Kirsten McCane

Milestone 2

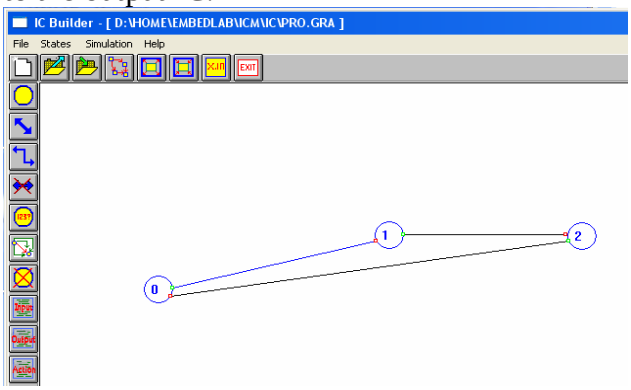
CS 2310

April 3, 2008

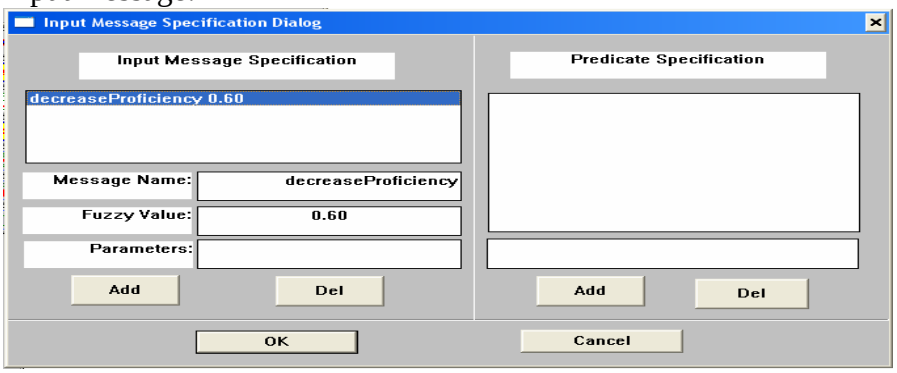
For milestone two I used the IC builder to create three ICs: One for each of the learning IC. The builder created .in files for each of these ICs. It input each of these ICs into the file ic.dat. I used the icc compiler to compile this data. The compiler produced the output files of actions.c ic_func2.c ic_func3.c app.h fuzzy.h db_def.h. I then used the VC++ to attempt to compile the appropiate .c and .h files. This resulted in errors that I was unable to debug. Another member in my group attempted the same sequence in a unix environment and also encountered errors at the same point. I will now pursue the simplest method which I believe is the unix environment to compile my project. I hope to work with my partner to resolve this error and continue in implementing the complete system. Errors were anticipated at this point and outside help may need to be consulted to get solid ideas for debugging the errors. Below are some of the screen shots from the process I took up to this milestone. Also included are the output files and action files I created.

These are screenshots of PRO.gra and its input and output messages:

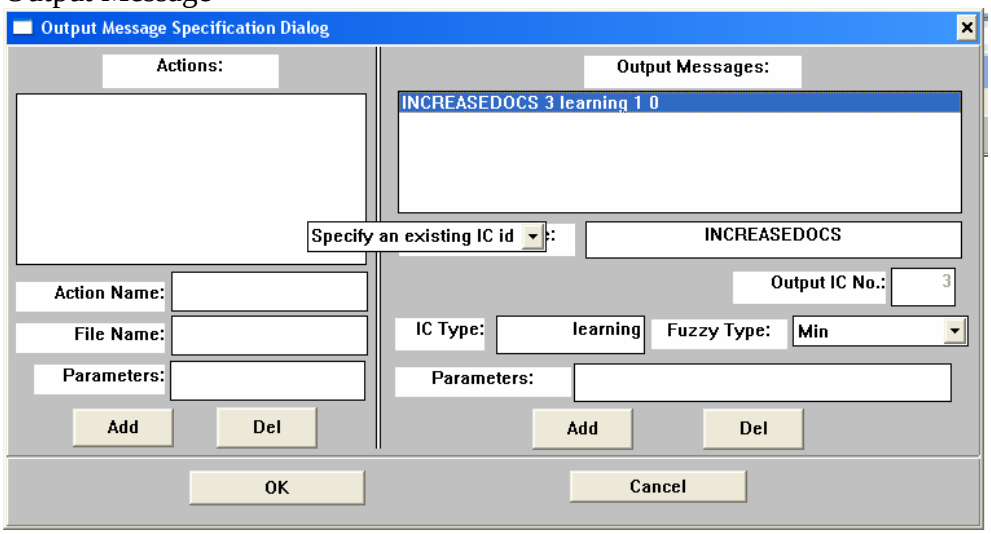
State Machine: I designed the stage machine that each time the IC receives input it goes to the next state. When it recieves three of these inputs it outputs a makeHarder message to the output IC.



Input message:



Output Message



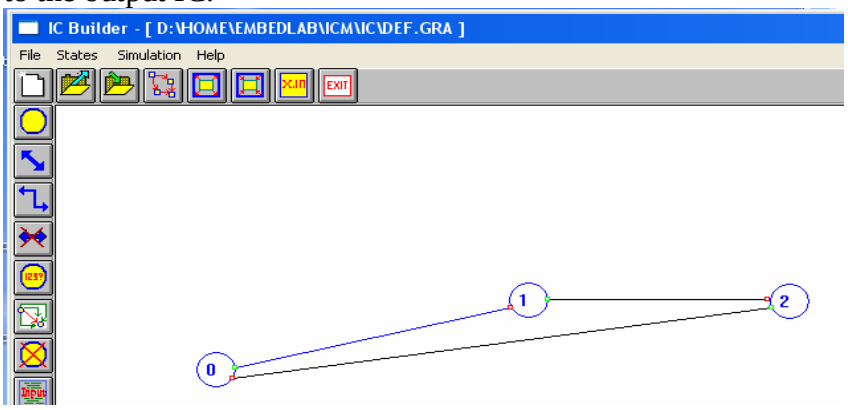
This is the PRO.in file created for the proficiency IC

```
0       // current state
1       // next state
1       // 1 input message(s)
0:0.60 // increaseProficiency
0       //          no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
0       // 0 action(s)
1       // current state
2       // next state
1       // 1 input message(s)
0:0.60 // increaseProficiency
0       //          no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
```

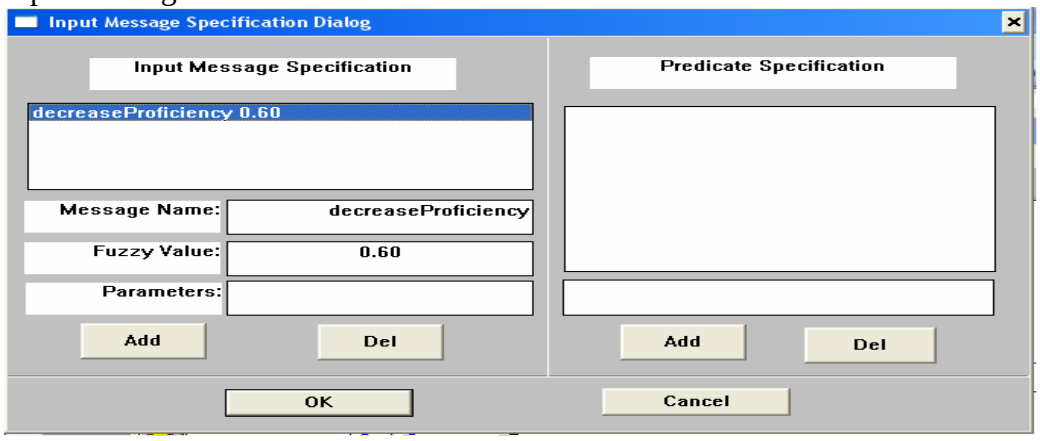
0 // 0 action(s)
2 // current state
0 // next state
1 // 1 input message(s)
0:0.60 // increaseProficiency
0 // no. predicate
1 // 1 output ic(s)
3,learning
1 // 1 output message(s)
1:0 // INCREASEDOCS
0 // 0 action(s)

Deficiency IC:

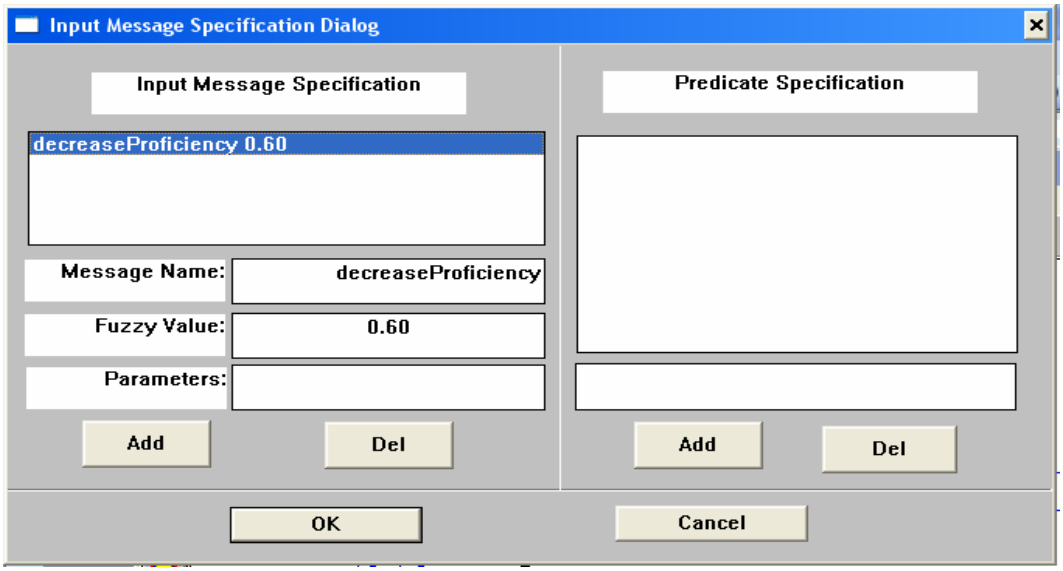
State Machine: I designed the stage machine that each time the IC receives input it goes to the next state. When it receives three of these inputs it outputs a makeEasier message to the output IC.



Input Message:



Output Message:

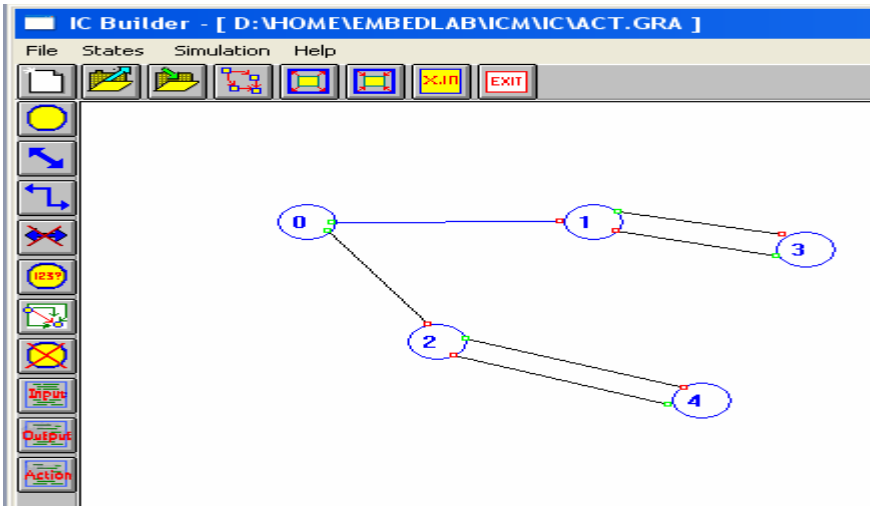


This is the DEF.in file created for the deficiency IC

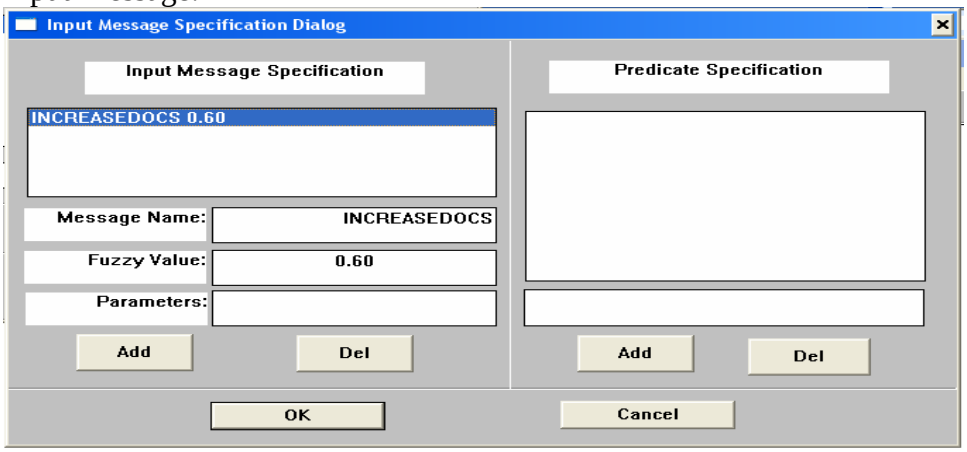
```
0         // current state
1         // next state
1         // 1 input message(s)
2:0.60 // decreaseProficiency
0       //          no. predicate
0         // 0 output ic(s)
0         // 0 output message(s)
0         // 0 action(s)
1         // current state
2         // next state
1         // 1 input message(s)
2:0.60 // decreaseProficiency
0       //          no. predicate
0         // 0 output ic(s)
0         // 0 output message(s)
0         // 0 action(s)
2         // current state
0         // next state
1         // 1 input message(s)
2:0.60 // decreaseProficiency
0       //          no. predicate
1         // 1 output ic(s)
3,learning
1         // 1 output message(s)
3:0 // DECREASEDOCS
     0         // 0 action(s)
```

Active IC:

State Machine: The state machine works by taking either increaseDocs or decreaseDocs input. Each time either of these inputs are received the action of harder.c or easier.c is enacted.



Input Message:



This is the ACT.in file created for the active IC

```
0       // current state
1       // next state
1       // 1 input message(s)
1:0.60 // INCREASEDOCS
0      //          no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
1       // 1 action(s)
0 // MAKE_HARDER harder.c
0       // current state
```

```
2       // next state
1       // 1 input message(s)
3:0.60 // DECREASEDOCS
0       //      no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
1       // 1 action(s)
1 // MAKE_EASIER easier.c
1       // current state
3       // next state
1       // 1 input message(s)
1:0.60 // INCREASEDOCS
0       //      no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
1       // 1 action(s)
0 // MAKE_HARDER harder.c
3       // current state
1       // next state
1       // 1 input message(s)
1:0.60 // INCREASEDOCS
0       //      no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
1       // 1 action(s)
0 // MAKE_HARDER harder.c
2       // current state
4       // next state
1       // 1 input message(s)
3:0.60 // DECREASEDOCS
0       //      no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
1       // 1 action(s)
1 // MAKE_EASIER easier.c
4       // current state
2       // next state
1       // 1 input message(s)
3:0.60 // DECREASEDOCS
0       //      no. predicate
0       // 0 output ic(s)
0       // 0 output message(s)
1       // 1 action(s)
1 // MAKE_EASIER easier.c
```

```
This is the IC.dat file used by the IC compiler
//////////////////////////////////////////////////////////////////////////
//
//    This is the sample of input to the IC code generator.
//
//    Format:
//
//      Each definition type header must be prefixed by '$'.
//      All definition lines follow their definition type header without prefixed
//      by any special character.
//      A definition types must be ended with '%'.
//      A comment line must begin with '//'.
//      A space line is allowed.
//
//      Supported definition types:
//
//      1. Header $MESSAGE   -->  message_name/message_id
//      2. Header $INCLUDE_FILE  -->  file_name
//      3. Header $ACTION/AUTO_GEN:YES|NO
//                  -->  action_name/action_id[/function_name[/file_name]]
//      4. Header $IC_ID     -->  name/id
//      5. Header $MUST_FUNC/AUTO_GEN:YES|NO
//                  -->  func_group_name[/file_name]
//      6. Header $THRESHOLD -->  ic_type/fuzzy_number
//
//    Output of IC code generator:
//
//       app.h, actions.c, ic_func2.c, ic_func3.c
//
//////////////////////////////////////////////////////////////////////////
// define input and output messages of IC
$MESSAGE
INCREASEPROFICIENCY/0
INCREASEDOCS/1
DECREASEPROFICIENCY/2
DECREASEDOCS/3
%

// add including files into app.h, if any
$INCLUDE_FILE
%

// define actions of IC
//
```

```
// AUTO_GEN:YES|NO
//      if YES, actions.c will be automatically generated by collecting
//      specified file_names; otherwise, the file_name
//      will be ignored and the user has to provide an actions.c by
//      himself.  If AUTO_GEN equals YES but file_name is
//      not specified, a default template of the corresponding function will
//      inserted into the actions.c.
//
//      If 'actions.c' exists, the old 'actions.c' will be moved to a backup
//      file 'actions.b*', for example, actions.b0, actions.b1....
//
$ACTION/AUTO_GEN:YES
MAKE_HARDER/0/make_harder/harder.c
MAKE_EASIER/1/make_easier/easier.c
%

// define IDs of IC
// Note: ic_id EXTERNAL has been defined as -1 in ic.h
$IC_ID
UNKNOWN/0
LEARNING/3
%

$MUST_FUNC/AUTO_GEN:YES
FILL_OUTPUT_MSG_GROUP/out_msg.c
PREDICATE_GROUP/predicat.c
INTERNAL_MM_GROUP/inter_mm.c
FILL_OUTPUT_IC_GROUP/out_ic.c
USER_DEFINE_FUNC_VAR_GROUP/func_var.c
%

// define thresholds for fuzzy computation
$THRESHOLD
DEFAULT/0.6
PRO/0.6
DEF/0.6
ACT/0.6
%
```

Finally, the errors I encountered looked as follows:

```
Linking...
util.obj : error LNK2019: unresolved external symbol _ix referenced in function _dump_ix
ic_functions.obj : error LNK2001: unresolved external symbol _ix
Ic_state.obj : error LNK2001: unresolved external symbol _ix
ic_func2.obj : error LNK2019: unresolved external symbol _add_vg_cell referenced in function _restore_mm
ic_functions.obj : error LNK2019: unresolved external symbol _locating referenced in function _get_next_receiver
ic_functions.obj : error LNK2019: unresolved external symbol _icb referenced in function _found_ic
ic_functions.obj : error LNK2019: unresolved external symbol _global_clock referenced in function _create_ic_insert_ix
Ic_state.obj : error LNK2001: unresolved external symbol _global_clock
ic_functions.obj : error LNK2019: unresolved external symbol _new_ic_sys_name referenced in function _create_ic_insert_ix
ic_functions.obj : error LNK2019: unresolved external symbol _do_predicate referenced in function _predicate_match
Icm.obj : error LNK2019: unresolved external symbol _initKB referenced in function _init
Icm.obj : error LNK2019: unresolved external symbol _initVG referenced in function _init
Icm.obj : error LNK2019: unresolved external symbol _IC_manager referenced in function _call_ic_manager
Icm.obj : error LNK2019: unresolved external symbol _show_config referenced in function _main
Icm.obj : error LNK2019: unresolved external symbol _update_config referenced in function _main
Icm.obj : error LNK2019: unresolved external symbol _read_config referenced in function _main
C:\Documents and Settings\clem\My Documents\Visual Studio 2005\Projects\ic3\Debug\ic3.exe : fatal error LNK1120: 13 unresolved extern
Build log was saved at "file://c:\Documents and Settings\clem\My Documents\Visual Studio 2005\Projects\ic3\ic3\Debug\BuildLog.htm"
```

Was able to locate the of where the external symbols should come from but all attempts at solving the error failed to date.