I-DLV / DLV2  specific features
External Computations
- allow the introduction of new constants from «outside» (besides the Herbrand Universe and beyond arithmetics)
- allow the evaluation of atoms with respect to explicitly defined external semantics (besides «classic» interpretations and models)
External Sources of Computations in I-DLV / DLV2

**External Atom**

\[ \&p(t_0, \ldots, t_n; u_0, \ldots, u_m) \]

- \( n + m > 0 \)
- \( \&p \) is a external predicate
- \( t_0, \ldots, t_n \) are input terms
- \( u_0, \ldots, u_m \) are output terms
External semantics that define the truth-values of external atoms is provided via PYTHON functions.

For each external predicate of the form

\[ \&p(t_0, \ldots, t_n; u_0, \ldots, u_m) \]

a Python function called «p» must be defined, with \( n \) input parameters and \( m \) output parameters.

Python functions are used by I-DLV in order to completely evaluate external atoms as true or false.
Restrictions:

• The definition of the external functions must comply to Python version 3+.

• Each occurrence of an external predicate of the form
\[ p(t_0, \ldots, t_n; u_0, \ldots, u_m) \]
must appear with the same number of parameters (input/output) throughout the program.
def sum(X, Y):
    return X+Y

compute sum(X, Y, Z) :- number(X),
                        number(Y), sum(X, Y; Z).
def rev(S):
    return S[::-1]
External Atoms: returned values

A conversion from Python types to ASP-Core-2 terms is needed.

Value returned by the Python function can be

- Numeric
- String
- Boolean

Default policy:
Integer → numeric constants
All other values → symbolic constants,
    if it is not possible → string constants
I-DLV allows the user to customize the mapping policy of a particular external predicate by means of explicit directives.

```
#external_predicate_conversion(&p,type: TO₁, . . . , TOₙ).
```

This specifies the sequence of type conversions for an external atom «p» featuring \( n \) output terms.

Directives can be specified anywhere in the ASP program and have global effect.
A conversion type can be:

- `@U_INT` unsigned integer
- `@UT_INT` truncated to an unsigned integer
- `@T_INT` truncated to an integer
- `@UR_INT` rounded to an unsigned integer
- `@R_INT` rounded to an integer
- `@CONST` string without quotes
- `@Q_CONST` quoted string

Example: force output of the sum external predicate to a quoted string:

```
#external_predicate_conversion(&compute_sum,type: Q_CONST).
```
An external built-it can be

- **Functional**: returns a single «tuple» for each combination of the input values
- **Relational**: returns a set of «tuples» for each combination of the input values

In general, a functional external atom with $m > 0$ output terms must return a (Python) sequence of $m$ values. If $m = 1$, output can be a sequence consisting of a single value, or a single value (see «compute_sum» before). If $m = 0$, the Python function must return a boolean value.

A relational external atom with $m > 0$ output terms is defined by a Python function returning a sequence of $m$-sequences (i.e., each inner sequence must feature $m$ values).
Ex.: Given a number X, returns all its prime factors as a sequence of single values.

```python
def cpf(n):
    i = 2
    factors = []
    while i * i <= n:
        if n % i:
            i += 1
        else:
            n //= i
            factors.append(i)
    if n > 1:
        factors.append(n)
    return factors
```
Interoperability
**Problem**

When you have to perform complex reasoning tasks but the data is available in database relations, or the output must be permanently stored in a database for further elaborations.

**Solution**

- Providing ASP with an easy way to access distributed data
- Most of the existing systems are tailored on custom DBMSs.
I-DLV inherits from DLV directives for importing/exporting data from/to relational DBs:

- Handle input and output with directives:

  #import_sql(databasename, username, password, query, predname, predarity, typeConv).

  #export_sql(databasename, username, password, predname, predarity, tablename).
Graph Database Support

Data can be imported via SPARQL queries
Supports Local DBs in RDF files and remote SPARQL EndPoints

#import local sparql(rdf_file, query, predname, predarity, typeConv).
#import remote sparql(endpnt_url, query, predname, predarity, typeConv).
KRR with external computation

Given a list of students, a list of topics and questions related to the topics, along with corresponding student answers, we want to automatically assign a score to each student.

Score is computed depending on the scores obtained to questions on each topic; each topic may have higher/lower relevance w.r.t. others. Score computation can be involved → use an external predicate

Data Model:

Student(ID)

Topic(T)

Question(ID, Topic, Possible_Answers, Correct_Answer)

answer(Student_ID, Question_ID, Given_Answer)
KRR with external computation

correctAnswers(St, To, N) :- topic(To), student(St), N = #count{QID : question(QID, To, Tx, Ca), answer(St, QID, Ca)}. % # of correct answers for St

wrongAnswers(St, To, N) :- topic(To), student(St), N = #count{QID : question(QID, To, Tx, Ca), answer(St, QID, Ans), Ans! = Ca}. % # of wrong answers for St

topicScore(St, To, Sc) :- correctAnswers(St, To, Cn), wrongAnswers(St, To, Wn), &assignScore(To, Cn, Wn; Sc). % score for St on topics To

#external predicate conversion(predicate=&assignScore,type=R INT). % round score to an integer

testScore(St, Sc) :- student(St), Sc = #sum{Sc : topicScore(St, To, Sc)}. % total score for St
A possible implementation of the function computing the scores:

```python
def assignScore(topic, numCorrectAns, numWrongAns):
    if (topic=="ComputerScience" or topic=="Mathematics"):
        return numCorrectAns * 2 - numWrongAns * 0.5
    return numCorrectAns - numWrongAns * 0.5
```
Retrieve all questions from an external DB:

#import sql(relDB, "user", "pwd", "SELECT * FROM question", question, type:U_INT,Q_CONST,Q_CONST,Q_CONST).

user e pwd are needed to access the external DB, the quoted string is the SQL query that fills the extension of predicate « question » with tuples consisting of an integer and three quoted strings each.
Retrieve answers from all students from an external XML file:

```
# import local sparql("answers.rdf",
   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   PREFIX my: <http://sample/rdf#>
   SELECT ?St, ?Qe, ?Ans
      ?X my:question ?Qe. ?X my:answer ?Ans."},
   answer, 3, type:U_INT, U_INT, Q_CONST).
```

`answers.rdf` contains the answers, the long quoted string is the SPARQL query that fills the « answer » relation with two integers and a quoted string for each tuple.