

How to convert Eclipse examples into Sicstus examples and vice versa

Matthias Knorr

October 21, 2004

The main intention for writing this short guide is to give some hints for changing an Eclipse example into a program in Sicstus, simply because FLUX and thus also all examples were at first written in Eclipse. Nevertheless, it should be possible to use it also the other way around.

1 Defining additional constraints

In general, you can add constraints to the already existing ones in `fluent.chr`, respectively `fluent.pl`, either directly in these files or in a separate file. In case you want to store them in a separate file there are several things you should take into account:

In Eclipse, constraints are stored in `chr` files which are called using the `chr` command (cf. `flux.pl` in the FLUX system for Eclipse).

In Sicstus, you simply use `pl` files but there are some rules. At first, each file containing constraints also has to contain the definition of a handler (cf. `fluent.pl` in the FLUX system for Sicstus). But if you load two files (e.g. `fluent.pl` and your own file) containing handler definitions into the module `user`, they clash. So instead, the additional file has to be defined as a module using the command

```
:- module(name,exportlist).
```

where `name` is freely chosen and `exportlist` a list of those predicates (constraints) c/n including their arity n which you want to use outside the module. This file can now be loaded as usual but you should notice that you can use predicates and constraints defined outside the module only via the call `module : predicate`. (e.g. having loaded `flux.pl` you can put calls like `user : notholds(F, Z)` into your separate module)

Finally, note that putting additional constraints in a separate file slows down the system a little bit whereas adding them to the `fluent` file results in a loss of structuredness.

2 Constraint handling

This section shows some differences concerning constraint handling and constraint solving.

The most important thing is the difference of the syntax for domain constraints. Sicstus allows only X in $n..m$ whereas in Eclipse you may write

$X :: n..m$ or $X :: [n..m]$ and you can even collect several variables with the same restrictions: $[X, Y] :: n..m$.

FLUX and the corresponding constraint solver are intended to handle integer values. Nevertheless, apart from some special cases, Eclipse can handle also strings which is quite nice since programs become more readable. But these special cases may change the behavior of the system and it is quite difficult to find these errors. So it is instead suggested to use a kind of enumeration-like representation of your entities which is necessary in case of Sicstus anyway since the system is unable to deal with arbitrary terms as arguments.

A specialty of Sicstus is that the guard of a constraint handling rule is considered as logical *or* and vice versa, which means that if you have a CHR without a guard but an logical *or* (e.g. ;) in the body then this is considered to be the guard of the rule which obviously changes the meaning of the whole constraint. So in this case it is necessary to add a guard *true*.

3 Further differences

This is just an incomplete list of predicates which are different.

- The syntax for loading additional modules is different (cf. `flux.pl` in Eclipse / Sicstus). Also the names of the modules and their contents are usually not the same.
- The predicate *subsumeschk/2* in Sicstus corresponds to *instance/2* in Eclipse where the order of the arguments is changed.
- Random numbers are generated differently (cf. the program `wumpus` for Sicstus and Eclipse and the corresponding simulator). In fact, you need to load an additional module in case of Sicstus.
- The usual *assert/1* in Eclipse corresponds to *assertz/1* in Sicstus because of the different behavior of Sicstus and Eclipse when adding dynamic clauses.
- *currentpredicate/2* in Sicstus corresponds to *ispredicate/2* in Eclipse.