



SSS Integrated Development Environment Project

- *What is the problem to be solved by SSS System ?*

```

/* 8-queens problem */
?- literalize( global(counter)).
?- literalize( vertical(pos)).
?- literalize( queen_position(h_pos, v_pos)).

context(all).

?- add_context(all).

?- make( global(1, counter 1)).

?- make( vertical(1, pos 1)).
?- make( vertical(1, pos 2)).
?- make( vertical(1, pos 3)).
?- make( vertical(1, pos 4)).
?- make( vertical(1, pos 5)).
?- make( vertical(1, pos 6)).
?- make( vertical(1, pos 7)).
?- make( vertical(1, pos 8)).

conflicts(H1,V1,H2,V2):- H1=H2,!.
conflicts(H1,V1,H2,V2):- V1= V2,!.
conflicts(H1,V1,H2,V2):- abs(H1-H2)=abs(V1-V2).

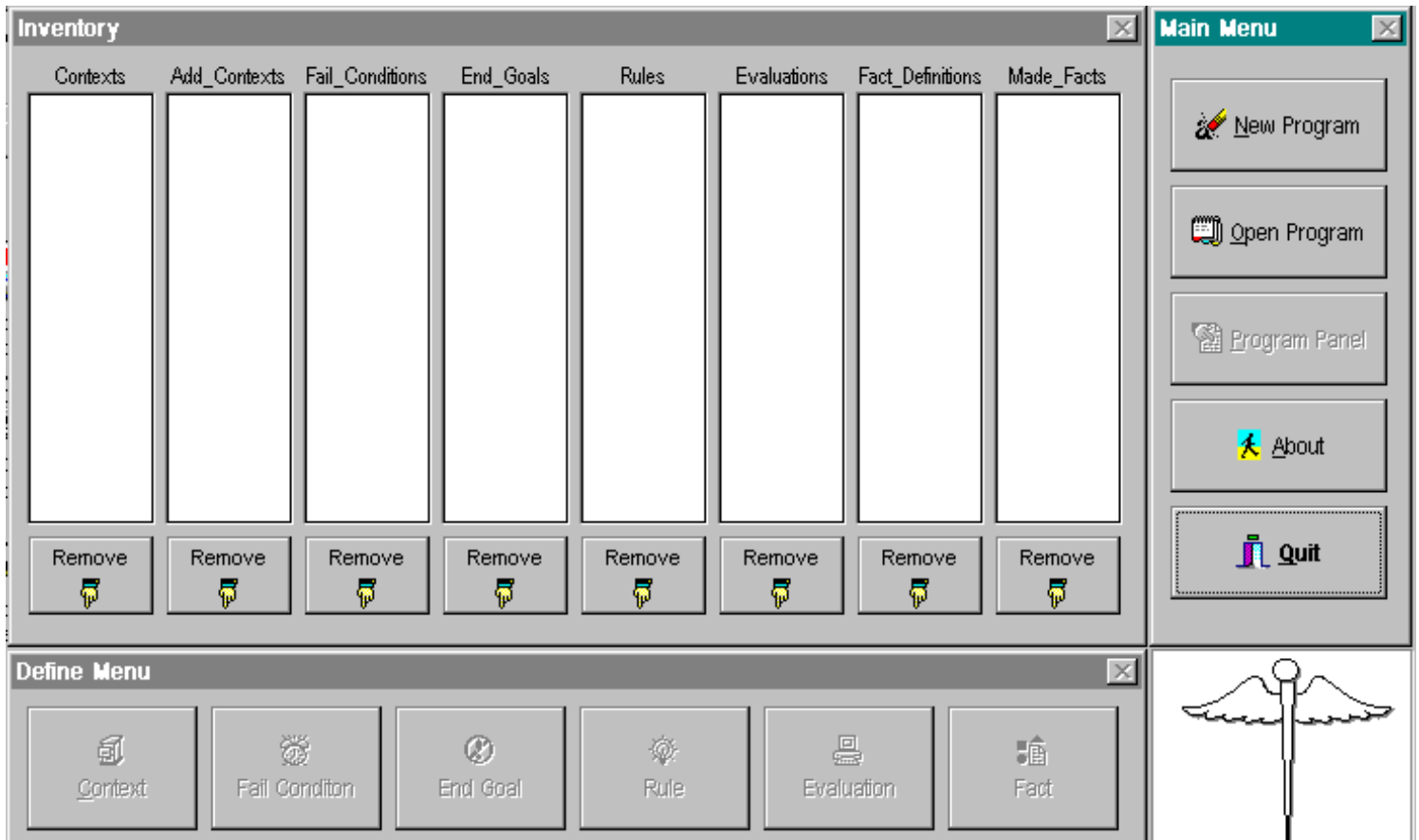
rule( r1,
      [all],

      global(1, counter H1) and
      vertical(1, pos V1) and
      not_true( 1, queen_position(1, h_pos H2, v_pos V2)
      and evaluate(1, conflicts(H1,V1,H2,V2)))
      -->
      modify(1,1, counter H1+1) and
      remove(2) and
      make(queen_position(1, h_pos H1, v_pos V1))).

end_goal(g1, [all], vertical(-1,pos Y)).

```

- *SSS Desktop Overview*



Notes :

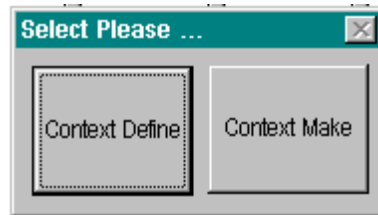
- a) New Program and Open Program is self explanatory.
- b) Inventory shows the structures which are present in the current program.

- *New Concepts In SSS Language*

- Contexts
- Fail Conditions
- Facts
- Evaluations
- Rules
- End Goals

- *Contexts*

- Shows data contexts.



- Define Contexts

e.g. `context(all)`.



- Make Contexts

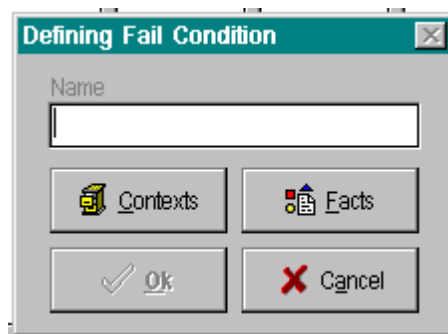
- The very same window is also used here.

e.g. `?- add_context(all)`.

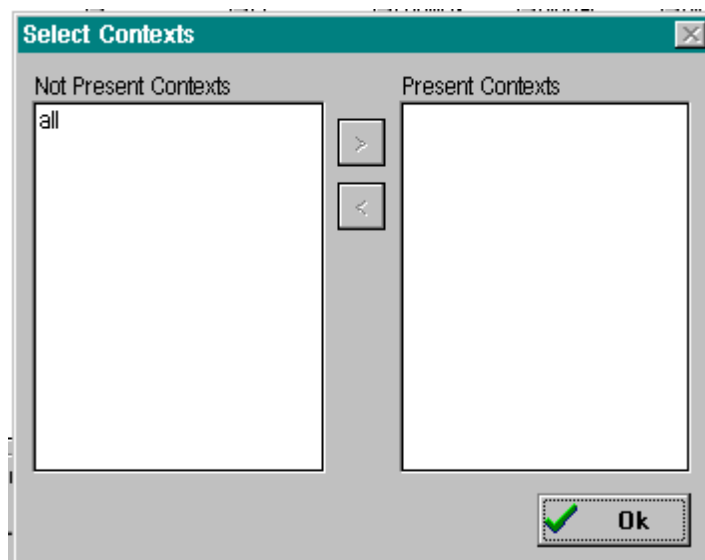
Notes :

- a) Contexts are like common data storage.
- b) Switching between contexts provides flexibility.
- c) Union contexts are possible.

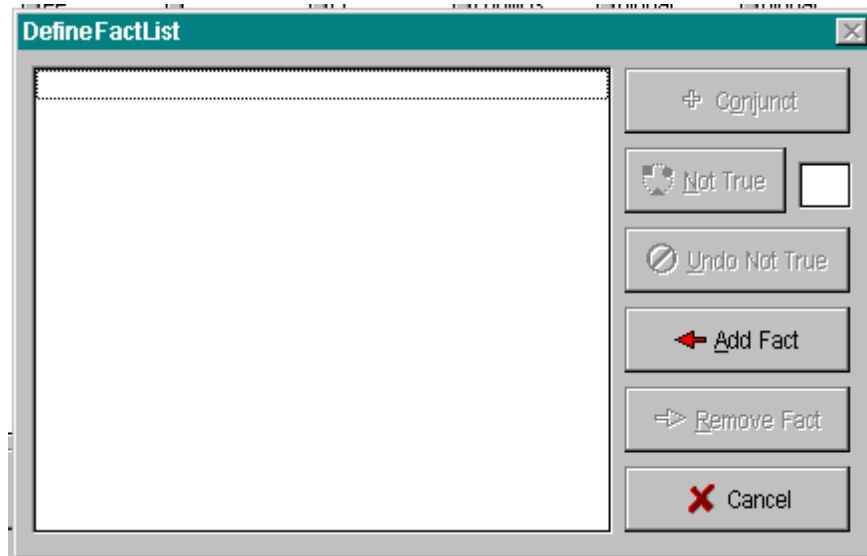
- *Fail Conditions*
 - Shows the conditions that will be never true in system
- Define Fail Conditions



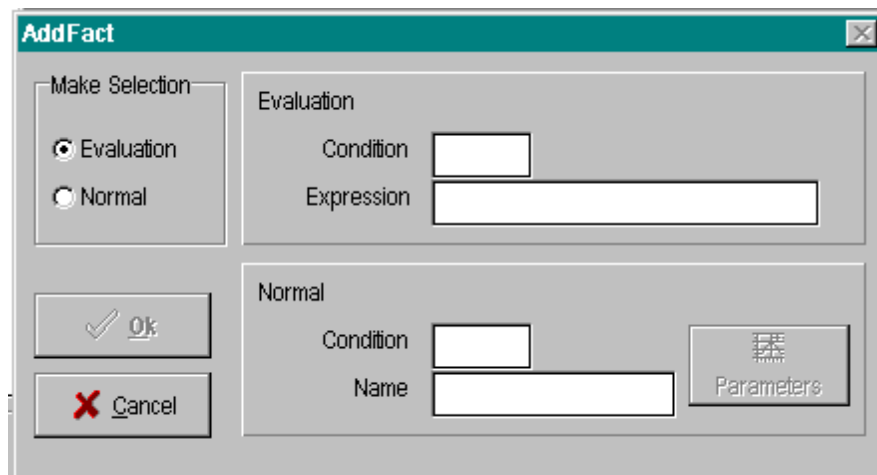
- Contexts In Fail Conditions
 - Define which contexts will be enabled here.



- Facts In Fail Conditions
 - These define what will be always false.



- Adding a new fact to the list



Notes :

- a) Fail Conditions are only calculated in the relevant contexts.

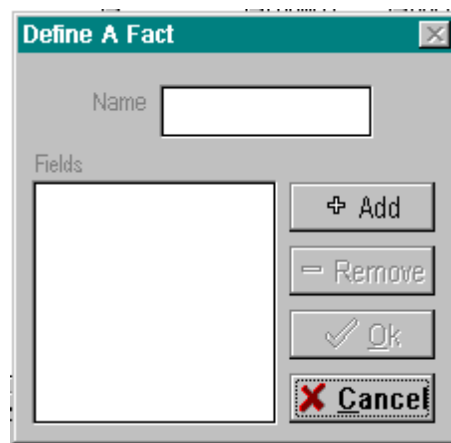
- b) The facts part may contain the facts defined or direct calculations.
- c) Conditional values define the probabilistic correctness.

e.g. `fail_condition(f1,[all],evaluate(1,M>0)).`

- *Facts*

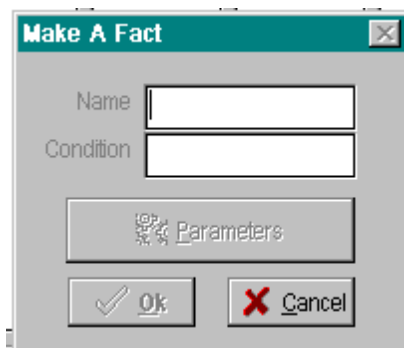
- Logical facts which are present in all ' Logic Languages '.
- Facts can be defined or can be enabled, just as contexts.

- Defining a Fact



- Defining does not mean that that fact is enabled
e.g. `?- literalize(queen_position(h_pos, v_pos)).`

- Making (Enabling) a Fact



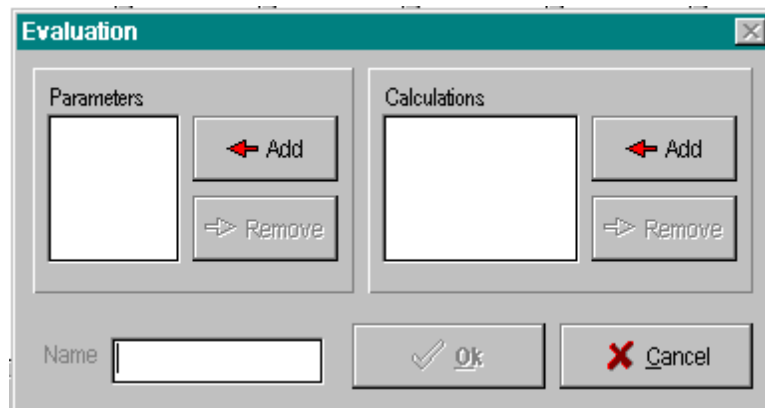
e.g. `?- make(vertical(1, pos 8)).`

Notes :

- a) Only after making, a fact is ready to use.
- b) To make a fact, it must be defined first.
- c) Parameters button checks whether this fact exists.

- *Evaluations*

- Evaluations are also facts.
- Their truth value is a composite value of all expressions.



Notes :

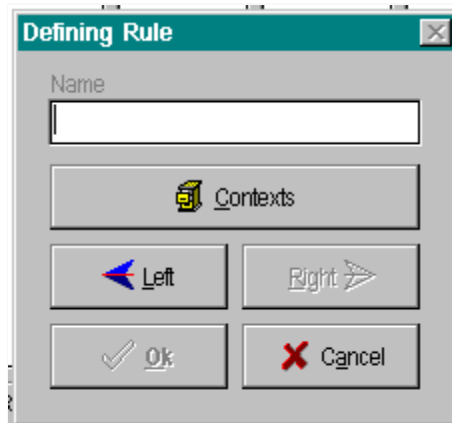
- a) The validity of the expression structure is checked by the parser.

e.g. `conflicts(H1,V1,H2,V2):- V1= V2,!.`

- *Rules*

- Rules are the most complex structures in SSS language.
- These correspond to Prolog-like rules, but have specialities.
- Right-Hand-Sides are particularly restricted.

- Defining a Rule



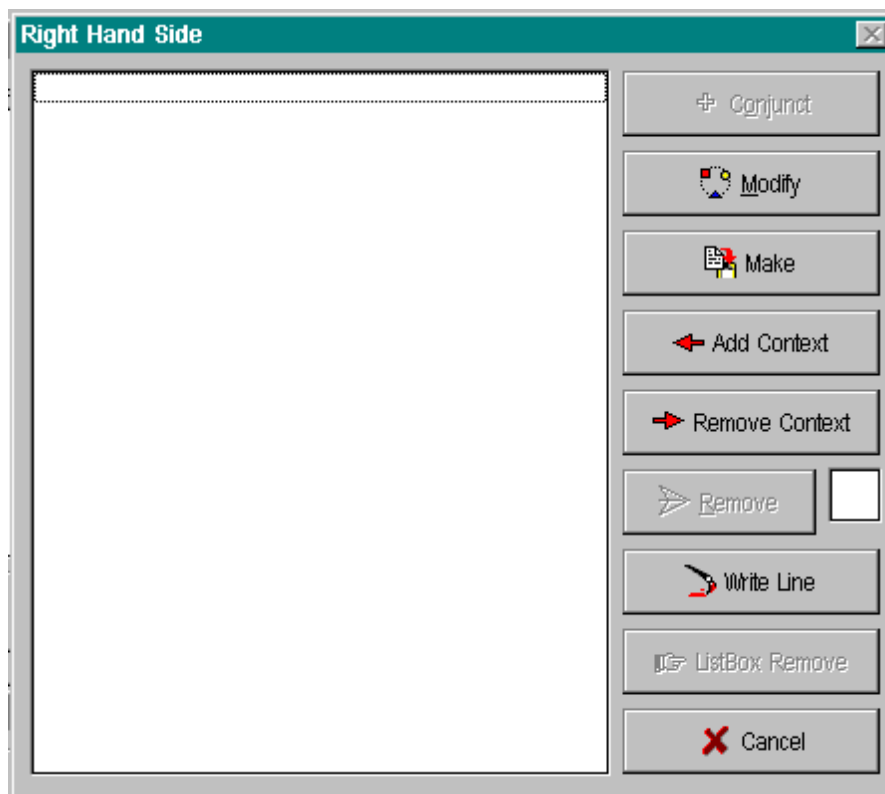
- Defining Context in a Rule

- This part is exactly the same as for other structures.
- The form presented is indeed the same.

- Left-Hand-Side

- This part is nothing more than defining a list of facts
- Here, the ' Fact List Define ' window is presented.

- Right-Hand-Side
 - This part is restricted to certain structures.
 - Modify, Add_Context, Remove etc.



Notes :

- a) Conjunct makes everything joined by 'and' s.
- b) Every rule can be separated into certain facts like Modify, Remove etc.
- c) Rules are the main structures managing the decisions taken.

```
e.g. rule( r1,[all],global(1, counter H1) and
          vertical(1, pos V1) and
          not_true( 1,
                    queen_position(1, h_pos H2, v_pos V2)
                    and evaluate(1, conflicts(H1,V1,H2,V2)))
-->
          modify(1,1, counter H1+1) and
          remove(2) and
          make(queen_position(1, h_pos H1, v_pos V1))).
```

- *End Goals*

- These are mainly the finish conditions indicating the end of program.
- They are defined exactly like the fail conditions.
- They usually contain one fact, but this is not a restriction.

e.g. `end_goal(g1, [all], vertical(-1, pos Y)).`

- *Program Panel*

- Main window to view the code generated so far
- The operations of this window consists of
 - Send code to parser to check type consistency and syntax
 - Send code to executor to execute the code
 - Save the code to a file named by the project plus extension 'sss'

