UNIVERSITÀ
DELLA CALABRIA

Normal Form Nested Programs

In Disjunctive Logic Programming (DLP) the heads (resp. the bodies) of rules are disjunctions (resp. conjunctions) of simple constructs, viz. atoms and literals. DLP, under the answer set semantics, is established as an important tool for knowledge representation and reasoning. Lifschitz, Tang and Turner extended the answer set semantics (in the propositional or ground case) to a class of logic programs where the heads and the bodies of rules are nested expressions. These expressions are formed from negation-as-failure literals, conjunction and disjunction, nested arbitrarily. This class of programs, called nested logic programs, generalizes the class of (ground) disjunctive logic programs. Moreover, nested logic programs can be transformed into disjunctive logic programs. These results allow for evaluating ground nested logic programs using DLP systems, such as DLV, GnT, or Cmodels3. These methods introduce new symbols implying that the result is not equivalent in the classical sense to the original program. Anyway, there is a one-to-one correspondence between the answer sets. However, given that these transformations work only for ground nested logic programs, one of the strongest features of logic programming, namely variables, cannot be used in problem representations. This restriction limits the suitability of nested logic programs in many application domains, especially when reasoning is to be done on large numbers of input facts. Unfortunately, a generalization of these techniques to programs with variables is not straightforward. A major obstacle is domain dependence, a property first studied in the realm of database systems. Essentially, when variables are present, the semantics of rules will in general depend on the particular domain that is chosen for their interpretation. This entails several undesirable effects such as a strong dependence on the context, even if this context is completely independent, issues with finitenessand in general unintuitive semantics. When one would just add variables to the ground method, one easily obtains domain dependent rules.

Domain dependence is also an issue in DLP, and in this context (as in databases) a syntactic requirement is imposed on programs, which guarantees domain independence and therefore avoids all of the problems that domain dependence entails. This requirement is known as safety, which for DLP rules means that each variable in a rule must occur in a positive body literal. Motivated by these considerations, in the thesis non-ground DLPs are extended to a class of programs, in which rule heads are formulas in disjunctive normal form consisting of atoms, and in which the rule bodies are formulas in conjunctive normal form consisting of literals. These programs are referred to as Normal Form Nested (NFN) programs, and are different to nested logic programs, since they may contain variables. We study semantic and domain dependence properties of this class of programs, and provide a definition of safety (which guarantees domain independence) and a polynomial translation from NFN programs to DLP, which maintains safety. The need for extending DLP with conjunction in the heads and disjunction in the body arises quite often in real world applications.

As an example we show a problem that we met in a real-world data-integration application.
Consider a global relation pers(ID, name, surname, age) (for persons) with a key-constraint on the first attribute ID. To perform consistent query answering, when two tuples share the same key, the relation

person is "repaired" by intensionally deleting one of them. In DLP, this is obtained by the following rules (where p stands for deleted tuples, and p' is the resulting consistent relation on which query answers are computed).

```
p(I,N,S,A) v p(I,M,T,B) :- pers(I,N,S,A), pers(I,M,T,B),N<>M.
p(I,N,S,A) v p(I,M,T,B) :- pers(I,N,S,A), pers(I,M,T,B),S<>T.
p(I,N,S,A) v p(I,M,T,B) :- pers(I,N,S,A), pers(I,M,T,B),A<>B.
p'(I,N,S,A) :- pers(I,N,S,A), not p(I,N,S,A).
```

The first rule deletes one of two tuples sharing the same key and having different names. Similarly, the second rule deletes one of two tuples sharing the same key and having different surnames. Finally, the third rule deletes one of two tuples if they have the same ID but different ages. The last rule
builds the repaired database. The first three DLP rules can be equivalently encoded by a single NFN rule, which is much more succinct and readable:

```
p(I,N,S,A) v p(I,M,T,B) :- pers(I,N,S,A), pers(I,M,T,B),(N<>M v S<>T v A<>B):
```

In detail, the NFN rule deletes one of two tuples if the tuples have the same ID and different names, different surnames or different ages.