Hum[*] 8

Bill J. Ellis
February 28, 2005

A little story of a large bug

Contents

1 Introduction **2**

2 A Bug's Story **2**

2.1 Execute original NuSPADE . . . 2
2.2 Execution of library NuSPADE within SPADEase-Parser . . . 2
2.3 Initial analysis . . . 2
2.4 Deeper analysis . . . 2
2.5 Debug trace . . . 3
2.6 Trace the trace . . . 4
2.7 Got you! . . . 5
2.8 But why the different behaviour? . . . 6
2.9 Solution? . . . 7
2.10 And finally. 7

3 Executive summary **7**

[*]Hums are short notes intended for distribution between those involved with EPSRC grant GR/T12289/01. Hums describe ϵ-baked ideas, where $1 \geq \epsilon \geq 0$. (Hum refers to both the Praxis Humming bird and Winnie-the-Pooh's indirect approach to writing: "Poetry and Hums aren't things which you get, they're things which get you.").

1 Introduction

A significant task in building SPADEase is transforming NuSPADE from an executable program into an accessible library. In theory, this should be quite straight forward, sourcing the NuSPADE code into a suitable Prolog module. In practise, a difficult bug was encountered[1].

2 A Bug's Story

2.1 Execute original NuSPADE

Works fine. No problems. Perfect. Ok. Good.

2.2 Execution of library NuSPADE within SPADEase-Parser

```
% consulted u:/bill ellis/data/parser/nuspade/clamsettings.pl in module nuspade, 15 msec 976 bytes
! Existence error in nuspade:make_key/3
! procedure nuspade:make_key/3 does not exist
! goal:  nuspade:make_key(['A'],_85,_86)
```

Dang.

2.3 Initial analysis

Well, where is `make_key/3` then? Was it neglected to be included when sourcing NuSPADE as a module? A quick grep...

```
bash-2.05b$ grep -r 'make_key' *
dialect-support/libs.pl:bag_of(Vars, Template, Generator, Bag) :- make_key(Vars, Length, Key),
```

So, `make_key/3` is called but never declared. Perhaps it is a system predicate? And a system module load is missing? Consult manual...

```
# lst (order_resource/2 option): Enumeration Predicates
# macro expansion: Term and Goal Expansion
# main thread: Calling Prolog Asynchronously
# make_directory/1 (system): System Utilities
# make_index:make_library_index/1: The Prolog Library
# map_assoc/[2,3] (assoc): Assoc
# map_tree/3 (trees): Trees
```

Apparently not. Hmm.

2.4 Deeper analysis

How come this predicate exists in the working version of NuSPADE?

Trace statements are added to the original version of NuSPADE. This reveals that the code containing `make_key/3` is never reached. `make_key/3` is not the problem. This is a predicate which never existed in NuSPADE, and somehow this is only called in the library version of NuSPADE.

[1] "The difference between theory and practise is greater in practise than in theory" - Usenet signature of unknown origin

2.5 Debug trace

The trace of both the original and library NuSPADE is captured to try and figure out where these are different. The original version:

```
...
>       597     12 Exit: nuspade:an_with_measure('A',pair('A',[[0]]))
>       596     11 Exit: call(nuspade:an_with_measure('A',pair('A',[[0]])))
>       696     11 Call: asserta(nuspade:setof_stack(_181713,found(pair('A',[[0]]))))
>       696     11 Exit: asserta(nuspade:setof_stack(_181713,found(pair('A',[[0]]))))
>       594     10 Exit: nuspade:save_instances(_167541,an_with_measure('A',_167541))
>       697     10 Call: nuspade:list_instances(_167811)
>       698     11 Call: nuspade:list_instances([],_167811)
>       699     12 Call: retract(nuspade:setof_stack(_182409,_182410))
> ?     699     12 Exit: retract(nuspade:setof_stack(_182409,found(pair('A',[[0]]))))
>       700     12 Call: nuspade:'list instances'(found(pair('A',[[0]])),[],_167811)
>       701     13 Call: nuspade:list_instances([pair('A',[[0]])],_167811)
>       702     14 Call: retract(nuspade:setof_stack(_183371,_183372))
>       702     14 Exit: retract(nuspade:setof_stack(_183371,[]))
...
```

The library version:

```
...
< ?     597     12 Exit: nuspade:an_with_measure('A',pair('A',[[0]]))
< ?     596     11 Exit: call(nuspade:an_with_measure('A',pair('A',[[0]])))
<       696     11 Call: asserta(nuspade:setof_stack(_181827,found(pair('A',[[0]]))))
<       696     11 Exit: asserta(nuspade:setof_stack(_181827,found(pair('A',[[0]]))))
<       596     11 Redo: call(nuspade:an_with_measure('A',pair('A',[[0]])))
<       597     12 Redo: nuspade:an_with_measure('A',pair('A',[[0]]))
<       618     13 Redo: nuspade:measure(out,'A',[[0]])
<       619     14 Redo: nuspade:weakenings('A',['A'])
<       620     15 Redo: nuspade:set_of(_192407,weakest('A',_192407),['A'])
<       621     16 Redo: nuspade:bag_of(_192407,weakest('A',_192407),['A'])
<       622     17 Redo: nuspade:free_variables(weakest('A',_192407),_192407,[],[])
<       624     18 Redo: nuspade:free_variables_1(weakest('A',_192407),_192407,[],[])
<       625     19 Redo: nuspade:data_variables(weakest('A',_192407),_192407,[],[])
<       628     20 Redo: nuspade:data_variables(2,weakest('A',_192407),_192407,[],[])
<       638     21 Redo: nuspade:data_variables(1,weakest('A',_192407),_192407,[],[])
<       642     22 Redo: nuspade:data_variables('A',_192407,[],[])
<       697     23 Call: nuspade:term_is_free_of(_192407,'A')
...
```

Wha?

It seems the library version is backtracking while the original version does not? Is it perhaps the case that the original version is even more broken than the library version? The library version is backtracking, finding *more* results, then failing on the absence of `make_key/3`. Perhaps the library version is NuSPADE working correctly, only, somehow, `make_key/3` has been misplaced from the NuSPADE code base?

2.6 Trace the trace

What is the key predicate? Where do things go wrong from?

After much searching, and learning how to use SICStus debugger[2] the culprit was caught red handed. The original version:

```
nuspade:data_variables('A',A   ,[],[]).
        1      1 Call: nuspade:data_variables('A',_489,[],[])
        2      2 Call: nonvar('A')
        2      2 Exit: nonvar('A')
        3      2 Call: functor('A',_777,_778)
        3      2 Exit: functor('A','A',0)
        4      2 Call: nuspade:data_variables(0,'A',_489,[],[])
        5      3 Call: 0=:=0
        5      3 Exit: 0=:=0
        6      3 Call: []=[]
        6      3 Exit: []=[]
        4      2 Exit: nuspade:data_variables(0,'A',_489,[],[])
        1      1 Exit: nuspade:data_variables('A',_489,[],[])
true;
 ?
no
```

The library version:

```
nuspade:data_variables('A',A   ,[],[]).
        1      1 Call: nuspade:data_variables('A',_522,[],[])
        2      2 Call: nonvar('A')
        2      2 Exit: nonvar('A')
        3      2 Call: functor('A',_810,_811)
        3      2 Exit: functor('A','A',0)
        4      2 Call: nuspade:data_variables(0,'A',_522,[],[])
        5      3 Call: 0=:=0
        5      3 Exit: 0=:=0
        6      3 Call: []=[]
        6      3 Exit: []=[]
        4      2 Exit: nuspade:data_variables(0,'A',_522,[],[])
?       1      1 Exit: nuspade:data_variables('A',_522,[],[])
true;
 ?
        1      1 Redo: nuspade:data_variables('A',_522,[],[])
        7      2 Call: nuspade:term_is_free_of(_522,'A')
        8      3 Call: var(_522)
        8      3 Exit: var(_522)
        9      3 Call: _522\=='A'
        9      3 Exit: _522\=='A'
        7      2 Exit: nuspade:term_is_free_of(_522,'A')
       10      2 Call: nuspade:list_is_free_of([],'A')
       10      2 Exit: nuspade:list_is_free_of([],'A')
       11      2 Call: []=['A']
       11      2 Fail: []=['A']
       12      2 Call: []=[]
       12      2 Exit: []=[]
        1      1 Exit: nuspade:data_variables('A',_522,[],[])
true;
 ?
no
```

The library version backtracks where the original version does not. This leads to an extra result in the calling predicate, which causes a much higher predicate to (eventually) follow the branch of code that leads to the absent `make_key/3`. `data_variables/4` is the problem.

[2] Read: Learning how not to use the SICStus debugger, and amending actions accordingly.

2.7 Got you!

The code for `data_variables/4` takes the following (horrendous) form.

```
data_variables(Term, Bound, Vars0, Vars) :-
        (   nonvar(Term) ->
            functor(Term, _, N),
            data_variables(N, Term, Bound, Vars0, Vars)
        ;   term_is_free_of(Bound, Term),
            list_is_free_of(Vars0, Term)
        ->  Vars = [Term|Vars0]
        ;   Vars = Vars0
        ).

data_variables(N, Term, Bound) -->
    (   { N =:= 0 } -> []
    ;   { arg(N, Term, Arg), M is N-1 },
        data_variables(Arg, Bound),
        data_variables(M, Term, Bound)
    ).
```

Take a moment to appreciate the shear horror of this Prolog. The latter predicate is declared in Definite Clausal Grammar (DCG) form (and is called directly in its expanded form via `data_variables/5`). This short segment of code contains no less than 3 `->` (if-then) and 3 `;` (or) constructs[3]. Throw in Prolog recursion. And, finally, brilliantly, there is not a single explicit Prolog cut. The backtracking semantics for this is quite impenetrable. Fantastic!

[3]Some Prolog commentators advocate *never* using these constructs - and most would argue against their nesting.

2.8 But why the different behaviour?

Ah. My bad. The original version:

```
current_op(A,B,'->').
A = 1100,
B = xfy ?
```

The library version:

```
current_op(A,B,'->').
A = 830,
B = xfy
```

It turns out that the operator `->` is overloaded to denote implication, and has its precedence redeclared in SPADEase. This has, until now, not been a problem. However, `data_variables/4` is rather dependent on this precedence for its semantics. Note the brackets in the following. The original version:

```
listing([nuspade:data_variables]).
nuspade:data_variables(A, B, C, D) :-
        (   nonvar(A) ->
            functor(A, _, E),
            nuspade:data_variables(E, A, B, C, D)
        ;   nuspade:term_is_free_of(B, A),
            nuspade:list_is_free_of(C, A) ->
            D=[A|C]
        ;   D=C
        ).

nuspade:data_variables(A, B, C, D, E) :-
        (   A=:=0 ->
            E=D
        ;   arg(A, B, F),
            G is A-1,
            nuspade:data_variables(F, C, D, H),
            nuspade:data_variables(G, B, C, H, E)
        ).
```

The library version:

```
listing([nuspade:data_variables]).
nuspade:data_variables(A, B, C, D) :-
        (   (   nonvar(A) ->
                functor(A, _, E)
            ),
            nuspade:data_variables(E, A, B, C, D)
        ;   nuspade:term_is_free_of(B, A),
            (   nuspade:list_is_free_of(C, A) ->
                D=[A|C]
            )
        ;   D=C
        ).

nuspade:data_variables(A, B, C, D, E) :-
        (   A=:=0 ->
            E=D
        ;   arg(A, B, F),
            G is A-1,
            nuspade:data_variables(F, C, D, H),
            nuspade:data_variables(G, B, C, H, E)
        ).
```

The different precedence of `->` leads to completely different semantics for `data_variables/4`, which in turn leads to additional backtracking, whose extra results lead to the missing predicate `make_key/3` being called.

2.9 Solution?

Temporally change the precedence of `->` as the NuSPADE code is sourced. If `->` should ever find its way into SPADEase code it should be surrounded by brackets to force its interpretation[4].

2.10 And finally...

Here is something I wrote a long, long time ago:

```
...
%As changing the SPADE-PC precedence would mean substantial
%rule changes, its precedence must remain. Thus where used
%as a conditional operator brackets may be required.
...
:-op(830        ,xfy  ,->                  ).
```

AAUGH!

3 Executive summary

Reports on the discovery and eventual detection of a tricky bug in the emerging SPADEase-Parser system[5].

[4] Or, preferably, the code should be refactored to eliminate the nasty `->` construct.

[5] In short: Shooting yourself in the foot, forgetting about it for a few years, experiencing a little bit of difficulty walking, decide to run a few tests, reach an inevitable conclusion, and proclaim 'ouch'.