево может быть представлено в виде *списка списков*, то есть списка, элементами которого будут конечные состояния, то есть простые списки частиц. Преобразование ветви дерева, содержащей только частицы и '*'-узлы, в список производится функцией `tree2list()` (см. Приложение Ж.8). Каждый полученный так список частиц сортируется по их `id`, и в нем приводятся подобные члены.[4] Для вышеприведенной реакции *список списков*

[3]Операция объединения в список обозначается '*'. Символ '+' обозначает логическое *или*, символ '-' обозначает *за исключением*.

[4]Например, список вида $\{p_3, p_1, p_2, p_1\}$ преобразуется в $\{^2p_1, p_2, p_3\}$.

имеет вид:

$$\left\{{}^1\left\{{}^1n, {}^1f_1(1285)\left\langle\left\{{}^1\pi^-, {}^1\pi^+, {}^1\eta\left\langle{}^2\gamma\right\rangle\right\}\right\rangle\right\}, {}^1\left\{{}^1n, {}^1\eta'\right\}\right\}, \tag{3.1}$$

где обозначения вида ${}^1\{\ldots\}$, ${}^2\gamma$ показывают, сколько раз данный элемент входит в свой список, а цепочки распадов показаны как $\langle\ldots\rangle$. *Список списков* в свою очередь сортируется, и в нем также приводятся подобные члены. Полученный таким образом *список списков* определяется однозначно для всех возможных эквивалентных записей одной и той же реакции и называется *нормальным видом* реакции. Например, конечное состояние реакции, записанной в виде

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

дает тот же нормальный вид (3.1).

Точное определение перечисленных операций дается программным кодом, приведенным в Приложении Ж.8.

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.

Программный код доступен в Git-репозитории ИФВЭ:

`https://gitlab.ihep.su/zenin_o/reac/tree/scratch`

Следует отметить, что в данной работе 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.

Литература

[1] V.V. Ezhela, “Particle Physics Data Systematisation at IHEP”, Comp. Phys. Commun. **33** (1984) 225;
“Руководство по BDMS/4”, ИФВЭ, 1997,
`http://hermes.ihep.su:8001/bdms4manual.ps`

[2] См. главу 3 в “Руководстве по BDMS/4” [1]

[3] Free Software Foundation, Inc, “GCC, the GNU Compiler Collection”, `https://gcc.gnu.org/`

[4] FLEX (the Fast Lexical Analyzer), `https://www.gnu.org/software/flex/`

[5] Free Software Foundation, Inc., GNU Bison, `https://www.gnu.org/software/bison/`

Приложения

A Структура записи DataGuide

```
DEFINE
FILE=DOCUMENT;
DE. = SC       ; TY=CHAR;   KEY; LENG=7; IPRO=EXTERNAL; KPRO=INTERNAL;
DE. = PDGSC    ; TY=CHAR; PAR=SC; KEY; LENG=4; IPRO=EXTERNAL; KPRO=INTERNAL;
DE. = IRN      ; TY=CHAR;   KEY; SYN=SLAC-HEP-link; LENG=3;
DE. = IRNW     ; TY=CHAR; PAR=IRN; SYN=IRN-WARNING; KEY; LENG=2;
DE. = R        ; TY=CHAR;   KEY; LENG=10; KPRO=INTERNAL;
DE. = COD;     ; TY=CHAR; PAR=R; SYN=RCO;CODEN; KEY; LENG=5;
DE. = TY       ; TY=CHAR; PAR=R; KEY; LENG=1; IPRO=EXTERNAL; KPRO=INTERNAL;
DE. = D        ; TY=CHAR; PAR=R; KEY; LENG=1; IPRO=EXTERNAL; KPRO=INTERNAL;
DE. = KTO      ; TY=CHAR;   KEY; LENG=2; IPRO=EXTERNAL; KPRO=INTERNAL;
DE.=  KRD      ; TY=CHAR;   KEY; LENG=1; IPRO=EXTERNAL; KPRO=INTERNAL;
DE. = CONF     ; TY=CHAR;   KEY; LENG=10;
DE. = AUTHORS  ; TY=NODE; SYN=AUTH;
DE. = A        ; TY=CHAR;PAR=AUTHORS; KEY; LENG=5; KPRO=EXTERNAL;
DE. = SPOKESMAN;TY=NODE; PAR=A;SYN=SPOKES;SPOK; KEY; LENG=0;
DE. = SPP      ; TY=CHAR;   KEY; LENG=5; KPRO=EXTERNAL;
DE. = I        ; TY=CHAR;PAR=AUTHORS;IPRO=EXTERNAL; KEY; LENG=1;
DE. = ILBL     ; TY=CHAR;PAR=I; KEY; LENG=1;
DE. = NA       ; TY=INTE;SYN=NUMBER_OF_COAUTHORS; KEY; LENG=1;
DE. = T        ; TY=CHAR;   KEY; LENG=1; IPRO=EXTERNAL; OPRO=EXTERNAL;
DE. = TW       ; TY=CHAR;PAR=T;KEY;LENG=5;VIRT;OPRO=EXTERNAL;
DE. = TITLE-TEX; TY=CHAR;PAR=T;SYN=TIT; KEY;
DE. = TA       ; TY=CHAR;PAR=T; KEY; LENG=1; IPRO=EXTERNAL;
DE. = TA-TEX   ; TY=CHAR;PAR=TA;SYN=TAT; KEY; LENG=0; IPRO=EXTERNAL;
DE. = ABS      ; TY=CHAR;   KEY;  IPRO=EXTERNAL;
DE. = ABS-TEX  ; TY=CHAR; PAR=ABS;SYN=ABST; IPRO=EXTERNAL;     KEY; LENG=0;
DE. = DK       ; TY=CHAR; SYN=DESY-KEY-WORD; KEY; LENG=1;
DE. = L        ; TY=CHAR;   KEY; LENG=1;
DE. = CIT      ; TY=CHAR; SYN=C; KEY; LENG=10;
DE. = CCOD     ; TY=CHAR; SYN=CODEN-CIT; KEY; LENG=5;
DE. = CB       ; TY=CHAR;   KEY; LENG=4;
DE. = CB-TEX   ; TY=CHAR; PAR=CB;SYN=CBT; KEY; LENG=4;
DE. = URL      ; TY=CHAR;   KEY; LENG=10;
DE. = PPF; TY=CHAR;         KEY; LENG=2;
DE. = OR       ; TY=CHAR;   KEY; LENG=4;
DE. = SB       ; TY=CHAR;   KEY; LENG=1; IPRO=EXTERNAL; KPRO=EXTERNAL;
DE. = AV       ; TY=CHAR;   KEY; LENG=2;
DE. = COMP     ; TY=CHAR;   KEY; LENG=5;
DE. = ES       ; TY=CHAR;   KEY; LENG=1;
DE. = EXPERIMENT; TY=NODE; SYN=EXP;
DE. = AC   ; TY=CHAR;PAR=EXPERIMENT; KEY; LENG=4; IPRO=EXTERNAL; KPRO=INTERNAL;
DE. = DE       ; TY=CHAR;PAR=EXPERIMENT; IPRO=EXTERNAL; KEY; LENG=4;
DE. = PR       ; TY=CHAR;PAR=EXPERIMENT; IPRO=EXTERNAL; KEY; LENG=6;
DE. = COL      ; TY=CHAR;PAR=EXPERIMENT; KEY; LENG=14;
```

```
DE. = RR       ; TY=CHAR;PAR=EXPERIMENT; KEY; LENG=10;
DE. = RCOD    ; TY=CHAR;PAR=RR; SYN = CODEN-RR; KEY; LENG=5;
DE. = PUBLISHED-PAPERS; SYN=RRLBL;TY=CHAR; KEY; LENG=10;
DE. = PL       ; TY=CHAR;PAR=EXPERIMENT; IPRO=EXTERNAL; KEY; LENG=1;
DE. = CE       ; TY=CHAR;PAR=EXPERIMENT; IPRO=EXTERNAL; KEY; LENG=10;
DE. = CE-TEX   ; TY=CHAR;PAR=EXPERIMENT; SYN=CET; KEY; LENG=1;
DE. = CD       ; TY=CHAR;PAR=EXPERIMENT; IPRO=EXTERNAL; KEY; LENG=10;
DE. = CD-TEX   ; TY=CHAR;PAR=EXPERIMENT; SYN=CDT; KEY; LENG=1;
DE. = REAC-DATA;TY=NODE;SYN=REDA;PAR=EXPERIMENT; KEY; LENG=0;
DE. = RE       ; TY=CHAR;PAR=REAC-DATA;IPRO=CONVERT ;KEY;LENG=30;KPRO=INTERNAL;
                                    OPRO=CONVERT; QPRO=CONVERT;
DE. = FSP      ; TY=CHAR;PAR=REAC-DATA; KEY;LENG=1; VIRT; QPRO=CONVERT;
DE. = BP       ; TY=CHAR;PAR=REAC-DATA; KEY;LENG=1; VIRT; QPRO=CONVERT;
DE. = TP       ; TY=CHAR;PAR=REAC-DATA; KEY;LENG=1; VIRT; QPRO=CONVERT;
DE. = B        ; TY=CHAR;PAR=REAC-DATA; KEY; LENG=5;
DE. = PLAB     ; TY=REAL;PAR=REAC-DATA; KEY; LENG=2;
DE. = ECM      ; TY=REAL;PAR=REAC-DATA; KEY; LENG=2;
DE. = Q2       ; TY=REAL;PAR=REAC-DATA; KEY; LENG=2;
DE. = W2       ; TY=REAL;PAR=REAC-DATA; KEY; LENG=2;
DE. = NU       ; TY=REAL;PAR=REAC-DATA; KEY; LENG=2;
DE. = DD       ; TY=CHAR;PAR=REAC-DATA;IPRO=EXTERNAL;KEY; LENG=3;KPRO=INTERNAL;
DE. = INR      ; TY=CHAR;PAR=REAC-DATA; KEY; LENG=0;
DE. = PART-PROP;TY=NODE; SYN=PAPR;  PAR=EXPERIMENT; KEY; LENG=0;
DE. = P        ; TY=CHAR;PAR=PART-PROP;IPRO=CONVERT ;KEY; LENG=1;KPRO=INTERNAL;
                                    OPRO=INTERNAL; QPRO=CONVERT ;
DE. = PP       ; TY=CHAR;PAR=PART-PROP;IPRO=EXTERNAL;KEY; LENG=1;KPRO=INTERNAL;
DE. = INP      ; TY=CHAR;PAR=PART-PROP; KEY; LENG=0;
DE. = CR       ; TY=CHAR;SYN=COMMENT_RECORD; KEY; LENG=4;
DE. = CR-TEX   ; TY=CHAR;SYN=CRT;COMM_REC_TEX; KEY; LENG=1;
DE. = CRW      ; TY=CHAR;PAR=CR;VIRT;OPRO=EXTERNAL; KEY; LENG=1;
DE. = INDEX    ; TY=CHAR; KEY; LENG=10;
DE. = RD       ; TY=CHAR; SYN=RD-link; KEY; LENG=6;
DE. = CS       ; TY=CHAR; SYN=CS-link;CS_base;KEY; LENG=6;
DE. = RPP      ; TY=CHAR; SYN=RPP-link;KEY; LENG=6;
DE. = DEN      ; TY=CHAR; KEY; LENG=2;
DE. = DCH      ; TY=CHAR; KEY; LENG=2;
DE. = ACTION   ; TY=CHAR; VIRT; SYN=SLAC;
DE. = QUESTION; TY=CHAR; SYN=QUES; KEY; LENG=8;
DE. = ANSWER   ; TY=CHAR; PAR=QUESTION;SYN=ANSW; KEY; LENG=5;
DE. = ERROR    ; TY=CHAR; SYN=ERRO; KEY; LENG=5;
**
COMPILE
STOP
```

Б Запись в хешах

```
(null) {
  SC { v:62464478336d75474439775453596b71714b66355431 }
  IRN { v:3375655378743654597a3569754d6a765558356a7331 }
  R { v:4a2f67524c6b2e6e7a30573939756a3542535a773531
```

```
    COD { v:5279767236573870434b354c48532f7341527269422e }
    TY  { v:4b6148526847614879664l4e556d3167356c71346c2f }
    D   { v:437a4859347032352e65714f6f6461526a3763795631 }
  }
  AUTHORS {
    A { v:477131383559574a4d4d4e754d46447634796451492e   }
  ...
  }
  T { v:6c7166675537613049324d4e6d5057636865426b6f2e
    TITLE-TEX { v:465a4f7539546a2f5637303033676145757152335l2f }
  }
  ABS { v:54387447583075344f434165745132634f3771494c2e }
  ...
  EXPERIMENT {
    AC { v:6c706e6a537a5a41496c3030356e656d73336274442f }
    DE { v:386d766b3774353744527033546845773O6a2f4d2f2f }
    PR { v:4b386a303638424a7258693256313468466e7a4e702f }
    COL { v:3953522f726b35684e745158475a37683346796c4331 }
    REAC-DATA {
      RE { v:4863714f504955386c6c552e71757752486778672e30 }
      B { v:546e6f3538422f45466b7048597830437a316f432f2e }
      DD { v:4b794c3659796d6956592f7234676f42515073616931 }
    }
    REAC-DATA {
      RE { v:4b5a7a496270494f64336b36314d435467656b496131 }
      B { v:546e6f3538422f45466b7048597830437a316f432f2e }
      DD { v:75714362655662526d456a5a48574c6936524c49562f }
    }
    ..........
    PART-PROP {
      P { v:000000000000000000000000000000000000000091d }
      P { v:000000000000000000000000000000000000000062 }
      P { v:000000000000000000000000000000000000000025a }
      PP { v:4b794c3659796d6956592f7234676f42515073616931 }
      PP { v:79496f39304437736561527a70772e55644741574130 }
      PP { v:647a34425a436842445647626966426359424e395731 }
    }
    PART-PROP {
      P { v:0000000000000000000000000000000000000000fb }
      PP { v:3832772f5259383565337959456d4b2e7772426a632f }
    }
  }
}
```

B Полная запись struct record_key keys[]

```
struct record_key
keys[] = {

{"SC"              ,_SC_                    ,_RECORD_    , 0, 0,0,0,0,0},
```

```
{"PDGSC"           ,_PDGSC_                ,_SC_         , 0, 0,0,0,0,0},
{"IRN"             ,_IRN_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"IRNW"            ,_IRNW_                 ,_IRN_        , 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},
{"KTO"             ,_KTO_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"KRD"             ,_KRD_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"CONF"            ,_CONF_                 ,_RECORD_     , 0, 0,0,0,0,0},
{"AUTHORS"         ,_AUTHORS_              ,_RECORD_     , 1, 0,0,0,0,0},
{"A"               ,_A_                    ,_AUTHORS_    , 0, 0,0,0,0,0},
{"SPOKESMAN"       ,_SPOKESMAN_            ,_A_          , 1, 0,0,0,0,0},
{"SPP"             ,_SPP_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"I"               ,_I_                    ,_AUTHORS_    , 0, 0,0,0,0,0},
{"ILBL"            ,_ILBL_                 ,_I_          , 0, 0,0,0,0,0},
{"NA"              ,_NA_                   ,_RECORD_     , 0, 0,0,0,0,0},
{"T"               ,_T_                    ,_RECORD_     , 0, 0,0,0,0,0},
{"TW"              ,_TW_                   ,_T_          , 0, 0,0,0,0,0},
{"TITLE-TEX"       ,_TITLE_TEX_            ,_T_          , 0, 0,0,0,0,0},
{"TA"              ,_TA_                   ,_T_          , 0, 0,0,0,0,0},
{"TA-TEX"          ,_TA_TEX_               ,_TA_         , 0, 0,0,0,0,0},
{"ABS"             ,_ABS_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"ABS-TEX"         ,_ABS_TEX_              ,_ABS_        , 0, 0,0,0,0,0},
{"DK"              ,_DK_                   ,_RECORD_     , 0, 0,0,0,0,0},
{"L"               ,_L_                    ,_RECORD_     , 0, 0,0,0,0,0},
{"CIT"             ,_CIT_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"CCOD"            ,_CCOD_                 ,_RECORD_     , 0, 0,0,0,0,0},
{"CB"              ,_CB_                   ,_RECORD_     , 0, 0,0,0,0,0},
{"CB-TEX"          ,_CB_TEX_               ,_CB_         , 0, 0,0,0,0,0},
{"URL"             ,_URL_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"PPF"             ,_PPF_                  ,_RECORD_     , 0, 0,0,0,0,0},
{"OR"              ,_OR_                   ,_RECORD_     , 0, 0,0,0,0,0},
{"SB"              ,_SB_                   ,_RECORD_     , 0, 0,0,0,0,0},
{"AV"              ,_AV_                   ,_RECORD_     , 0, 0,0,0,0,0},
{"COMP"            ,_COMP_                 ,_RECORD_     , 0, 0,0,0,0,0},
{"ES"              ,_ES_                   ,_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},
{"RCOD"            ,_RCOD_                 ,_RR_         , 0, 0,0,0,0,0},
{"PUBLISHED-PAPERS",_PUBLISHED_PAPERS_     ,_RECORD_     , 0, 0,0,0,0,0},
{"PL"              ,_PL_                   ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"CE"              ,_CE_                   ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"CE-TEX"          ,_CE_TEX_               ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"CD"              ,_CD_                   ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"CD-TEX"          ,_CD_TEX_               ,_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},
{"FSP"             ,_FSP_               ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"BP"              ,_BP_                ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"TP"              ,_TP_                ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"B"               ,_B_                 ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"PLAB"            ,_PLAB_              ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"ECM"             ,_ECM_               ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"Q2"              ,_Q2_                ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"W2"              ,_W2_                ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"NU"              ,_NU_                ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"DD"              ,_DD_                ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"INR"             ,_INR_               ,_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},
{"INP"             ,_INP_               ,_PART_PROP_ , 0, 0,0,0,0,0},
{"CR"              ,_CR_                ,_RECORD_    , 0, 0,0,0,0,0},
{"CR-TEX"          ,_CR_TEX_            ,_RECORD_    , 0, 0,0,0,0,0},
{"CRW"             ,_CRW_               ,_CR_        , 0, 0,0,0,0,0},
{"INDEX"           ,_INDEX_             ,_RECORD_    , 0, 0,0,0,0,0},
{"RD"              ,_RD_                ,_RECORD_    , 0, 0,0,0,0,0},
{"CS"              ,_CS_                ,_RECORD_    , 0, 0,0,0,0,0},
{"RPP"             ,_RPP_               ,_RECORD_    , 0, 0,0,0,0,0},
{"DEN"             ,_DEN_               ,_RECORD_    , 0, 0,0,0,0,0},
{"DCH"             ,_DCH_               ,_RECORD_    , 0, 0,0,0,0,0},
{"ACTION"          ,_ACTION_            ,_RECORD_    , 0, 0,0,0,0,0},
{"QUESTION"        ,_QUESTION_          ,_RECORD_    , 0, 0,0,0,0,0},
{"ANSWER"          ,_ANSWER_            ,_QUESTION_  , 0, 0,0,0,0,0},
{"ERROR"           ,_ERROR_             ,_RECORD_    , 0, 0,0,0,0,0},

{0                 ,0                   ,0           , 0, 0,0,0,0,0}

};
```

Γ vy_init_P()

```
void
vy_init_P (void) {
char *vy_base = 0, *vy_fn = 0, c, *p, *n1, *n2, *n2_min;
FILE *vy_f = 0;
int i,j, j_min, n;
long d, d_min;
struct type_P p_tmp;
if (_P) {
fprintf(stderr,"ERROR: vy_init_P(): _P[] is already initialized [EXIT]\n");
exit(1);
}
if (!(vy_base = getenv("PPDS_VY"))) {
fprintf(stderr,"ERROR: vy_init_P(): PPDS_VY env. pointing to the "
       "vocabulary directory is not defined [EXIT]\n"
);
exit(1);
}
vy_fn = (char*)malloc(sizeof(char)*(strlen(vy_base) + 1 + 1 + 1 + 2 + 1));
sprintf(vy_fn,"%s/P/vy",vy_base);
printf("INFO: reading particle definitions from PPDS_VY = %s\n",vy_fn);
if (!(vy_f = fopen(vy_fn,"r"))) {
fprintf(stderr,"ERROR: vy_init_P(): can't open %s [EXIT]\n",vy_fn);
exit(1);
}
_P = (struct type_P*) malloc(sizeof(struct type_P) * (n = 256) );
  /* particle format: id bar lep S C B T anti heavy_nuc class m I3 Q orcode name: */
for (n_P = 0;
     fscanf(vy_f,"%d %d %d  %d %d %d %d  %d %d %d  %lf %lf %lf %s",
     &_P[n_P].id,
     &_P[n_P].bar,
     &_P[n_P].lep,
     &_P[n_P].S,
     &_P[n_P].C,
     &_P[n_P].B,
     &_P[n_P].T,
     &_P[n_P].anti,
     &_P[n_P].heavy_nuc,
     &_P[n_P].class,
     &_P[n_P].m,
     &_P[n_P].I3,
     &_P[n_P].Q,
      _P[n_P].orcode
    ) == 14;
     n_P++) {
_P[n_P].val = (char**) malloc(sizeof(char*) * 256);
_P[n_P].nval = 0;
_P[n_P].val0sum = 0;
_P[n_P].val[_P[n_P].nval] = 0;
while ((c = fgetc(vy_f)) != EOF) {
```

```
switch (c) {
  case ' ' :
  case '\t':
  case '\n':
if (_P[n_P].val[_P[n_P].nval]) {
*p = '\0';
_P[n_P].val[_P[n_P].nval] = (char*) realloc (
_P[n_P].val[_P[n_P].nval],
  sizeof(char) * (strlen(_P[n_P].val[_P[n_P].nval]) + 1)
 );
_P[n_P].nval ++;
_P[n_P].val[_P[n_P].nval] = 0;
}
if (c == '\n')
goto out;
break;
  default:
if (!_P[n_P].val[_P[n_P].nval])
p = _P[n_P].val[_P[n_P].nval] = (char*)malloc(sizeof(char)*64);
*p++ = c;
break;
}

}
out:
_P[n_P].val = (char**) realloc(_P[n_P].val, sizeof(char*) * _P[n_P].nval);
if (_P[n_P].nval)
_P[n_P].val0sum = s2sum(_P[n_P].val[0]);
if(_DBG_)
printf("DBG: particle read: ");
if (_P[n_P].nval == 0)
printf("WARNING: no name for particle: ");
if (_DBG_ || _P[n_P].nval == 0) {
for (i = 0; i < _P[n_P].nval; i++)
printf("%s ",_P[n_P].val[i]);
printf(" or: %s bar: %d lep: %d S,C,B,T: %d %d %d %d "
      "p/anti-p: %d hvy/N: %d class: %d"
      "m: %lf I3: %lf Q: %lf\n",
      _P[n_P].orcode,
      _P[n_P].bar,
      _P[n_P].lep,
      _P[n_P].S,
      _P[n_P].C,
      _P[n_P].B,
      _P[n_P].T,
      _P[n_P].anti,
      _P[n_P].heavy_nuc,
      _P[n_P].class,
      _P[n_P].m,
      _P[n_P].I3,
      _P[n_P].Q
```

```
     );
}
if (n_P + 2 > n)
_P = (struct type_P*)realloc(_P, sizeof(struct type_P) * (n *= 2));
}
_P = (struct type_P*)realloc(_P, sizeof(struct type_P) * n_P);
printf("INFO: %d particles in the vocabulary\n",n_P);
_DBG_&&printf("DBG: sorting by name[0] ...\n");
for (i = 0; i < n_P; i++) {
d_min = _P[i].val0sum;
n1 = _P[i].nval ? _P[i].val[0] : 0;
j_min = i;
for (j = i+1; j < n_P; j++) {
d =  _P[j].val0sum;
n2 = _P[j].nval ? _P[j].val[0] : 0;
if (d < d_min) {
d_min = d;
j_min = j;
n2_min = n2;
}
}
if (j_min != i) {
p_tmp = _P[j_min];
_P[j_min] = _P[i];
_P[i] = p_tmp;
}
}
if(_DBG_) {
printf("DBG: particles sorted by name[0] ....... \n");
for (i = 0; i < n_P; i++) {
for (j = 0; j < _P[i].nval; j++)
printf("%s ",_P[i].val[j]);
printf(" or: %s bar: %d lep: %d S,C,B,T: %d %d %d %d "
      "p/anti-p: %d hvy/N: %d class: %d"
      "m: %lf I3: %lf Q: %lf\n",
      _P[i].orcode,
      _P[i].bar,
      _P[i].lep,
      _P[i].S,
      _P[i].C,
      _P[i].B,
      _P[i].T,
      _P[i].anti,
      _P[i].heavy_nuc,
      _P[i].class,
      _P[i].m,
      _P[i].I3,
      _P[i].Q
      );
}
printf("................................................\n");
```

```
}

return;
}
```

Д Разбор записанной на PPDL реакции

Д.1 Лексический разбор средствами LEX

LEX, он же Flex (Fast Lexical Analyzer) – генератор лексических анализаторов, то есть программ, выполняющих лексический анализ. Лексический анализ – процесс аналитического разбора входной последовательности символов на распознанные группы -— лексемы, с целью получения на выходе идентифицированных последовательностей. LEX-файл имеет следующую структуру:

Блок определений. Содержит макросы и заголовочные файлы.
%%
Блок правил. Содержит описание лексем.
%%
Блок кода на Си

Получая на вход текст, на выход он выдает код анализатора в виде функции на языке Си.
Таким образом, flex выделяет лексемы внутри реакции. Реакция составлена из следующих лексем:

- PRT – частица
- NUM – целое неотрицательное число
- ';' – конец записи
- '+' – логическое ИЛИ
- '-' – логическое исключение
- '(', ')' – группировка, например, в выражении вида $a(b+c)d$
- '<', '>' – выделяет цепочку распада, например $a\langle b\langle cd\rangle e\rangle$
- ' ' – операция объединения в список

В случае появления на входе последовательности символов, не соответствующей ни одному из приведенных ниже правил, программа возвращает ошибку.

```
[ \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); }
```

Полный программный код приведен в Приложении Ж.4.

Д.2 Синтаксический разбор средствами YACC

Генератор синтаксических анализаторов YACC (Yet Another Compiler of Compilers) [5] принимает на вход последовательность лексем, и выполняет операции над ними в соответствии с грамматическими правилами, заданными следующим образом:

```
%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; }
  ;
%%
```

Вызовы функции `new_node()` создают узлы *дерева реакции* (см. полный программный код в Приложении Ж.5).

E Программа для чтения дампа DataGuide и записи разобранных записей в хранилище

E.1 Makefile

```
CFLAGS = -g -O0

INCFLAGS = -I. -I../../src/util

LIBS = -lcrypt

l: lex.yy.o fdt.o P/f.o tree2base.o ../../src/util/s2hash.o
        cc -o $@ $^ $(LIBS)

l.l: settings.h

lex.yy.c: l.l tree.h settings.h
        lex $<

../../src/util/s2hash.o: ../../src/util/s2hash.c
        cd ../../src/util && make s2hash.o CFLAGS="$(CFLAGS)"

P/f.o: P/f.c P/P.h
        cd P/ && make f.o CFLAGS="$(CFLAGS)"

%.o: %.c settings.h
        cc $(CFLAGS) $(INCFLAGS) -c -o $@ $<

clean:
        rm -f l *.a *.o lex.yy.c  *~
        for d in */ ; do cd $$d ; make clean ; cd .. ; done
```

E.2 tree.h

```
enum keytype {
_UNKNOWN_               ,
_NOKEY_                 ,
_RECORD_                ,
_SC_                    ,
_PDGSC_                 ,
_IRN_                   ,
_IRNW_                  ,
_R_                     ,
_COD_                   ,
_TY_                    ,
_D_                     ,
_KTO_                   ,
_KRD_                   ,
_CONF_                  ,
_AUTHORS_               ,
_A_                     ,
_SPOKESMAN_             ,
_SPP_                   ,
_I_                     ,
_ILBL_                  ,
_NA_                    ,
_T_                     ,
_TW_                    ,
_TITLE_TEX_             ,
_TA_                    ,
_TA_TEX_                ,
_ABS_                   ,
_ABS_TEX_               ,
_DK_                    ,
_L_                     ,
_CIT_                   ,
_CCOD_                  ,
```

```
_CB_                    ,
_CB_TEX_                ,
_URL_                   ,
_PPF_                   ,
_OR_                    ,
_SB_                    ,
_AV_                    ,
_COMP_                  ,
_ES_                    ,
_EXPERIMENT_            ,
_AC_                    ,
_DE_                    ,
_PR_                    ,
_COL_                   ,
_RR_                    ,
_RCOD_                  ,
_PUBLISHED_PAPERS_      ,
_PL_                    ,
_CE_                    ,
_CE_TEX_                ,
_CD_                    ,
_CD_TEX_                ,
_REAC_DATA_             ,
_RE_                    ,
_FSP_                   ,
_BP_                    ,
_TP_                    ,
_B_                     ,
_PLAB_                  ,
_ECM_                   ,
_Q2_                    ,
_W2_                    ,
_NU_                    ,
_DD_                    ,
_INR_                   ,
_PART_PROP_             ,
_P_                     ,
_PP_                    ,
_INP_                   ,
_CR_                    ,
_CR_TEX_                ,
_CRW_                   ,
_INDEX_                 ,
_RD_                    ,
_CS_                    ,
_RPP_                   ,
_DEN_                   ,
_DCH_                   ,
_ACTION_                ,
_QUESTION_              ,
_ANSWER_                ,
_ERROR_
};

struct record_key {
char *s;
enum keytype key, par;
int  noval;
void  (*vy_init)(void);
void* (*s2t)(char*);
char* (*t2s)(void*);
char* (*t2h)(void*);
int   (*eq )(void*,void*);
};

struct node {
enum keytype type;
char **val;
int   nval;
struct node **ch, *par;
int   nch;
};


char*
```

```
k2s(int k);

int
k2noval(int k);

enum keytype
k2par(int k);

int
k2ik (int k);

int
s2k(char *s);

int
s2ik(char *s);
```

E.3 l.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "tree.h"

#include "settings.h"

int rec_loaded;

char buf[65536], *p, *py;

enum { _val_, _inkey_, _ex_, _er_ } st;

int mult_val;

enum keytype key, key_par;

extern
struct record_key
keys[];

struct node
*no_root, *no_cur, *no_par;

struct node*
new_node(enum keytype type, struct node *par) {
struct node *no = (struct node*)malloc(sizeof(struct node));
no->type = type;
no->par = par;
no->ch = NULL;
no->nch = 0;
no->val = NULL;
no->nval = 0;
if (par) {
par->ch = (struct node**)realloc(par->ch, sizeof(struct node*) * (++ par->nch));
par->ch[ par->nch - 1 ] = no;
}
return no;
}

void
val_node(struct node* no, char* val) {
if (!no) {
fprintf(stderr,"ERROR: val_node(NULL, %s) called. EXIT\n", val);
exit(1);
}
no->val = (char**)realloc(no->val, sizeof(char*) * (no->nval + 1));
if (val)
while (*val == ' ' || *val == '\t' || *val == '\n')
val++;
no->val[ no->nval ] = (char*)malloc(sizeof(char) * (strlen(val) + 1));
strcpy(no->val[ no->nval++ ], val );
```

```
return;
}

void
free_tree (struct node* no) {
if (!no)
return;
if (no->val) {
for (; no->nval > 0; no->nval--)
free((void*)no->val[ no->nval - 1 ]);
free((void*)no->val);
}
if (no->ch) {
for (; no->nch > 0; no->nch --)
free_tree(no->ch[ no->nch - 1 ]);
free((void*)no->ch);
}
free((void*)no);
return;
}

void
print_tree (struct node *no, int ident) {
int i, ik;
char *h;
for (i = 0; i < ident; i++)
putchar(' ');
if (!no) {
printf("[NULL]\n");
return;
}
if (k2noval(no->type)) {
printf("%s;\n",k2s(no->type));
}
else {
ik = k2ik(no->type);
printf("%s = ", keys[ik].s);
for (i = 0; i < no->nval; i++) {
printf("%s", no->val[i]);
if (keys[ik].s2t && keys[ik].t2h) {
h = keys[ik].t2h( keys[ik].s2t(no->val[i]) );
printf(" { %s } ", h ? h : "<<<<<<<<<<<<<< ERROR <<<<<<<<<<<<<<<<<<<<<<<" );
}
printf(";");
}
printf("\n");
}
for (i = 0; i < no->nch; i++)
print_tree(no->ch[i], ident+1);
return;
}




struct node*
find_par(enum keytype key, struct node* no) {
if (!no)
return NULL;
if (no->type == k2par(key))
return no;
else
return find_par(key,no->par);
}


%}

%%

SC|PDGSC|IRN|IRNW|R|COD|TY|D|KTO|KRD|CONF|AUTHORS|A|SPOKESMAN|SPP|I|ILBL|NA|T|TW|TITLE-TEX|TA|TA-TEX|ABS|ABS-TEX|DK|L|CIT|CCOD|C
/* printf("dbg: yytext: %s\n",yytext);*/
switch (st) {
case _val_:
for (py = yytext; *py; *p++ = *py++);
```

```
break;
case  _ex_ :
case  _er_ :
if ((key = s2k(yytext)) == _UNKNOWN_) {
fprintf(stderr,"ERROR: unknown key: %s. EXIT\n",yytext);
printf("ERROR: ^^^^^^^^ broken record follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
}
mult_val = 0;
st = _inkey_;
break;
case  _inkey_:
fprintf(stderr,"ERROR: spurious \"%s\" on the input. EXIT\n",yytext);
printf("ERROR: ^^^^^^^^ broken record follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
break;
}
}

"." {
switch(st) {
case _val_:
*p++ = *yytext;
break;
case _ex_:
case _er_:
fprintf(stderr,"ERROR: spurious '.' after ';' or '*E'. EXIT\n");
printf("ERROR: ^^^^^^^^ broken record follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
break;
case _inkey_:
mult_val = 1;
break;
}
}

"=" {
switch (st) {
case  _val_:
*p++ = *yytext;
break;
case  _ex_:
p = buf;
*p++ = *yytext;
st = _val_;
break;
case  _er_:
fprintf(stderr,"ERROR: spurious '=' after '*E'. EXIT\n");
printf("ERROR: ^^^^^^^^ broken record follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit (1);
break;
case  _inkey_:
if (no_par = find_par(key,no_cur)) {
no_cur = new_node(key,no_par);
}
else {
if (k2noval(key_par = k2par(key))) {
if (no_par = find_par(key_par,no_cur)) {
no_cur = new_node(key_par,no_par);
}
else {
fprintf(stderr,
 "ERROR: can't find parent to insert implied %s\n",
 k2s(key_par)
);
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
}
no_cur = new_node(key,no_cur);
}
```

```
else {
fprintf(stderr,"ERROR: can't find parent for %s. EXIT\n",
k2s(key)
);
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
}
}
p = buf;
st = _val_;
break;
}
}

";" {
switch (st) {
case  _val_:
if (p > buf && *(p-1) == '\\') {
*p++ = *yytext;
}
else {
*p = '\0';
_DBG_&&printf("key: %s%sval: %s\n",k2s(key), mult_val ? " +" : "  ",  buf);
val_node(no_cur,buf);
p = buf;
st = _ex_;
}
break;
case  _ex_:
break;
case  _er_:
fprintf(stderr,"ERROR: spurious ';' after '*E'. EXIT\n");
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
break;
case  _inkey_:
if (k2noval(key) == 0) {
fprintf(stderr,"ERROR: spurious ';' after '='. EXIT\n");
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
}
else {
if (no_par = find_par(key,no_cur)) {
no_cur = new_node(key,no_par);
st = _ex_;
}
else {
fprintf(stderr,"ERROR: can't find parent node for %s. EXIT\n",
k2s(key)
);
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
}
}
break;
}
}

"*E" {
switch (st) {
case  _val_:
for (py = yytext; *py; *p++ = *py++);
break;
case  _ex_:
key = _NOKEY_;
st = _er_;
/*
if (_DBG_) {
printf("EOR, record tree follows: ......................\n");
*/
print_tree(no_root,0);
```

```
/*
printf("Tree -> base .....................\n");
}*/
if (!tree2base(no_root)) {
printf("INFO: ^^^^^^ broken record tree follows >>>>>>>>\n");
print_tree(no_root,0);
fprintf(stderr,"ERROR: ^^^^^ Invalid record [EXIT]\n");
exit(1);
}
rec_loaded ++;
if (rec_loaded % 100 == 0)
printf("INFO: %d records loaded ..................... \n",rec_loaded);
_DBG_&&printf("====================================\nNew record ...........\n");
free_tree(no_root);
no_par = no_cur = no_root = new_node(_RECORD_, NULL);
break;
case  _er_:
break;
case  _inkey_:
if (k2noval(key) == 0) {
fprintf(stderr,"ERROR: spurious '*E' after key. EXIT\n");
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
}
else {
printf("WARNING: *E immediately after %s\n",k2s(key));
printf("WARNING: ^^^^^^^^^^ record tree follows >>>>>>>>>\n");
print_tree(no_root,0);
}
break;
}
}

[ \t\n] {
switch (st) {
case  _val_:
*p++ = *yytext;
break;
case  _ex_:
case  _er_:
case  _inkey_:
break;
}
}

[^ \t\n] {
switch (st) {
case  _val_:
*p++ = *yytext;
break;
case  _ex_:
st = _val_;
*p++ = *yytext;
break;
case  _er_:
fprintf(stderr,"ERROR: spurious '%c' after '*E'. EXIT\n",*yytext);
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
case  _inkey_:
fprintf(stderr,"ERROR: spurious '%c' after key. EXIT\n",*yytext);
printf("ERROR: ^^^^^^^^ broken record tree follows >>>>>>>>>>>>>>>>>\n");
print_tree(no_root,0);
exit(1);
}
}




%%
int yywrap(void) {
return 1;
}
```

```
int
main(void) {
int ik;
printf("INFO: init ....\n");
for (ik = 0; keys[ik].s; ik++)
if (keys[ik].vy_init)
keys[ik].vy_init();
rec_loaded = 0;
st = _er_;
no_par = no_cur = no_root = new_node(_RECORD_,NULL);
yylex();
return 0;
}
```

E.4 fdt.c

```
#include "tree.h"

#include "P/P.h"

struct record_key
keys[] = {

{"SC"              ,_SC_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"PDGSC"           ,_PDGSC_              ,_SC_        , 0, 0,0,0,0,0},
{"IRN"             ,_IRN_                ,_RECORD_    , 0, 0,0,0,0,0},
{"IRNW"            ,_IRNW_               ,_IRN_       , 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},
{"KTO"             ,_KTO_                ,_RECORD_    , 0, 0,0,0,0,0},
{"KRD"             ,_KRD_                ,_RECORD_    , 0, 0,0,0,0,0},
{"CONF"            ,_CONF_               ,_RECORD_    , 0, 0,0,0,0,0},
{"AUTHORS"         ,_AUTHORS_            ,_RECORD_    , 1, 0,0,0,0,0},
{"A"               ,_A_                  ,_AUTHORS_   , 0, 0,0,0,0,0},
{"SPOKESMAN"       ,_SPOKESMAN_          ,_A_         , 1, 0,0,0,0,0},
{"SPP"             ,_SPP_                ,_RECORD_    , 0, 0,0,0,0,0},
{"I"               ,_I_                  ,_AUTHORS_   , 0, 0,0,0,0,0},
{"ILBL"            ,_ILBL_               ,_I_         , 0, 0,0,0,0,0},
{"NA"              ,_NA_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"T"               ,_T_                  ,_RECORD_    , 0, 0,0,0,0,0},
{"TW"              ,_TW_                 ,_T_         , 0, 0,0,0,0,0},
{"TITLE-TEX"       ,_TITLE_TEX_          ,_T_         , 0, 0,0,0,0,0},
{"TA"              ,_TA_                 ,_T_         , 0, 0,0,0,0,0},
{"TA-TEX"          ,_TA_TEX_             ,_TA_        , 0, 0,0,0,0,0},
{"ABS"             ,_ABS_                ,_RECORD_    , 0, 0,0,0,0,0},
{"ABS-TEX"         ,_ABS_TEX_            ,_ABS_       , 0, 0,0,0,0,0},
{"DK"              ,_DK_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"L"               ,_L_                  ,_RECORD_    , 0, 0,0,0,0,0},
{"CIT"             ,_CIT_                ,_RECORD_    , 0, 0,0,0,0,0},
{"CCOD"            ,_CCOD_               ,_RECORD_    , 0, 0,0,0,0,0},
{"CB"              ,_CB_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"CB-TEX"          ,_CB_TEX_             ,_CB_        , 0, 0,0,0,0,0},
{"URL"             ,_URL_                ,_RECORD_    , 0, 0,0,0,0,0},
{"PPF"             ,_PPF_                ,_RECORD_    , 0, 0,0,0,0,0},
{"OR"              ,_OR_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"SB"              ,_SB_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"AV"              ,_AV_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"COMP"            ,_COMP_               ,_RECORD_    , 0, 0,0,0,0,0},
{"ES"              ,_ES_                 ,_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},
{"RCOD"            ,_RCOD_               ,_RR_        , 0, 0,0,0,0,0},
{"PUBLISHED-PAPERS",_PUBLISHED_PAPERS_   ,_RECORD_    , 0, 0,0,0,0,0},
{"PL"              ,_PL_                 ,_EXPERIMENT_, 0, 0,0,0,0,0},
```

```
{"CE"              ,_CE_                  ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"CE-TEX"          ,_CE_TEX_              ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"CD"              ,_CD_                  ,_EXPERIMENT_, 0, 0,0,0,0,0},
{"CD-TEX"          ,_CD_TEX_              ,_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},
{"FSP"             ,_FSP_                 ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"BP"              ,_BP_                  ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"TP"              ,_TP_                  ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"B"               ,_B_                   ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"PLAB"            ,_PLAB_                ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"ECM"             ,_ECM_                 ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"Q2"              ,_Q2_                  ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"W2"              ,_W2_                  ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"NU"              ,_NU_                  ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"DD"              ,_DD_                  ,_REAC_DATA_ , 0, 0,0,0,0,0},
{"INR"             ,_INR_                 ,_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},
{"INP"             ,_INP_                 ,_PART_PROP_ , 0, 0,0,0,0,0},
{"CR"              ,_CR_                  ,_RECORD_    , 0, 0,0,0,0,0},
{"CR-TEX"          ,_CR_TEX_              ,_RECORD_    , 0, 0,0,0,0,0},
{"CRW"             ,_CRW_                 ,_CR_        , 0, 0,0,0,0,0},
{"INDEX"           ,_INDEX_               ,_RECORD_    , 0, 0,0,0,0,0},
{"RD"              ,_RD_                  ,_RECORD_    , 0, 0,0,0,0,0},
{"CS"              ,_CS_                  ,_RECORD_    , 0, 0,0,0,0,0},
{"RPP"             ,_RPP_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"DEN"             ,_DEN_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"DCH"             ,_DCH_                 ,_RECORD_    , 0, 0,0,0,0,0},
{"ACTION"          ,_ACTION_              ,_RECORD_    , 0, 0,0,0,0,0},
{"QUESTION"        ,_QUESTION_            ,_RECORD_    , 0, 0,0,0,0,0},
{"ANSWER"          ,_ANSWER_              ,_QUESTION_  , 0, 0,0,0,0,0},
{"ERROR"           ,_ERROR_               ,_RECORD_    , 0, 0,0,0,0,0},

{0                 ,0                     ,0           , 0, 0,0,0,0,0}

};


char*
k2s(int k) { /*FIXME быстрый поиск?*/
int ik;
char *s = 0;
for (ik = 0; keys[ik].s; ik++) {
if (keys[ik].key == k) {
s = keys[ik].s;
break;
}
}
return s;
}

int
k2noval(int k) { /*FIXME: быстрый поиск? */
int ik, noval = 0;
for (ik = 0; keys[ik].s; ik++) {
if (keys[ik].key == k) {
noval = keys[ik].noval;
break;
}
}
return noval;
}


enum keytype
k2par(int k) { /*FIXME: быстрый поиск */
int ik;
for (ik = 0; keys[ik].s; ik++) {
if (keys[ik].key == k) {
return keys[ik].par;
}
}
return _UNKNOWN_;
```

```
}

int
k2ik (int k) {
int ik;
for (ik = 0; keys[ik].s; ik++) {
if(keys[ik].key == k) {
return ik;
}
}
return -1;
}

int
s2k(char *s) {
int ik = _UNKNOWN_, k;
for (k = 0; keys[k].s; k++) {
if (strcmp(keys[k].s, s) == 0) {
ik = keys[k].key;
break;
}
}
return ik;
}

int
s2ik(char *s) {
int ik;
for (ik = 0; keys[ik].s; ik++) {
if (strcmp(keys[ik].s, s) == 0) {
return ik;
}
}
return -1;
}
```

E.5 tree2base.c

```
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>
#include<dirent.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

#include "tree.h"
#include "util.h"

#include "settings.h"


extern
struct record_key
keys[];

static int
record_cur = 0;

static
char
*base = NULL,
*curdir = ".",
*base_records = NULL,
*rec_dir = NULL;

static FILE*
new_rec_file (char *basepath, int* rec_num);

static int
del_rec_file(int rec_num);
```

```
static char*
hash_from_val(enum keytype key, char *val);


static void
touch_file(char *file, char *dir);

static int
write_node(FILE *f_rec, struct node *no, int ident);


int
tree2base (struct node* no) {
int i;
DIR *d;
FILE *f_rec = 0;
int ret = 1, rec_num;
if (!no) {
printf("WARNING: tree2base( NULL ) called!");
return 0;
}
if (!base) {
if (!(base = getenv("PPDS_BASE"))) {
printf("WARNING: no PPDS_BASE env. variable, using ./\n");
base = curdir;
}
else {
printf("INFO: PPDS_BASE = %s\n",base);
while (!(d = opendir(base))) {
printf("INFO: can't open %s, trying to create ...\n",base);
if (mkdir(base,0770) < 0) {
fprintf(stderr,"ERROR: can't create %s [EXIT]\n",base);
exit(1);
}
}
closedir(d);
}
}
if (!base_records) {
base_records = (char*)malloc(sizeof(char) * (strlen(base) + strlen("/records/") + 1));
sprintf(base_records,"%s/records/",base);
printf("INFO: dir. with records: %s\n",base_records);
}
while (!(d = opendir(base_records))) {
printf("INFO: can't open %s, trying to create ... \n",base_records);
if (mkdir(base_records,0770) < 0) {
fprintf(stderr,"ERROR: can't create %s [EXIT]\n",base_records);
exit(1);
}
}
closedir(d);
if (!(f_rec = new_rec_file(base_records,&rec_num))) {
fprintf(stderr,"ERROR: can't create new record file in %s [EXIT]\n",base_records);
exit(1);
}
if (!write_node(f_rec,no,0)) {
fprintf(stderr,
"ERROR: tree2base(): dump to base failed, removing the incomplete record: %d\n"
"       Remove references to it from %s/*/r !!!!\n",
rec_num, base
);
if (!del_rec_file(rec_num)) {
exit(1);
}
return 0;
}
fclose(f_rec);
return 1;
}

static char*
node_dir(char* dir, char* keyname) {
DIR *d = 0;
struct dirent *entr = 0;
int lk, k_found;
```

```
char *new_name = 0;
_DBG_&&printf("DBG: node_dir(%s,%s) ... \n",dir,keyname);
if (!dir) {
fprintf(stderr,"ERROR: node_dir(NULL,...) called [EXIT]\n");
exit(1);
}
if (!(d = opendir(dir))) {
fprintf(stderr,"ERROR: node_dir(%s,%s): can't open %s [EXIT]\n",dir,keyname,dir);
exit(1);
}
if (!keyname) {/* root node! */
_DBG_&&printf("DBG: node_dir(...,NULL): call closedir(%s) ...\n",dir);
closedir(d);
_DBG_&&printf("DBG: node_dir() bp000: return dir;\n");
return dir;
}
lk = strlen(keyname);
for(k_found = 0;;) {
entr = readdir(d);
if (entr) {
if (entr->d_type == 4) {
if (strcmp(entr->d_name,".") == 0 || strcmp(entr->d_name,"..") == 0)
continue;
if (strncmp(entr->d_name,keyname,lk) == 0
    && ( entr->d_name[lk] == '\0' || entr->d_name[lk] == '.' ) )
k_found ++;
}
}
else
break;
}
closedir(d);
_DBG_&&printf("DBG: node_dir(): k_found = %d\n",k_found);
new_name = (char*)malloc(sizeof(char) * (strlen(dir) + 1 + strlen(keyname) + 132));
sprintf(new_name,"%s/%s.%d",dir,keyname,k_found);
_DBG_&&printf("DBG: node_dir(): new_name:%s:\n",new_name);
if (mkdir(new_name,0770) < 0) {
fprintf(stderr,"ERROR: can't create %s/ [EXIT]\n",new_name);
exit(1);
}
_DBG_&&printf("DBG: node_dir(): bp001: return new_name;\n");
return new_name;
}

static int
write_node(FILE *f_rec, struct node *no, int ident) {
char *hash_val = NULL;
int i, ret;
_DBG_&&printf("DBG: write_node( ...)\n");
if (!f_rec) {
fprintf(stderr,"ERROR: write_node(NULL, ...) called [EXIT]\n");
exit(1);
}
if (!no) {
printf("WARNING: write_node(..., NULL, ...L) called. Doing nothing.\n");
return;
}
if (no->type == _SC_)
printf("INFO: SC = %s;\n",no->val[0]);
for (i = 0; i < ident; i++)
fputc(' ',f_rec);
fprintf(f_rec,"%s {\n",k2s(no->type));
if (!k2noval(no->type)) {
for (i = 0; i < no->nval; i++) {
if (!(hash_val = hash_from_val(no->type,no->val[i]))) {
fprintf(stderr,"ERROR: can't find hash for %s = %s; Check the vocabulary?\n",
k2s(no->type), no->val[i]
);
return 0;
}
for (i = 0; i < ident+2; i++)
fputc(' ',f_rec);
fprintf(f_rec,"v:%s\n",hash_val);
}
}
```

```
for (i = 0; i < no->nch; i++)
if (!write_node(f_rec, no->ch[i], ident+4))
return 0;
for (i = 0; i < ident; i++)
fputc(' ',f_rec);
fputc('}',f_rec);
fputc('\n',f_rec);
return 1;
}



static FILE*
new_rec_file (char *basepath, int *rec_num) {
DIR *bd = 0;
struct dirent *entr = 0;
char path[132],
     *rec_path = NULL;
int rec_max = 0,  i;
FILE *f_rec = 0;
if (!(bd = opendir(basepath))) {
fprintf(stderr,"ERROR: can't open %s [EXIT]\n",basepath);
exit(1);
}
for(;;) {
entr = readdir(bd);
if (entr) {
if (entr->d_type == 8) {
if (entr->d_name[0] == '0') {
fprintf(stderr,
"ERROR: record name starts with `0': %s%s. EXIT\n",
base_records,entr->d_name
);
exit(1);
}
for (*rec_num = 0, i = 0; entr->d_name[i]; i++) {
if (entr->d_name[i] >= '0' && entr->d_name[i] <= '9') {
*rec_num = 10*(*rec_num) + entr->d_name[i] - '0';
}
else {
fprintf(stderr,
"ERROR: non-numerical record name: %s%s. EXIT",
base_records,entr->d_name
);
exit(1);
}
}
/* printf("%d: %s rec: %d\n",entr->d_type,entr->d_name, rec_num);*/
if (*rec_num > rec_max)
rec_max = *rec_num;
}
}
else
break;
}
closedir(bd);
record_cur = *rec_num = rec_max + 1;
_DBG_&&printf("DBG: max record number: %d, using %d for the new record\n",rec_max,*rec_num);
rec_path = (char*)malloc(sizeof(char) * (strlen(base_records) + 100)); /*FIXME*/
sprintf(rec_path,"%s%d",base_records,*rec_num);
if (!(f_rec = fopen(rec_path,"w"))) {
fprintf(stderr,"ERROR: can't create record file: %s  [EXIT]\n",rec_path);
exit(1);
}
free((void*)rec_path);
return f_rec;
}

int
del_rec_file(int rec_num) {
char *rec_path = (char*)malloc(sizeof(char) * (strlen(base_records) + 100)); /*FIXME*/
sprintf(rec_path,"%s%d",base_records,rec_num);
if (remove(rec_path) < 0) {
fprintf(stderr,"ERROR: can't remove %s [EXIT]\n",rec_path);
```

```
exit(1);
}
free((void*)rec_path);
return 1;
}

static char
hash_buf[256];

static char*
write_val_hash(char *fn_hsh, char *fn_val, char *val, enum keytype key) {
char hsh[64], *h,*p, c;
int rec_pos, n;
int found = 0;
FILE *f_hsh = fopen(fn_hsh,"r"),
     *f_val = 0;
int ik = k2ik(key);
h = (keys[ik].s2t && keys[ik].t2h) ? keys[ik].t2h(keys[ik].s2t(val)) : s2hash(val,hash_buf);
if (!h)
return NULL;
_DBG_&&printf("DBG: write_val_hash(): new hash: %s s:%s:\n",h,val);
if (!f_hsh) {
f_hsh = fopen(fn_hsh,"w");
if (f_val = fopen(fn_val,"r")) {
fprintf(stderr,
"ERROR: hash file %s is empty while the record file %s exists [EXIT]\n",
fn_hsh, fn_val
);
fclose(f_val);
fclose(f_hsh);
exit(1);
}
f_val = fopen(fn_val,"w");
fprintf(f_hsh,"%s %10d\n",h, 0);
fclose(f_hsh);
for (p = h; *p; p++)
fputc(*p,f_val);
fputc(' ',f_val);
for (p = val; *p; p++)
fputc(*p, f_val);
fputc('\0',f_val);
fputc('\n',f_val);
fclose(f_val);
}
else {
found = 0;
_DBG_&&printf("DBG: write_val_hash(): looking through %s ...\n",fn_hsh);
while ((n = fscanf(f_hsh,"%s%d",hsh,&rec_pos)) >= 0)  {
_DBG_&&printf("n: %d\n",n);
_DBG_&&printf("DBG: h,p: %s %d\n",hsh,rec_pos);
if (strcmp(hsh,h) == 0)  {
_DBG_&&printf("DBG: hash found\n");
found = 1;
break;
}
}
fclose(f_hsh);
if (!found) {
_DBG_&& printf("DBG: hash not found\n");
f_hsh = fopen(fn_hsh,"a");
f_val = fopen(fn_val,"a");
rec_pos = ftell(f_val);
_DBG_&&printf("DBG: f_val pos: %d\n",rec_pos);
fprintf(f_hsh,"%s %d\n",h,rec_pos);
fclose(f_hsh);
for (p = h; *p; p++)
fputc(*p,f_val);
fputc(' ',f_val);
for (p = val; *p; p++)
fputc(*p,f_val);
fputc('\0',f_val);
fputc('\n',f_val);
fclose(f_val);
}
else {
```

```
if (_DBG_) {
printf("DBG: checking rec at pos = %d\n",rec_pos);
f_val = fopen(fn_val,"r");
fseek(f_val,rec_pos,SEEK_SET);
printf("DBG: read: hsh:");
while ((c = fgetc(f_val)) != ' ')
putchar(c);
printf(": rec:");
while ((c = fgetc(f_val)) != '\0')
putchar(c);
printf(":\n");
fclose(f_val);
}
}
}
return h;
}


static char*
hash_from_val(enum keytype key, char *val) {
char *key_name = k2s(key),
     *key_dir  = (char*)malloc(sizeof(char) * (strlen(base) + 1 + strlen(key_name) + 1)),
     *hash_file_path = 0,
     *val_file_path = 0,
     *hash = 0;
DIR *d;
FILE *f_val = 0, *f_hash = 0, *f_record = 0;
sprintf(key_dir,"%s/%s",base,key_name);
while (!(d = opendir(key_dir))) {
printf("INFO: can't open %s, trying to create ... \n",key_dir);
if (mkdir(key_dir,0770) < 0) {
fprintf(stderr,"ERROR: can't create %s [EXIT]\n",key_dir);
exit(1);
}
}
closedir(d);
hash_file_path = (char*)malloc(sizeof(char) * (strlen(key_dir) + 1 + 1 + 1));
val_file_path  = (char*)malloc(sizeof(char) * (strlen(key_dir) + 1 + 1 + 1));
sprintf(hash_file_path,"%s/h",key_dir);
sprintf(val_file_path, "%s/v",key_dir);
if (!(hash = write_val_hash(hash_file_path,val_file_path, val, key)))
return NULL;
free((void*)hash_file_path);
sprintf(val_file_path, "%s/r",key_dir);
f_record = fopen(val_file_path,"a");
fprintf(f_record,"%s %d\n",hash,record_cur);
fclose(f_record);
free((void*)val_file_path);
return hash;
}


static void
touch_file (char *fname, char* dir) {
char *fname_cut,  *new_path;
DIR *d;
FILE *f;
int i;
if (!(d = opendir(dir))) {
fprintf(stderr,"ERROR: %s doesn't exist [EXIT]\n",dir);
exit(1);
}
closedir(d);
new_path = (char*)malloc(sizeof(char) * (strlen(dir) + 1 + strlen(fname) + 1));
sprintf(new_path,"%s/%s",dir,fname);
if (!(f = fopen(new_path,"a"))) {
fprintf(stderr,"ERROR: can't create %s  [EXIT]\n",new_path);
exit(1);
}
fclose(f);
free((void*)new_path);
return;
}
```

E.6 P/Makefile

```
%.o: %.c P.h ../../../src/util/util.h ../settings.h
cc $(CFLAGS) -I. -I.. -I../../../src/util  -c -o $@ $<

clean:
rm -f *.o *~
```

E.7 P/P.h

```
struct type_P {
int id;
int  bar,lep, S, C, B, T, anti, heavy_nuc, class;
double m, I3, Q;
char orcode[32];
int nval;
char **val;
long val0sum;
};

void
vy_init_P (void);

void*
s2t_P (char*);

char*
t2s_P(void*);

char*
t2h_P(void*);

int
eq_P (void*,void*);

char*
id2s_P(int id);
```

E.8 P/f.c

```
#include "P.h"
#include "settings.h"
#include "util.h"

#include<stdio.h>
#include<stdlib.h>
#include<string.h>


static
int n_P = 0;

static
struct type_P*
_P = NULL;


void
vy_init_P (void) {
char *vy_base = 0, *vy_fn = 0, c, *p, *n1, *n2, *n2_min;
FILE *vy_f = 0;
int i,j, j_min, n;
long d, d_min;
struct type_P p_tmp;
if (_P) {
fprintf(stderr,"ERROR: vy_init_P(): _P[] is already initialized [EXIT]\n");
exit(1);
}
```

```
if (!(vy_base = getenv("PPDS_VY"))) {
fprintf(stderr,"ERROR: vy_init_P(): PPDS_VY env. pointing to the "
       "vocabulary directory is not defined [EXIT]\n"
);
exit(1);
}
vy_fn = (char*)malloc(sizeof(char)*(strlen(vy_base) + 1 + 1 + 1 + 2 + 1));
sprintf(vy_fn,"%s/P/vy",vy_base);
printf("INFO: reading particle definitions from PPDS_VY = %s\n",vy_fn);
if (!(vy_f = fopen(vy_fn,"r"))) {
fprintf(stderr,"ERROR: vy_init_P(): can't open %s [EXIT]\n",vy_fn);
exit(1);
}
_P = (struct type_P*) malloc(sizeof(struct type_P) * (n = 256) );
  /* particle format: id bar lep S C B T anti heavy_nuc class m I3 Q orcode name: */
for (n_P = 0;
     fscanf(vy_f,"%d %d %d  %d %d %d %d  %d %d %d  %lf %lf %lf %s",
     &_P[n_P].id,
     &_P[n_P].bar,
     &_P[n_P].lep,
     &_P[n_P].S,
     &_P[n_P].C,
     &_P[n_P].B,
     &_P[n_P].T,
     &_P[n_P].anti,
     &_P[n_P].heavy_nuc,
     &_P[n_P].class,
     &_P[n_P].m,
     &_P[n_P].I3,
     &_P[n_P].Q,
      _P[n_P].orcode
    ) == 14;
     n_P++) {
_P[n_P].val = (char**) malloc(sizeof(char*) * 256);
_P[n_P].nval = 0;
_P[n_P].val0sum = 0;
_P[n_P].val[_P[n_P].nval] = 0;
while ((c = fgetc(vy_f)) != EOF) {
switch (c) {
  case ' ' :
  case '\t':
  case '\n':
if (_P[n_P].val[_P[n_P].nval]) {
*p = '\0';
_P[n_P].val[_P[n_P].nval] = (char*) realloc (
_P[n_P].val[_P[n_P].nval],
  sizeof(char) * (strlen(_P[n_P].val[_P[n_P].nval]) + 1)
 );
_P[n_P].nval ++;
_P[n_P].val[_P[n_P].nval] = 0;
}
if (c == '\n')
goto out;
break;
  default:
if (!_P[n_P].val[_P[n_P].nval])
p = _P[n_P].val[_P[n_P].nval] = (char*)malloc(sizeof(char)*64);
*p++ = c;
break;
}

}
out:
_P[n_P].val = (char**) realloc(_P[n_P].val, sizeof(char*) * _P[n_P].nval);
if (_P[n_P].nval)
_P[n_P].val0sum = s2sum(_P[n_P].val[0]);
if(_DBG_)
printf("DBG: particle read: ");
if (_P[n_P].nval == 0)
printf("WARNING: no name for particle: ");
if (_DBG_ || _P[n_P].nval == 0) {
for (i = 0; i < _P[n_P].nval; i++)
printf("%s ",_P[n_P].val[i]);
printf(" or: %s bar: %d lep: %d S,C,B,T: %d %d %d %d "
      "p/anti-p: %d hvy/N: %d class: %d"
```

```
      "m: %lf I3: %lf Q: %lf\n",
      _P[n_P].orcode,
      _P[n_P].bar,
      _P[n_P].lep,
      _P[n_P].S,
      _P[n_P].C,
      _P[n_P].B,
      _P[n_P].T,
      _P[n_P].anti,
      _P[n_P].heavy_nuc,
      _P[n_P].class,
      _P[n_P].m,
      _P[n_P].I3,
      _P[n_P].Q
     );
}
if (n_P + 2 > n)
_P = (struct type_P*)realloc(_P, sizeof(struct type_P) * (n *= 2));
}
_P = (struct type_P*)realloc(_P, sizeof(struct type_P) * n_P);
printf("INFO: %d particles in the vocabulary\n",n_P);
_DBG_&&printf("DBG: sorting by name[0] ...\n");
for (i = 0; i < n_P; i++) {
d_min = _P[i].val0sum;
n1 = _P[i].nval ? _P[i].val[0] : 0;
j_min = i;
for (j = i+1; j < n_P; j++) {
d =  _P[j].val0sum;
n2 = _P[j].nval ? _P[j].val[0] : 0;
if (d < d_min) {
d_min = d;
j_min = j;
n2_min = n2;
}
}
if (j_min != i) {
p_tmp = _P[j_min];
_P[j_min] = _P[i];
_P[i] = p_tmp;
}
}
if(_DBG_) {
printf("DBG: particles sorted by name[0] ....... \n");
for (i = 0; i < n_P; i++) {
for (j = 0; j < _P[i].nval; j++)
printf("%s ",_P[i].val[j]);
printf(" or: %s bar: %d lep: %d S,C,B,T: %d %d %d %d "
      "p/anti-p: %d hvy/N: %d class: %d"
      "m: %lf I3: %lf Q: %lf\n",
      _P[i].orcode,
      _P[i].bar,
      _P[i].lep,
      _P[i].S,
      _P[i].C,
      _P[i].B,
      _P[i].T,
      _P[i].anti,
      _P[i].heavy_nuc,
      _P[i].class,
      _P[i].m,
      _P[i].I3,
      _P[i].Q
      );
}
printf("................................................\n");
}

return;
}


void*
s2t_P (char* s) {
int i, j;
for (i = 0; i < n_P; i++)
```

```
for (j = 0; j < _P[i].nval; j++)
if(strcmp(_P[i].val[j],s) == 0)
return (void*)&_P[i];
return NULL;
}

char*
t2s_P(void* t) {
return NULL;
}

static char hbuf[132];

char*
t2h_P(void *t) {
struct type_P *p = (struct type_P*) t;
if (p) {
sprintf(hbuf,"%044x", p->id);
return hbuf;
}
return NULL;
/* p ? p->orcode : "$$ERR$$"; */
}

int
eq_P (void* t1,void* t2) {
return 0;
}

static
char *_s_P_unknown = "$ERROR";

char*
id2s_P(int id) {
int i;
if (id == -1)
return _s_P_unknown;
for (i = 0; i < n_P; i++)
if (_P[i].id == id)
return _P[i].val[0];
return _s_P_unknown;
}
```

Ж Процессор реакций

Ж.1 Makefile

```
CFLAGS = -g -O0


main: main.o re/y.a ../dg/src/P/f.o util/s2hash.o
cc -o $@ $^ -lm -lcrypt

util/s2hash.o: util/s2hash.c
cd util && make s2hash.o

main.o: main.c re/re.h ../dg/src/P/P.h
cc $(CFLAGS) -I ../dg/src -c -o $@ $<

%/y.a: re/settings.h
cd $$( echo $@ | sed 's,/[^/]*$$,,') && make y.a CFLAGS="$(CFLAGS)"

clean:
rm -f *.o main *~
for f in */ ; do cd $$f && make clean ; cd .. ; done
```

Ж.2 main.c

```
#include <stdio.h>

#include "re/re.h"

#include "util/util.h"

#include "P/P.h"


int
main (void) {
char s[2048];
int r;
struct listlist *is, *fs;
char *buf;
vy_init_P();
while (gets(s)) {
r = string2listlist(s, &is,&fs);
printf("ret st: %d\n",r);
if (is) {
/*
print_listlist(is);
*/
buf = listlist2string(is);
printf("\n\nmain: >> IS -> string >>>>>>>>>>> \n\n%s\n\n",buf);
printf("main: <<<<<<<<<<<<<<< \n\n");
}
if (fs) {
printf("\n\n main: >>>>> FS >>>>>>> \n\n");
/*
print_listlist(fs);
*/
buf = listlist2string(fs);
printf("\n\nmain: >>> FS -> string >>>>\n\n%s\n\n",buf);
printf("main: <<<< \n\n");
}
printf("\n########################################\n");
}
return 0;
}
```

Ж.3 re/Makefile

```
#CFLAGS = -g -O0

y.tab.h y.tab.c: y.y tree.h
bison -d $<

lex.yy.c: l.l y.tab.h v.h
lex $<

y.a: y.tab.o  lex.yy.o v.o tree.o
rm -f $@
ar r $@ $^

%.o: %.c
cc $(CFLAGS) -I../../dg/src/ -c -o $@ $<


clean:
rm -f *.a *.o y lex.yy.c y.tab.[ch] *~
```

Ж.4 re/l.l

```
%{
```

```
#include <stdio.h>
#include <stdlib.h>

#include "y.tab.h"

#include "v.h"

%}

%%
[ \t]+\-\->[ \t]+ {
printf("lex: arr:\n");
return ARROW;
}

[A-Za-z][A-Za-z0-9_\*\+\-\(\)]*|\([A-Za-z][A-Za-z0-9_\*\+\-\(\)]*\) {
printf("lex: prt: %s -> %d\n",yytext,yylval = prt_id(yytext) );
return PRT;
}

[0-9]+ {
printf("lex: num: %d\n",yylval  = atoi(yytext));
return NUM;
}

^[ \t]+        { printf("l:^[ ]\n"); }

[ \t]*\;       { printf("l:;\n"); return ';'; }

[ \t]+\+[ \t]+ { printf("l:+\n"); return '+'; }

[ \t]+\-[ \t]+ { printf("l:-\n"); return '-'; }

\([ \t]+       { return '('; }

[ \t]+\)       { return ')'; }

[ \t]+\<[ \t]+ { printf("l:<\n"); return '<'; }

[ \t]+\>       { printf("l:>\n"); return '>'; }

[ \t]+\.[ \t]+ { return '.'; }

[ \t]+         { printf("l:*\n"); return '*'; }

.              { fprintf(stderr,"Invalid char:%c:\n",*yytext); }



%%
int yywrap(void) {
return 1;
}


/*
int  main(void) {
yylex();
return 0;
}
*/
```

Ж.5 re/y.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* lex stuff: */

typedef struct yy_buffer_state *YY_BUFFER_STATE;
```

```
extern
YY_BUFFER_STATE
yy_scan_string (char*);

extern
void
yy_delete_buffer (YY_BUFFER_STATE);

/***/


#include "tree.h"

#include "settings.h"


#define YYSTYPE union r_union_type

union r_union_type {
        int id;
        struct node *no;
};

extern
struct node*
new_node(enum op_type op, int id, int n, struct node* dec, struct node *left, struct node *right);

extern
void
print_prt(int id);

extern
void
print_tree(struct node* no);

extern
struct node*
norm_tree(struct node*);

extern
struct node_list*
tree2treelist(struct node*);

extern
struct listlist*
treelist2listlist (struct listlist* prev, struct node_list* no, int n);

extern
struct listlist*
contract_listlist (struct listlist*);

extern
void
print_listlist (struct listlist *l);



static
struct node
*fs, *is, *fs_norm, *is_norm;

static
struct node_list
*fs_norm_list, *is_norm_list;

static
struct listlist
*fs_listlist, *is_listlist, *fs_listlist_contr, *is_listlist_contr;


void
yyerror(char*);

%}
```

```
%token ARROW PRT NUM
%left '+' '-'
%left '*'
%left '.'


%%


input:
     input re ';' {
if (_DBG_) {
         printf("\n>>>>>>>>>>>> RES: >>>>>>>>>>>>>>>>>> \n\n");
   if (is) {
printf("   --- IS: >>>>>>>>>\n");
print_tree( is );
printf("\n\n .... norm: .....\n\n");
print_tree( is_norm );
printf("\n\n ...... T2TL:  .....\n\n");
print_treelist( is_norm_list );
printf("\n\n ...... -> LL ....... \n\n");
print_listlist( is_listlist );
printf("\n\n ....., -> CLL .......\n\n");
print_listlist( is_listlist_contr );
printf("\n\n   <<<<<<<<<<<<<<<<<<<<<<< IS ---\n\n");
   }
   if (fs) {
printf("   --> FS: >>>>>>>>>\n");
print_tree( fs );
printf("\n\n .... norm: .....\n\n");
print_tree( fs_norm );
printf("\n\n ...... T2TL:  .....\n\n");
print_treelist( fs_norm_list );
printf("\n\n ...... -> LL ....... \n\n");
print_listlist( fs_listlist );
printf("\n\n ....., -> CLL .......\n\n");
print_listlist( fs_listlist_contr );
printf("\n\n   <<<<<<<<<<<<<<<<<<<<<<< FS ---\n\n");
   }
}
     }
     |
     ;

re:
    expr ARROW expr {
     is = $<no>1;
fs = $<no>3;
     is_norm = norm_tree(is);
fs_norm = norm_tree(fs);
is_norm_list = tree2treelist( is_norm );
fs_norm_list = tree2treelist( fs_norm );
is_listlist = treelist2listlist(NULL, is_norm_list, 1);
fs_listlist = treelist2listlist(NULL, fs_norm_list, 1);
is_listlist_contr = contract_listlist( is_listlist );
fs_listlist_contr = contract_listlist( fs_listlist );
    }
    |
    expr            {
/*_DBG_&&printf("   --- IS: >>>>>>>>>\n");*/
     is = $<no>1;
/* _DBG_&&print_tree( is );*/
fs = NULL;
/* _DBG_&&printf("\n\n .... norm: .....\n\n");*/
     is_norm = norm_tree(is);
/* _DBG_&&print_tree( is_norm );*/
fs_norm = NULL;
/* _DBG_&&printf("\n\n ...... T2TL:  .....\n\n");*/
is_norm_list = tree2treelist( is_norm );
/* _DBG_&&print_treelist( is_norm_list );*/
fs_norm_list = NULL;
/* _DBG_&&printf("\n\n ...... -> LL ....... \n\n");*/
is_listlist = treelist2listlist(NULL, is_norm_list, 1);
/* _DBG_&&print_listlist( is_listlist );*/
fs_listlist =  NULL;
```

```
/* _DBG_&&printf("\n\n ....., -> CLL .......\n\n");*/
is_listlist_contr = contract_listlist( is_listlist );
/* _DBG_&&print_listlist( is_listlist_contr );*/
fs_listlist_contr = 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(_PRJ_,0,1, $<no>1,$<no>3); }
    |
    */
    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; }
    ;
%%

void
yyerror(char* s) {
fprintf(stderr,"%s\n",s);
}


int
string2listlist (char *s, struct listlist** is, struct listlist** fs) {
int ret;
YY_BUFFER_STATE buffer;
_DBG_&&printf("DBG:string2listlist( %s ): bp000 >>>>>>\n",s);
buffer = yy_scan_string(s);
ret = yyparse();
yy_delete_buffer(buffer);
*is = is_listlist_contr;
*fs = fs_listlist_contr;
return ret;
}

/*****
int
main (void) {
char s[2048];
int r;
struct listlist *is, *fs;
while (gets(s)) {
r = string2listlist(s, &is,&fs);
printf("ret st: %d\n",r);
if (is) {
printf("\n\n >>>>> IS >>>>>>> \n\n");
print_listlist(is);
printf("\n\n <<<<<<<<<<<<<<< \n\n");
}
if (fs) {
printf("\n\n >>>>> FS >>>>>>> \n\n");
print_listlist(fs);
printf("\n\n <<<<<<<<<<<<<<< \n\n");
}
printf("\n#########################################\n");
}
return 0;
}
*****/
```

Ж.6 re/re.h

```
#ifndef _re_h_

struct list {
int id, n;
struct listlist *dec;
struct list *prev, *next;
};

struct listlist {
struct list *l;
int n;
struct listlist *prev, *next;
};


struct listlist*
contract_listlist (struct listlist*);

void
print_listlist (struct listlist *l);

int
string2listlist (char *s, struct listlist** is, struct listlist** fs);

char*
listlist2string (struct listlist*l);

#define _re_h_

#endif
```

Ж.7 re/tree.h

```
#ifndef _tree_h_

enum op_type {
_TERM_,
_PLUS_,
_MINUS_,
_MUL_,
_PRJ_
};

struct node {
enum op_type op;
int id, n;
struct node *dec;
struct node *left, *right;
};

struct list {
int id, n;
struct listlist *dec;
struct list *prev, *next;
};

struct node_list {
enum op_type op;
struct list* l;
struct node_list *left, *right;
};

struct listlist {
struct list *l;
int n;
struct listlist *prev, *next;
};

#define _tree_h_
```

```
#endif
```

ℵ.8 re/tree.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "tree.h"

#include "settings.h"

#include "P/P.h"

struct node*
new_node(enum op_type op, int id, int n, struct node *dec, struct node *left, struct node *right) {
struct node* new = (struct node*)malloc(sizeof(struct node));
new->op = op;
new->id = id;
new->n  = n;
new->dec = dec;
new->left = left ;
new->right = right;
return new;
}

struct node*
copy_tree (struct node *no) {
struct node *new = (struct node*) malloc(sizeof(struct node));
new->id = no->id;
new->n  = no->n;
if (no->dec)
new->dec = copy_tree(no->dec);
else
new->dec = NULL;
if ((new->op = no->op) != _TERM_) {
new->left = copy_tree(no->left);
new->right = copy_tree(no->right);
}
else {
new->left = new->right = NULL;
}
return new;
}

struct node*
norm_tree (struct node *no) {
enum op_type op_left, op_right, op_old, op_new;
struct node
*new_r = 0,
*new_l1 = 0, *new_l2 = 0,
*new_node = 0,
*new_r1 = 0, *new_r2 = 0,
*new_l = 0;
if (!no)
return NULL;
op_old = no->op;
if (!no)
return no;
if (op_old == _TERM_) {
no->dec = norm_tree(no->dec);
return no;
}
no->left = norm_tree(no->left);
no->right = norm_tree(no->right);
if (op_old == _MUL_ /*|| op_old == _PRJ_*/)  {
if ((op_left = no->left->op) == _PLUS_ || op_left == _MINUS_) {
new_l1 = norm_tree(no->left->left);
new_l2 = norm_tree(no->left->right);
new_r = norm_tree(no->right);
new_node = (struct node*) malloc(sizeof(struct node));
new_node->op = op_old;
```

```
new_node->id = 0;
new_node->n  = 1;
new_node->dec = NULL;
new_node->left = new_l2;
new_node->right = new_r;
no->right = new_node;
no->left->left = new_l1;
no->left->right= copy_tree(new_r);
no->left->op = op_old;
no->op = op_left;
}
else if ((op_right = no->right->op) == _PLUS_ || op_right == _MINUS_) {
new_r1 = norm_tree(no->right->left);
new_r2 = norm_tree(no->right->right);
new_l = norm_tree(no->left);
new_node = (struct node*) malloc(sizeof(struct node));
new_node->op = op_old;
new_node->id = 0;
new_node->n  = 1;
new_node->dec = NULL;
new_node->left = new_r1;
new_node->right = new_l;
no->left = new_node;
no->right->left = new_r2;
no->right->right = copy_tree(new_l);
no->right->op = op_old;
no->op = op_right;
}
}
no->left = norm_tree(no->left);
no->right = norm_tree(no->right);
return no;
}

void
print_prt(int id) {
printf("prt(%s)",id2s_P(id));
return;
}

void
print_tree(struct node* no) {
/* _DBG_&&printf("DBG: print_tree(): op = %d\n",(int)no->op);*/
switch(no->op) {
case _TERM_:
if(no->n != 1)
printf("%d*",no->n);
print_prt(no->id);
if (no->dec) {
printf(" < ");
print_tree(no->dec);
printf(" > ");
}
break;
case _PLUS_:
case _MINUS_:
printf(no->op == _PLUS_ ? " + ( " : " - ( " );
print_tree(no->left);
printf(" , ");
print_tree(no->right) ;
printf(" ) ");
break;
case _MUL_:
printf(" * ( ");
print_tree(no->left);
printf(" , ");
print_tree(no->right);
printf(" ) ");
break;
/*
case _PRJ_:
printf(" . ( ");
print_tree(no->left);
printf(" , ");
print_tree(no->right);
```

```
printf(" ) ");
break;
*/
default:
printf("ERROR: print_tree(): no->op = %d\n",(int)no->op);
break;
}
/* _DBG_&&printf("DBG: print_tree(): bp1\n");*/
return;
}


/*****************************************************************************/

void
free_list(struct list* l);

struct listlist*
treelist2listlist (struct listlist* prev, struct node_list* no, int n);

struct node_list*
tree2treelist(struct node*);

int
eq_listlist(struct listlist* l1, struct listlist* l2);

struct listlist*
contract_listlist(struct listlist *l);

void
free_listlist(struct listlist* l);

void
print_listlist (struct listlist *l);

char*
listlist2string (struct listlist *l);

void
print_list(struct list* l);

static
char*
_list2s (struct list* l, char* buf);

static
char*
_listlist2s (struct listlist *l, char *buf);

/*-------------------------------------------------------*/


struct list*
last (struct list* l) {
struct list* no = l;
while (no->next)
no = no->next;
return no;
}
void
mul (struct list* l, int n) {
struct list *no = l;
while (no->next)
no->n *= n;
return;
}
struct list*
cat (struct list* l1, int n1, struct list* l2, int n2) {
struct list* l1_last = last (l1);
if (n1 != 1)
mul(l1,n1);
if (n2 != 1)
mul(l2,n2);
l1_last->next = l2;
l2->prev = l1_last;
```

```
return l1;
}

void
free_list(struct list* l) {
struct list* no, *no1;
_DBG_&&printf("FL(): >>>>>\n");
if (!l) {
_DBG_&&printf("FL:bp000: l == 0, ret <<<<<\n");
return;
}
for (no = l; no;) {
if (no->dec) {
_DBG_&&printf("FL:bp001: call FLL( no->dec ) >>>>\n");
free_listlist(no->dec);
_DBG_&&printf("FL:bp001-1: <<<<<<<\n");
}
no1 = no->next;
_DBG_&&printf("FL:bp0002 free(no)\n");
free((void*)no);
no = no1;
}
_DBG_&&printf("FL: return <<<<<<\n");
return;
}

struct list*
tree2list(struct list* prev, struct node* no) {
struct list* new = NULL;
_DBG_&&printf("BP1111111: t2l() .... >>>>>\n");
switch(no->op) {
case _TERM_:
new = (struct list*)malloc(sizeof(struct list));
_DBG_&&printf("t2l:bp000: op = _TERM_ id: %d n: %d >>>\n",no->id,no->n);
new->id = no->id;
new->n  = no->n;
if (no->dec) {
_DBG_&&printf("t2l:bp0001: dec != 0: call tl2ll( ... t2tl(no->dec)) >>>>\n");
new->dec = treelist2listlist(NULL, tree2treelist(no->dec), 1);
_DBG_&&printf("t2l:bp0001-1:  <<<<<<\n");
}
else {
_DBG_&&printf("t2l:bp0002: dec == 0\n");
new->dec = NULL;
}
if (prev)
prev->next = new;
new->prev = prev;
new->next = NULL;
_DBG_&&printf("t2l:bp000-1: <<<<<\n");
break;
case _MUL_:
_DBG_&&printf("t2l:bp001: op = _MUL_ new = cat (t2l(left),t2l(right) >>>>\n");
new = cat( tree2list(NULL,no->left),1, tree2list(NULL,no->right),1 );
_DBG_&&printf("t2l:bp001-1 <<<<<\n");
if (prev)
prev->next = new;
new->prev = prev;
break;
default:
fprintf(stderr,"ERROR: tree2list(): wrong no->op: %d\n",(int)no->op);
exit(1);
break;
}
_DBG_&&printf("BP1111111-1: t2l() <<<<<<<<\n");
return new;
}

struct list*
sort_list (struct list* l) {
struct list *no = l, *no1, *no2;
struct listlist* no_dec;
int min_id, no_id, no_n;
_DBG_&&printf("sort_list(): bp0000 >>>> \n");
if (!l)
```

```
return l;
do {
if (no->dec) {
_DBG_&&printf("sort_list(): bp0001: no->dec = contract_listlist() >>>\n");
no->dec = contract_listlist(no->dec);
_DBG_&&printf("sort_list(): bp0001-1:  contract_listlist() returned <<<<\n");
}
min_id = no->id;
no2 = NULL;
for (no1 = no->next; no1; no1 = no1->next) {
if (no1->id < min_id) {
no2 = no1;
min_id = no1->id;
}
}
if (min_id < no->id) {
no_id = no->id;
no_n = no->n;
no_dec = no->dec;
no->id = no2->id;
no->n = no2->n;
no->dec = no2->dec;
no2->id = no_id;
no2->n = no_n;
no2->dec = no_dec;
}
} while (no = no->next);
_DBG_&&printf("sort_list(): bp0000-1: return l <<<\n");
return l;
}

struct list*
contract_list(struct list* l) {
struct list *no = l, *no1, *no2;
if(_DBG_) {
printf("contract_list: bp0000 >>>>> \n");
print_list(l);
}
do {
if (no->dec)
no->dec = contract_listlist(no->dec);
} while (no = no->next);
no = l;
do {
for (no1 = no->next; no1;) {
_DBG_&&printf("contract_list: bp00001 ... eq_listlist(c_ll,cll) ...\n");
if (   no1->id == no->id  && eq_listlist( no->dec, no1->dec ) ) {
no->n += no1->n;
no1->prev->next = no1->next;
if (no2 = no1->next)
no2->prev = no1->prev;
free_listlist(no1->dec);
free((void*)no1);
no1 = no2;
}
else {
no1 = no1->next;
}
}
} while (no = no->next);
_DBG_&&printf("contract_list: bp0001 <<<<<< return sort_list(l)\n");
return sort_list(l);
}


int
eq_list(struct list *l1, struct list *l2) {
struct list *no1 = l1, *no2 = l2;
for(;;) {
if (no1 == NULL && no2 == NULL)
return 1;
if (no1 == NULL || no2 == NULL)
return 0;
if (!(no1->id == no2->id && no1->n == no2->n && eq_listlist(no1->dec,no2->dec)))
return 0;
```

```
no1 = no1->next;
no2 = no2->next;
}
}

/*-----------------------------------------------------*/

int
eq_listlist(struct listlist* l1, struct listlist* l2) {
struct listlist *no1 = l1, *no2 = l2;
for (;;) {
if (no1 == NULL && no2 == NULL)
return 1;
if (no1 == NULL || no2 == NULL)
return 0;
if (! (no1->n == no2->n && eq_list(no1->l,no2->l)) )
return 0;
no1 = no1->next;
no2 = no2->next;
}
}

void
free_listlist(struct listlist* l) { //FFFFIIIIIIIXXXXXXMMMMMEEEEE
struct listlist *no, *no1;
if(!l)
return;
for (no = l; no;) {
if (no->l) {
_DBG_&&printf("FLL: no: "); print_list(no->l);
_DBG_&&printf("FLL: bp000: free_list(no->l) >>>>\n");
free_list(no->l);
_DBG_&&printf("FLL: bp000-1: <<<<\n");
}
no1 = no->next;
free((void*)no);
no = no1;
}
return;
}

struct node_list*
tree2treelist (struct node* no) {
struct node_list* new = (struct node_list*)malloc(sizeof(struct node_list));
_DBG_&&printf("BP000000: t2tl() >>>>>>\n");
switch(no->op) {
case _TERM_:
_DBG_&&printf("op = _TERM_: no->id = %d no->n = %d\n",no->id,no->n);
new->op = _TERM_;
new->left =
new->right = NULL;
new->l  = (struct list*)malloc(sizeof(struct list));
new->l->id = no->id;
new->l->n  = no->n;
if (no->dec) {
_DBG_&&printf(" bp000: no->dec != 0 tl2ll( ... t2tl( no->dec )) >>>> \n");
new->l->dec = treelist2listlist(NULL, tree2treelist(no->dec), 1);
_DBG_&&printf(" bp000-1: <<< \n");
}
else {
_DBG_&&printf(" bp001: no->dec = 0\n");
new->l->dec = NULL;
}
new->l->prev =
new->l->next = NULL;
break;
case _MUL_:
new->op = _TERM_;
new->left =
new->right = NULL;
_DBG_&&printf(" b002: op = * ; new->l = cat t2l(left) t2l(right) >>>> \n");
new->l = cat( tree2list(NULL,no->left),1, tree2list(NULL,no->right),1 );
_DBG_&&printf(" bp002-1: <<<<\n");
break;
/*
```

```
case _PRJ_:
new->op = _TERM_;
new->left =
new->right = NULL;
new->l = cat( tree2list(NULL,no->left),-1, tree2list(NULL,no->right),1 );
break;
*/
case _PLUS_:
new->op = _PLUS_;
_DBG_&&printf(" bp003: new->l,r = t2tl(left), t2tl(right) >>>>> \n");
new->left  = tree2treelist(no->left);
new->right = tree2treelist(no->right);
_DBG_&&printf(" bp003-1: <<<<< \n");
break;
case _MINUS_:
new->op = _MINUS_;
new->left  = tree2treelist(no->left);
new->right = tree2treelist(no->right);
break;
default:
fprintf(stderr,"ERROR: unknown no->op\n");
exit(1);
break;
}
_DBG_&&printf("BP000000-1: <<<<<<<<<<,\n");
return new;
}

void
print_list(struct list* l) {
struct list *no = l;
/*
if (!l) {
printf(" null ");
return;
}
*/
do {
printf("%d*%s ",no->n, id2s_P(no->id));
if (no->dec) {
printf("< ");
print_listlist(no->dec);
printf(" > ");
}
} while (no = no->next);
return;
}

static
char*
_list2s(struct list* l, char *buf) {
struct list *no = l;
char *p = buf ? buf + strlen(buf) : 0;
do {
printf("_list2s():bp00 buf:%s\n",buf);
buf = (char*)realloc(buf,sizeof(char) * ((buf ? strlen(buf) : 0) + 132));
if (!p)
p = buf;
printf("_list2s():bp01 n:%d s(no->id):%s buf:%s\n",no->n,id2s_P(no->id),buf);
sprintf(p,"%d*%s ",no->n, id2s_P(no->id));
p = buf + strlen(buf);
if (no->dec) {
sprintf(p,"< ");
_listlist2s(no->dec,buf);
buf = (char*)realloc(buf,sizeof(char) * (strlen(buf) + 4));
p = buf + strlen(buf);
sprintf(p," > ");
p = buf + strlen(buf);
}
} while (no = no->next);
printf("_list2s( return buf: %s\n",buf);
return buf;
}

void
```

```
print_treelist(struct node_list* no) {
switch(no->op) {
case _TERM_:
print_list(no->l);
break;
case _PLUS_:
case _MINUS_:
printf(no->op == _PLUS_ ? " + [ " : " - [ ");
print_treelist(no->left);
printf (" , ");
print_treelist(no->right);
printf (" ] ");
break;
default:
fprintf(stderr,"ERROR:print_treelist(): wrong no->op = %d\n",(int)no->op);
exit(1);
break;
}
}

/******/

struct listlist*
last_listlist(struct listlist *l) {
struct listlist* no = l;
while (no->next)
no = no->next;
return no;
}

struct listlist*
cat_listlist( struct listlist* l1, struct listlist *l2 ) {
struct listlist* l1_last = last_listlist(l1);
l1_last->next = l2;
l2->prev = l1_last;
return l1;
}

struct listlist*
treelist2listlist (struct listlist* prev, struct node_list* no, int n) {
struct listlist* new = NULL;
/*printf(" T2LL{ ");*/
switch(no->op) {
case _TERM_:
/*printf(" dbg: _term_: n: %d list( ",n); print_list(no->l); printf(" ) ");*/
new = (struct listlist*) malloc(sizeof(struct listlist));
new->l = no->l;
new->n = n;
if (prev)
prev->next = new;
new->prev = prev;
new->next = NULL;
break;
case _PLUS_:
/*printf(" dbg: plus_cat< ");*/
new = cat_listlist( treelist2listlist(NULL,no->left,n),
    treelist2listlist(NULL,no->right,n)
 );
/*printf( " > ");*/
if (prev)
prev->next = new;
new->prev = prev;
break;
case _MINUS_:
/*printf(" dbg: minus_cat< ");*/
new = cat_listlist( treelist2listlist(NULL,no->left,n),
    treelist2listlist(NULL,no->right,-n)
 );
/*printf(" > ");*/
if (prev)
prev->next = new;
new->prev = prev;
break;
default:
fprintf(stderr,"ERROR: treelist2listlist(): wrong no->op: %d\n",no->op);
```

```
exit(1);
break;
}
/*printf(" ret new: n: %d } ",new->n);*/
return new;
}

struct listlist*
contract_listlist(struct listlist *l) {
struct listlist *no = l, *no1, *no2, *no_first = l;
if (_DBG_) {
printf("CLL(): bp000: contrast_listlist() ... >>>>>\n");
print_listlist(l);
}
if (l == NULL) {
_DBG_&&printf("CLL(0): bp000-1: <<<<\n");
return l;
}
do {
if(_DBG_) {
printf("CLL: bp001: no->l = contract_list(no->l) >>>>\n");
print_list(no->l);
}
no->l = contract_list(no->l);
if(_DBG_) {
print_list(no->l);
printf("CLL: bp001-1: <<<<<<\n");
}
} while (no = no->next);
_DBG_&&printf("CLL(): bp000.1: all lists contracted ------- \n");
no = no_first = l;
do {
for (no1 = no->next; no1;) {
_DBG_&&printf("CLL: if eq_list ... \n");
if (eq_list(no->l,no1->l)) {
_DBG_&&printf("CLL: bp002: no->l .eq. no1->l: no->n += %d >>>>\n",no->n);
no->n += no1->n;
no1->prev->next = no1->next;
if (no2 = no1->next)
no2->prev = no1->prev;
_DBG_&&printf("CLL: bp0021: free_listlist_el(no1) >>>>>\n");
free_list(no1->l);
free((void*)no1);
_DBG_&&printf("CLL: bp0021-1: <<<<< \n");
no1 = no2;
}
else {
_DBG_&&printf("CLL: bp003: >>> <<<\n");
no1 = no1->next;
}
}
if (no->n == 0) {
_DBG_&&printf("CLL: bp004: no->n == 0 >>>>\n");
if (no->prev)
no->prev->next = no->next;
else
no_first = no->next;
if (no2 = no->next)
no2->prev = no->prev;
_DBG_&&printf("CLL: bp0041: free_listlist_el(no) >>>>>\n");
free_list(no->l);
free((void*)no);
_DBG_&&printf("CLL: bp0041-1 <<<<< \n");
no = no2;
}
else {
no = no->next;
}
} while (no);
_DBG_&&printf("CLL: bp000-1: <<<<<<<<<\n");
return no_first;
}

void
print_listlist (struct listlist *l) {
```

```
struct listlist *no = l;
if (!no) {
printf(" [NULL] ");
return;
}
do {
printf(" (%d)*[ ",no->n);
print_list(no->l);
printf(" ]");
} while (no = no->next);
return;
}

static
char
*_s_null_ = " [NULL] ",
*_s_buf_  = 0;


static
char*
_listlist2s (struct listlist *l, char *buf) {
struct listlist *no = l;
char *p = buf ? buf + strlen(buf) : 0;
if (!no) {
buf = (char*)realloc(buf, sizeof(char) * ((buf ? strlen(buf) : 0) + strlen(_s_null_) + 1));
if (!p)
p = buf;
sprintf(p,_s_null_);
return buf;
}
do {
buf = (char*)realloc(buf, sizeof(char) * ((buf ? strlen(buf) : 0) + 132));
if (!p)
p = buf;
_DBG_&&printf("_lislist2s(): (%d)*[ ...\n",no->n);
sprintf(p," (%d)*[ ",no->n);
_DBG_&&printf("_lislist2s(): buf = _list2s(...)\n");
buf = _list2s(no->l,buf);
buf = (char*)realloc(buf, sizeof(char) * ((buf ? strlen(buf) : 0) + 3));
p = buf + strlen(buf);
_DBG_&&printf("_lislist2s(): buf:%s: ... ]\n",buf);
sprintf(p," ]");
p = buf + strlen(buf);
_DBG_&&printf(" <... buf:%s:\n",buf);
} while (no = no->next);
return buf;
}

char*
listlist2string (struct listlist *l) {
/* (_DBG_)&&printf("listlist2string( %x ) ... \n", (unsigned int)l);*/
print_listlist(l); printf("\n .... \n");
return _listlist2s(l,0);
}

/*****************************************************************************/
```

Ж.9 re/v.h

```
int prt_id(char*);
```

Ж.10 re/v.c

```
#include "P/P.h"

int prt_id(char *s) {
struct type_P *p = s2t_P(s);
return p ? p->id : -1;
```

```
}
```

Ж.11 util/Makefile

```
%.o: %.c
cc $(CFLAGS) -c -o $@ $<

clean:
rm -f *.o *~
```

Ж.12 util/util.h

```
char*
s2hash(char* s, char* buf);

long
s2sum(char *s);
```

Ж.13 util/s2hash.c

```
#include <crypt.h>

char *
s2hex (char* s, char* buf) {
int j, a;
for (j = 0; *s; s++) {
if (*s < 0x10) {
buf[j++] = (*s) < 0x0a ? '0' + (*s) : 'a' + (*s) - 0x0a;
}
else {
a = (*s) / 0x10;
buf[j++] = a < 0x0a ? '0' + a : 'a' + a - 0x0a;
a = (*s) - a * 0x10;
buf[j++] = a < 0x0a ? '0' + a : 'a' + a - 0x0a;
}
}
buf[j] = '\0';
return buf;
}

char*
s2hash(char* s, char* buf) {
return s2hex(crypt(s,"$1$") + 4, buf);
}

long
s2sum(char *s) {
long sum = 0;
if (!s)
return 0;
while (*s)
sum = 256*sum + (*s++);
return sum;
}
```

Ж.14 Чтение словаря PPDL

Ж.15 Makefile

```
CFLAGS = -g -O0

l: lex.yy.o P/orcode.o P/dump.o
cc -o $@ $^
```

```
lex.yy.c: l.l tree.h settings.h
lex $<


P/%.o: P/%.c P/P.h tree.h settings.h
cd $$(echo $@ | sed 's,/.*$$,,') && make $$( echo $@ | sed 's,^P/,,' ) CFLAGS="$(CFLAGS)"

%.o: %.c
cc $(CFLAGS) -c -o $@ $<

clean:
rm -f l *.a *.o lex.yy.c  *~
for d in */ ; do cd $$d ; make clean ; cd .. ; done
```

Ж.16 l.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "tree.h"

#include "P/P.h"

int n_P;

char buf[2048], *p, *py;

enum { _val_, _inkey_, _ex_, _er_ } st;

int mult_val;

enum keytype key;

struct {
char *s;
enum keytype key, par;
} keys[] = {
{"ABBREVIATION",_ABBREVIATION_, _RECORD_},
{"AS"          ,_AS_          , _RECORD_},
{"AT"          ,_AT_          , _RECORD_},
{"NAME"        ,_NAME_        , _RECORD_},
{"USAGE"       ,_USAGE_       , _RECORD_},
{"COMMENT"     ,_COMMENT_     , _RECORD_},
{"COMMENT-TEX" ,_COMMENT_TEX_ , _RECORD_},
{"DESCRIPTION" ,_DESCRIPTION_ , _RECORD_},
{"ORCODE"      ,_ORCODE_      , _RECORD_},
{"IDENT"       ,_IDENT_       , _RECORD_},
{"Q"           ,_Q_           , _RECORD_},
{"MASS"        ,_MASS_        , _RECORD_},
{"RC"          ,_RC_          , _RECORD_},
{"RPP"         ,_RPP_         , _RECORD_},
{"CRPP"        ,_CRPP_        , _RECORD_},
{"PN"          ,_PN_          , _RECORD_},
{"SB"          ,_SB_          , _RECORD_},
{"DEN"         ,_DEN_         , _RECORD_},
{"DCH"         ,_DCH_         , _RECORD_},
{"QUESTION"    ,_QUESTION_    , _RECORD_},
{0, 0, 0                                 }
};


struct node
*no_root, *no_cur, *no_par;

struct node*
new_node(enum keytype type, struct node *par) {
struct node *no = (struct node*)malloc(sizeof(struct node));
no->type = type;
no->par = par;
no->ch = NULL;
```

```
no->nch = 0;
no->val = NULL;
no->nval = 0;
if (par) {
par->ch = (struct node**)realloc(par->ch, sizeof(struct node*) * (++ par->nch));
par->ch[ par->nch - 1 ] = no;
}
return no;
}

void
val_node(struct node* no, char* val) {
if (!no) {
fprintf(stderr,"ERROR: val_node(NULL, %s) called. EXIT\n", val);
exit(1);
}
no->val = (char**)realloc(no->val, sizeof(char*) * (no->nval + 1));
if (val)
while (*val == ' ' || *val == '\t' || *val == '\n')
val++;
no->val[ no->nval ] = (char*)malloc(sizeof(char) * (strlen(val) + 1));
strcpy(no->val[ no->nval++ ], val );
return;
}

void
free_tree (struct node* no) {
if (!no)
return;
if (no->val) {
for (; no->nval > 0; no->nval--)
free((void*)no->val[ no->nval - 1 ]);
free((void*)no->val);
}
if (no->ch) {
for (; no->nch > 0; no->nch --)
free_tree(no->ch[ no->nch - 1 ]);
free((void*)no->ch);
}
free((void*)no);
return;
}

char*
k2s(int k) { /*FIXME*/
int ik;
char *s = 0;
for (ik = 0; keys[ik].s; ik++) {
if (keys[ik].key == k) {
s = keys[ik].s;
break;
}
}
return s;
}

void
print_tree (struct node *no, int ident) {
int i;
for (i = 0; i < ident; i++)
putchar(' ');
if (!no) {
printf("[NULL]\n");
return;
}
printf("%s = ", k2s(no->type));
for (i = 0; i < no->nval; i++)
printf("%s; ", no->val[i]);
printf("\n");
for (i = 0; i < no->nch; i++)
print_tree(no->ch[i], ident+1);
return;
}
```

```
void
dump_tree(struct node *no) {
char *usage = 0;
int i;
if (no->type != _RECORD_) {
fprintf(stderr,"ERROR: dump_tree( %s ) called. EXIT\n", k2s(no->type));
exit(1);
}
for (i = 0; i < no->nch; i++) {
if (no->ch[i]->type == _USAGE_) {
usage = no->ch[i]->val[0];
break;
}
}
if (usage == 0) {
printf("WARNING: no USAGE key in the record. Don't know how to dump it\n");
return;
}
if (strcmp(usage,"P") == 0) {
dump_tree_P(no,n_P);
n_P++;
}
else {
printf("INFO: USAGE = %s, no dump required\n", usage);
}
return;
}

enum keytype
k2par(int k) {
int ik;
for (ik = 0; keys[ik].s; ik++) {
if (keys[ik].key == k) {
return keys[ik].par;
}
}
return _UNKNOWN_;
}

struct node*
find_par(enum keytype key, struct node* no) {
if (!no)
return NULL;
if (no->type == k2par(key))
return no;
else
return find_par(key,no->par);
}

int
s2k(char *s) {
int ik = _UNKNOWN_, k;
for (k = 0; keys[k].s; k++) {
if (strcmp(keys[k].s, s) == 0) {
ik = keys[k].key;
break;
}
}
return ik;
}

%}

%%

ABBREVIATION|AS|AT|NAME|USAGE|COMMENT|COMMENT-TEX|ORCODE|IDENT|Q|MASS|RC|RPP|CRPP|PN|SB|DEN|DCH|DESCRIPTION|QUESTION {
/* printf("yytext:%s:\n",yytext);*/
switch (st) {
case _val_:
for (py = yytext; *py; *p++ = *py++);
break;
case  _ex_ :
case  _er_ :
if ((key = s2k(yytext)) == _UNKNOWN_) {
fprintf(stderr,"ERROR: unknown key: %s. EXIT\n",yytext);
```

```
exit(1);
}
mult_val = 0;
st = _inkey_;
break;
case  _inkey_:
fprintf(stderr,"ERROR: spurious \"%s\" on the input. EXIT\n",yytext);
exit(1);
break;
}
}

"." {
switch(st) {
case _val_:
*p++ = *yytext;
break;
case _ex_:
case _er_:
fprintf(stderr,"ERROR: spurious '.' after ';' or '*E'. EXIT\n");
exit(1);
break;
case _inkey_:
mult_val = 1;
break;
}
}

"=" {
switch (st) {
case  _val_:
*p++ = *yytext;
break;
case  _ex_:
p = buf;
*p++ = *yytext;
st = _val_;
break;
case  _er_:
fprintf(stderr,"ERROR: spurious '=' after '*E'. EXIT\n");
exit (1);
break;
case  _inkey_:
if (mult_val) {
;
}
else {
if (no_par = find_par(key,no_cur)) {
no_cur = new_node(key,no_par);
}
else {
fprintf(stderr,"ERROR: can't find parent for child %s. EXIT\n",
k2s(key)
);
exit(1);
}
}
p = buf;
st = _val_;
break;
}
}

";" {
switch (st) {
case  _val_:
if (p > buf && *(p-1) == '\\') {
*p++ = *yytext;
}
else {
*p = '\0';
printf("key: %s%sval: %s\n",k2s(key), mult_val ? " +" : "  ",  buf);
val_node(no_cur,buf);
p = buf;
st = _ex_;
```

```
}
break;
case  _ex_:
break;
case  _er_:
fprintf(stderr,"ERROR: spurious ';' after '*E'. EXIT\n");
exit(1);
break;
case  _inkey_:
fprintf(stderr,"ERROR: spurious ';' after '='. EXIT\n");
exit(1);
break;
}
}

"*E" {
switch (st) {
case  _val_:
for (py = yytext; *py; *p++ = *py++);
break;
case  _ex_:
key = _NOKEY_;
st = _er_;
printf("EOR, current record: --\n");
print_tree(no_root,0);
printf ("Dumping ...\n");
dump_tree(no_root);
printf("====================================\nNew record ..........\n");
free_tree(no_root);
no_par = no_cur = no_root = new_node(_RECORD_, NULL);
break;
case  _er_:
break;
case  _inkey_:
fprintf(stderr,"ERROR: spurious '*E' after key. EXIT\n");
exit(1);
break;
}
}

[ \t\n] {
switch (st) {
case  _val_:
*p++ = *yytext;
break;
case  _ex_:
case  _er_:
case  _inkey_:
break;
}
}

[^ \t\n] {
switch (st) {
case  _val_:
*p++ = *yytext;
break;
case  _ex_:
st = _val_;
*p++ = *yytext;
break;
case  _er_:
fprintf(stderr,"ERROR: spurious '%c' after '*E'. EXIT\n",*yytext);
exit(1);
case  _inkey_:
fprintf(stderr,"ERROR: spurious '%c' after key. EXIT\n",*yytext);
exit(1);
}
}




%%
int yywrap(void) {
```

```
return 1;
}


int  main(void) {
n_P = 0;
st = _er_;
no_par = no_cur = no_root = new_node(_RECORD_,NULL);
yylex();
return 0;
}
```

Ж.17 tree.h

```
enum keytype {
_UNKNOWN_,
_NOKEY_,
_RECORD_,
_ABBREVIATION_,
_AS_,
_AT_,
_NAME_,
_USAGE_,
_COMMENT_,
_COMMENT_TEX_,
_DESCRIPTION_,
_ORCODE_,
_IDENT_,
_Q_,
_MASS_,
_RC_,
_RPP_,
_CRPP_,
_PN_,
_SB_,
_DEN_,
_DCH_,
_QUESTION_
};

struct node {
enum keytype type;
char **val;
int   nval;
struct node **ch, *par;
int   nch;
};
```

Ж.18 P/Makefile

```
orcode.o: orcode.c ../tree.h

dump.o: dump.c ../tree.h

%.o: %.c
cc $(CFLAGS) -c -o $@ $< -I. -I..

clean:
rm -f *.o *~
```

Ж.19 P/P.h

```
int
exp_orcode ( char *s,
int *bar,
int *lep,
int *S,
int *C,
```

```
int *B,
int *P,
int *heavy_nuc,
int *class,
double *I,
double *Q,
double *m
     );


void
dump_tree_P (struct node *no, int id);
```

ЖK.20 P/orcode.c

```
#include<stdio.h>
#include<stdlib.h>

#include "tree.h"

int
exp_orcode ( char *s,
int *bar,
int *lep,
int *S,
int *C,
int *B,
int *P,
int *heavy_nuc,
int *class,
double *I,
double *Q,
double *m

     ) {

int nm, ns, i;
enum { baryon, strange, mass, iso, anti, charge, charge3, end } check;

*bar =  *lep = *heavy_nuc = *class =
*S = *C = *B = *P = 0;
*I = *Q = 0.;
*m = -1.;

if (!s) {
fprintf(stderr,"\nERROR: orcode( NULL ). EXIT\n");
return -1;
}

printf("%s\n",s);

for (check = baryon;;) {

        if (check == end) {
       if (*s) {
       fprintf(stderr,"\nERROR: extra trailing digits: %s\n",s);
       return -1;
       }
       else {
break;
}
       }

if (!*s) {
fprintf(stderr,"\nERROR: more digits expected.\n");
return -1;
}

if (check == baryon) {
          switch (*s) {
       case '2':
*class = 79;
```

```
       printf("for abbreviations from classes 7) and 9). exit.\n");
return 0;
break;
       case '4':
*class = 6;
       printf("for abbreviations from class 6 ???\n");
return 0;
break;
       case '6':
       printf("gamma:\n");
*bar = 0;
*lep = 0;
       s++;
       check = strange;
       break;
       case '8':
       printf("lepton:\n");
*bar = 0;
*lep = 1;
s++;
       check = strange;
       break;
       case '1':
       if (*(s+1) == '6') {
       *bar = 9999;
*lep = 0;
       printf("heavy nucleus, |bar| > 4\n");
       *heavy_nuc = 1;
       }
       else if (*(s+1) == '8') {
       *bar = 0;
*lep = 0;
       printf("DD or NON-RES\n");
       }
       else {
       *bar = *(s+1) - '0';
*lep = 0;
       printf("baryon/meson: bar = %d\n",*bar);
       }
       s += 2;
check = strange;
       break;
       default:
       fprintf(stderr,"\nERROR: wrong orcode[0] = %d. EXIT\n", s[0] - '0');
return -1;
          }
       }
       else if (check == strange) {
   ns = (*s) - '0';
          if (*bar != 0) {
if (ns == 2) {
printf("Z* baryon: exotic I= unspecified, S=+1 baryon of unspecified mass\n");
*S = 1; *C = 0; *B = 0;
}
else if (ns <= 7) {
*S = 7 - ns; *B = *C = 0;
       printf("baryon except Z*, S = %d\n",*S);
}
else if (ns == 8) {
       printf("c-baryon\n");
       *S = 0; *C = 1; *B = 0;
}
else if (ns == 9) {
printf("b-baryon\n");
*S = 0; *C = 0; *B = 1;
}
else {
fprintf(stderr,"\nERROR: wrong NS: %c\n",*s);
return -1;
}
          }
   else {
if (ns == 6) {
       printf("c-meson\n");
       *B = 0; *S = 0; *C = 1;
```

```
}
else if (ns == 7) {
       printf("b-meson\n");
       *B = 1; *S = 0; *C = 0;
}
else {
       *S = *s - '0' - 3; *B = *C = 0;
       printf("meson: S = %d\n",S);
}
          }
          check = mass;
          s ++;
       }
       else if (check == mass) {
       for (i = 0, nm = 0; i < 4 && *(s+i); i ++)
       nm = 10*nm + *(s+i) - '0';
       if (i < 4) {
       fprintf(stderr,"\nERROR: < 4 NM digits. EXIT\n");
return -1;
       }
       printf("INFO: NM = %d\n",nm);
       if (*bar == 0) {
       *m = nm - 1000;
       printf("meson: m = %lf MeV\n",*m);
       }
       else if (abs(*bar) <= 4) {
       *m = 9000 - nm;
       printf("baryon(|B|<=4): m = %lf MeV\n",*m);
       }
       else {
       printf("xz: B = %d NM = %d\n", *bar,nm);
       }
       s += 4;
       if (!(*heavy_nuc))
       check = iso;
       else
       check = charge3;
       }
       else if (check == iso) {
       *I = (double)(*s - '0' - 3) / 2.;
       printf("I = %lf\n",I);
       s ++;
       check = anti;
       }
       else if (check == anti) {
       switch (*s) {
       case '7':
*P = 1;
       printf("particle\n");
       break;
       case '3':
*P = -1;
       printf("anti-particle\n");
       break;
       case '6':
*P = 2;
       printf("KS\n");
       break;
       case '4':
*P = 3;
       printf("KL\n");
       break;
       default:
       fprintf(stderr,"\nWARNING: undefined NP = %c\n",*s);
       break;
       }
       s ++;
       check = charge;
       }
       else if (check == charge) {
       *Q = *s - '0' - 5;
printf("Q = %lf\n",*Q);
       s ++;
       check = end;
       }
```

```
       else if (check == charge3) {
       for (i = 0, *Q = 0.; i < 3 && *(s+i); i++)
       *Q = 10*(*Q) + *(s+i) - '0';
       if (i < 3) {
       fprintf(stderr,"\nERROR: 3digits expected for heavy nucleus' charge\n");
return -1;
       }
       printf("heavy nuc.: Q = %lf\n",*Q);
       s++;
       check = end;
       }
       else {
       fprintf(stderr,"\nERROR: unknown check = %d\n",check);
return -1;
       }
}

if (*bar == 0) {
if (*I == -1.5)
*I = 0;
if (*Q == -5.)
*Q = 0.;
}

return 0;
}
/***********************************************************

 Each abbreviation has a number - OR-code, which serve to
 order names in final state, in this number, following numbers
 are packed from left to right: NB, NS, NM, II, NP, NC.

 NB (2 FIGURES)= B+10 for mesons and baryons with /B/ <=4; B -
 baryon number
 = 2 for abbreviations from classes 7) and 9)
 = 4 for abbreviations from class 6
 = 6 for PHOTON
 = 8 for LEPTONS
 = 16 for heavy nuclei with /B/>4
 = 18 for abbreviations DD, NON-RES
 NS (1 FIGURE ) = 3+/S/ for mesons, where S-strangeness
 = 7-/S/ for baryons except Z*
 = 2 for Z*
 = 6 for charmed mesons
 = 7 for beauty mesons,
 = 8 for charmed baryons.
 = 9 for beauty baryons.
 NM (4 FIGURES) = 9000-M for baryons with /b/<=4, where M - mass
 in MeV.
 = 1000+M for mesons
 = 1000+10*|B|+5 for nuclei with /B/>4.
 = 1000+10*|B|+5+N for isotopes where -5<=N<=4
 NI (1 FIGURE ) = 2*I+3, where I - isotopic spin
 NP (1 FIGURE ) = 7 for particles, 3 for antiparticles, 6 for
 KS, 4 for KL.

 NC (1 FIGURE ) = C+5, where C - charge. For heavy nucleus
 (NB=16) NI, NP, NC contain charge of nuclei.

************************************************************/

/*
int
main (int argc, char** argv) {
orcode(argv[1]);
return 0;
}
*/
```

ЖK.21 P/dump.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include "tree.h"
#include "P.h"

void
dump_tree_P (struct node *no, int id) {
int ia = -1, i, j, orcode = -1, ident = -1;
int bar,lep, S, C, B, P, heavy_nuc, class;
double m_or = -1., mass = -1., I, Q;
printf("INFO: dump particle: ..............\n");
for (i = 0; i < no->nch; i++) {
switch (no->ch[i]->type) {
case _ABBREVIATION_:
printf("  abbr:");
ia = i;
for (j = 0; j < no->ch[i]->nval; j++)
printf(" %s;",no->ch[i]->val[j]);
printf("\n");
break;
case _ORCODE_:
for (j = 0; j < no->ch[i]->nval; j++) {
printf(" .... or: %s\n",no->ch[i]->val[j]);
if (j == 0) {
orcode = atoi(no->ch[i]->val[j]);
exp_orcode( no->ch[i]->val[j],
   &bar,&lep,
   &S, &C, &B,
   &P,
   &heavy_nuc,&class,
   &I,&Q,
   &m_or
 );
printf(" ........\n");
}
}
break;
case _MASS_:
if (no->ch[i]->nval > 1)
printf("WARNING: > 1 masses for a particle, taking the 1st\n");
mass = atof(no->ch[i]->val[0]);
break;
case _IDENT_:
if (ident != -1)
fprintf(stderr,"WARNING: > 1 IDENT's for a particle\n");
ident = atoi(no->ch[i]->val[0]);
break;
default:
break;
}
}
if (m_or != mass)
printf("WARNING: mass = %lf != mass from orcode: %lf\n", mass, m_or);
printf("PRT(or, m, bar,lep, S C B, P, I Q, hnuc, class):\n");
printf("PRT: %d %d %d  %d %d %d %d  %d %d %d  %lf %lf %lf  %d",
id, bar, lep,
S,C,B,0,
P,heavy_nuc,class,
mass,I,Q,
orcode
);
if (ia >= 0) {
for (j = 0; j < no->ch[ia]->nval; j++)
printf(" %s",no->ch[ia]->val[j]);
printf("\n");
}
return;
}
```