External Computations and Interoperability in the new DLV Grounder

F. Calimeri[1,2], D. Fuscà[1], S. Perri[1], J. Zangari[1]
[1] Department of Mathematics and Computer Science, University of Calabria, Rende (CS), Italy
[2] DLVSystem Srl, Rende (CS), Italy
{calimeri, fusca, perri, zangari}@mat.unical.it

AI*IA 2017

UNIVERSITÀ DELLA CALABRIA

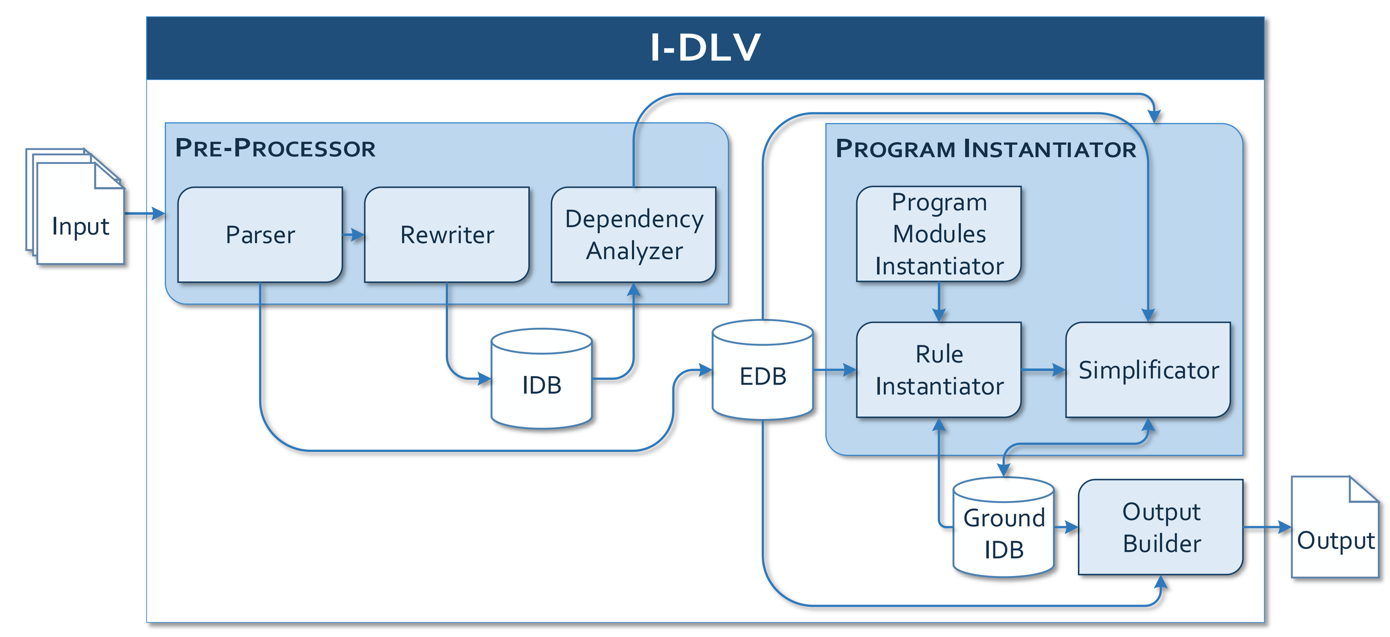
ANSWER SET PROGRAMMING

- Powerful declarative paradigm for KRR
- State-of-the art systems perform two steps: **Grounding** & **Solving**
- Reliable and efficient ASP systems available:
  - **Monolithic Systems**: DLV, CLINGO
  - **Standalone grounders and solvers:**
    - LPARSE, GRINGO, I-DLV
    - SMODELS, CLASP, WASP
- Several ASP applications in academy and industry

Design and Implementation of the I-DLV Grounder: Optimizations, Customizability and Interoperability

DOCTORAL CONSORTIUM - J. Zangari

- Fully supports ASP-Core-2 syntax
- Able to interoperate with state-of-the-art solvers
- Generalizes and improves DLV grounding strategy
- Incorporates new optimization techniques
- Devoted to flexibility and customizability
- Full-fledged Deductive Database system
- Along with the solver WASP, I-DLV has been integrated in the new version of DLV recently released



I-DLV NOVEL FEATURES: MOTIVATIONS

- Ease the interoperability and integration with external systems
- Accommodate external sources of computation and value invention within ASP programs

EXTERNAL SOURCES OF COMPUTATION

I-DLV supports calls to **Python** scripts via **external atoms** of form:

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

- $n+m \geq 0$
- $\&p$ is an external predicate
- $t_0,\ \ldots,\ t_n$ are input terms
- $u_0,\ \ldots,\ u_m$ are output terms

Semantics provided externally by means of «oracle» functions defined in Python.

EXAMPLE: REVERSING A STRING

reverseWord(Y):- word(X), &reverse (X;Y).

```
def reverse(s):
return [::-1]
```

← oracle Python function for *&reverse*.

INTEROPERABILITY MEANS

I-DLV supports **explicit native directives** for connecting with DBs.

Relational Databases

Data can be imported/exported from/to relational DBs via SQL queries.

```
#import_sql(db_name,"user","pwd","query",pName
[, type conv]).
```

```
#export_sql(db_name,"user","pwd", pName,pArity,tName,
"REPLACE where SQL-Condition").
```

Graph Databases

Data can be imported via SPARQL queries, both from local DBs in RDF/XML files and remote SPARQL end-points.

```
#import_local_sparql("rdf_file","query",pName,pArity,
[,typeConv]).
```

```
#import_remote_sparql("endpnt","query",pName,pArity,
[,typeConv]).
```

KRR WITH NOVEL FEATURES

Consider the problem of automatically assigning a score to students after an assessment test: given a list of students, a list of topics, and a set of questions regarding the given topics along with corresponding student answers, we want to determine the score of each student. It could be modelled as follows:

```
#import_sql(relDB, "user", "pwd",
  "SELECT * FROM question", question).
#import_local_sparql("answers.rdf",
  PREFIX my: <http://sample/rdf\#>
  SELECT ?St, ?Qe, ?Ans
  WHERE {?X rdf:type my:test.
    ?X my:student ?St. ?X my:question ?Qe.
    ?X my:answer ?Ans.}",
  answer, 3).
```

correctAnswers(St, To, N) :- topic(To), student(St), N =
#count{QID: question(QID, To, Tx, Ca), answer(St, QID, Ca)}.
wrongAnswers(St, To, N) :- topic(To), student(St), N =
#count{QID: question(QID, To, Tx, Ca),
answer(St, QID, Ans),Ans!=Ca}.
topicScore(St, To, Sc) :- correctAnswers(St, To, Cn),
wrongAnswers(St, To, Wn), ***&assignScore(To, Cn, Wn; Sc)****.*
testScore(St, Sc) :- student(St), Sc =
#sum{Sc: topicScore(St, To, Sc)}.

where the semantics of ***&assignScore*** could defined as:

```
def assignScore(topic, numCorrectAns, numWrongAns):
  if(topic==``ComputerScience''
    or topic==``Mathematics''):
    return numCorrectAns*2-numWrongAns*0.5
  return numCorrectAns-numWrongAns*0.5
```

ADDITIONAL DETAILS

I-DLV is an open source project, further details are available at: