Appendix C
Introduction to SPARC

In this appendix we give a brief introduction to SPARC (Balai, Gelfond, and Zhang, 2013), an extension of ASP that allows for the explicit representation of sorts and CR-rules (see Section 5.5). A more detailed description of the language and the corresponding software system can be found at `https://github.com/iensen/sparc/wiki`.

A SPARC program consists of three parts: *sort definitions*, *predicate declarations*, and *program rules*; the keywords `sorts`, `predicates`, and `rules` begin the corresponding sections. The first part explicitly defines sorts of objects of the program's domain, the second declares the program's predicates and the sorts of their parameters, and the third consists of a collection of standard ASP and cr-rules. For example, the family program from Section 4.1.1 can be rewritten in SPARC by

1. Explicitly specifying the sorts of the program.

```
sorts

#person = {john, sam, alice, bill, bob}.
#gender = {male, female}.
```

   Sort names of SPARC are identifiers preceded by #.
2. Declaring program predicates and their parameters.

```
predicates

father(#person,#person).
mother(#person,#person).
gender_of(#person,#gender).
parent(#person,#person).
child(#person,#person).
brother(#person,#person).
```

3. Removing from the program rules 1–7, 18, and 32, which contain the information already present in the sort definition; and removing the sort information from the bodies of other rules:

```
rules

father(john,sam).
mother(alice,sam).
...
-father(X,Y):- father(Z,Y), X != Z.
-father(X,Y) :- not father(X,Y).
```

Note that the last rules are obtained by dropping sort information `person(X)` from

```
-father(X,Y) :- person(X),
                father(Z,Y),
                X != Z.
```

and `person(X), person(Y)` from

```
-father(X,Y) :- person(X), person(Y),
                not father(X,Y).
```

The SPARC solver uses the first two sections of the program to automatically extract and use the necessary sort information. This solves the issue with rule safety, shortens the rules, and allows implementation of checking for sort errors in the program.

As mentioned earlier the rule part of a SPARC program can contain cr-rules. For instance, the first CR-Prolog program from Section 5.5 can be written in SPARC as follows:

```
1  sorts
2  #s1 = {a}.
3
4  predicates
5  p(#s1).
6  q(#s1).
7
8  rules
9  p(a) :- not q(a).
10 -p(a).
11 q(a):+.
```

In addition to the explicit set notation for defining sorts, SPARC allows a number of other constructs that facilitate such definitions. For instance, to define a sort $\#s1$ consisting of numbers from 1 to 100, one can simply say

```
#s1 = 1..100.
```

New sorts can be obtained from old ones using set-theoretic operators. For instance, given sorts $\#undergraduate$ and $\#graduate$, sort $\#student$ can be defined as

```
#student = #undergraduate + #graduate
```

where $+$ stands for set-theoretic union. To define a sort $\#s2$ consisting of names b1, b2, ..., b100, one can use the concatenation operator. The corresponding definition is

```
#s2 = [b][1..100]
```

or, if $\#s1$ is already defined,

```
#s2 = [b]#s1.
```

If f is an identifier and $\#s1$ and $\#s2$ are given, then the new sort

```
#s3 = f(#s1,#s2)
```

consists of records of the form $\mathrm{f(X, Y)}$ where X is in $\#s1$ and Y is in $\#s2$. For instance, sort $\#default\_name$ used by the uncaring John program from Section 5.1.1 as a parameter of predicate ab can be defined in SPARC as

```
#default_name = d_cares(#person,#person).
```

An accurate and complete description of constructs used for defining sorts of SPARC can be found in the SPARC manual. The manual also contains information on downloading and running the SPARC solver and its query-answering system. You can find it at `https://github.com/iensen/sparc/wiki`. This page also contains a link to a collection of useful examples, many of which are SPARC versions of programs from this book.