On Solving Hexadoku and Debugging Recursive Programs with Message Digests of the Data Stack

François Laagel, IEEE

September 12, 2023

Introduction to Hexadoku

E	2		A	0			B		F		C	6	4	9	
	C		F	1	5	8					0		B	A	7
	1					4							8		2
8	3	4				C	D		7	B	E		0	1	
		F	8								3			2	9
7		2	3			1				9	B	D			A
	6				D		8		A	E		5			
	1			5			6		8		D				
	7			A			3		E		6				
	D				6		2		B	3		8			
3		9	6			5				7	F	1			0
		0	5								9			7	6
0	9	E				2	A		3	F	5		C	6	
		7				6							5		3
	8		C	F	7	3					1		D	0	B
5	F		4	8			0		6		2	A	7	E	

The speculate Engine

```
 1  : speculate ( -- success-flag )
 2    rl+                          \ Increment recursion level
 3    get-unresolved               \ Look for an unresolved spot
 4    DUP 0= IF INVERT EXIT THEN   \ Problem solved
 5    DUP @                        \ S: saddr\backslashsval
 6    \ The list of set bits in TOS indicate the possibilities
 7    \ for the selected spot. Explore these alternatives.
 8    16 0 DO
 9      DUP I 2^n AND IF
10        OVER TRUE SWAP tstk-push \ Insert transaction boundary
11        OVER I 2^n SWAP
12          +ul |visual -ul !      \ Un-logged update-spot
13        infer IF                 \ No inconsistencies detected
14          RECURSE IF             \ Stop on the 1st solution found
15            2DROP UNLOOP TRUE EXIT
16          THEN
17        THEN
18        \ Backtrack up to the last transaction boundary.
19        BEGIN tstk-pop UNTIL
20        nbt 1+!              \ Increment the number of backtracks
21      THEN
22    LOOP
23    2DROP FALSE                  \ Dead end reached
24    rl- ;                        \ Decrement recursion level
```

Traditional Debugging Strategies

- Enforcing assertions
  Defensive programming with `ABORT"`
- A good logging subsystem
  Should be a part of the original code
  If not built in, first thing to address
- Documentation improvement/code reviews
  Expensive and time consuming
- Ad'hoc code instrumentation

Initial Logging Information

```
 1 >speculate at rl 15
 2                [ 4 , 1 ] <- 5          Cell owned at rl 15
 3                >infer
 4                <infer
 5 >speculate at rl 16
 6                 [ 4 , 12 ] <- 7
 7                 >infer
 8                 <infer
 9 >speculate at rl 17
10                  [ 4 , 4 ] <- C
11                  >infer
12                  [ 4 , 7 ] <- E
13                  [ 5 , 7 ] <- 4
14                  [ 10 , 7 ] <- 4        Vertical constraint violation
15                  [ 10 , 5 ] <- 8
16                  <infer
17                  >backtrack
18                  [ 10 , 5 ] <-
19                  [ 10 , 7 ] <-
20                  [ 5 , 7 ] <-
21                  [ 4 , 7 ] <-
22                  [ 4 , 4 ] <-
23                  <backtrack
24 <speculate at rl 17
25                >backtrack
26                [ 4 , 12 ] <-
27                <backtrack
28                [ 4 , 1 ] <- E          Cell contents altered at rl 16!
```

Stack Digests

- a convenient synthetic overview of the content of the data stack
- any message digest generation algorithm would do, including CRCs
- SHA1 selected because it is well specified and standardized
- a data stack integrity checking tool

`SDIGEST` (i*x u – i*x) Prints a cryptographic digest of the contents of the data stack, omitting the topmost *u* cells. The algorithm used for producing this digest is implementation defined.

Using Stack Digests to Debug the Solver

```
>speculate at rl 15
               [ 4 , 1 ] <- 5
               >infer 9BFF706E:A6677DE5:85F22898:8307CA39:48100DAF
               <infer 9BFF706E:A6677DE5:85F22898:8307CA39:48100DAF
>speculate at rl 16
                [ 4 , 12 ] <- 7
                >infer F4C4FD26:1FA81AE7:E5C3FAFC:56CFBE75:38EA8F21
                <infer F4C4FD26:1FA81AE7:E5C3FAFC:56CFBE75:38EA8F21
>speculate at rl 17
                 [ 4 , 4 ] <- C
                 >infer 9924F213:200BD2E2:27752C46:9C617B1E:486832CB
                 [ 4 , 7 ] <- E
                 [ 5 , 7 ] <- 4
                 [ 10 , 7 ] <- 4
                 [ 10 , 5 ] <- 8
                 <infer F4C4FD26:1FA81AE7:E5C3FAFC:56CFBE75:38EA8F21
```

Converging on the Problem's Root Cause

```
 1  : reduceall ( -- failure-flag )
 2    reduce4x4 IF                \ Constraint violated
 3      TRUE EXIT
 4    THEN
 5
 6    16 0 DO
 7      I get-horiz-mask IF       \ Constraint violated
 8        UNLOOP TRUE EXIT
 9      THEN
10      ( S: new-possibly-zero-mask ) I SWAP set-horiz-mask IF
11        UNLOOP TRUE EXIT
12      THEN
13
14      I get-vert-mask IF        \ Constraint violated
15        UNLOOP TRUE EXIT
16      THEN
17      ( S: new-possibly-zero-mask ) I SWAP set-vert-mask IF
18        UNLOOP TRUE EXIT
19      THEN
20
21    LOOP
22    FALSE ;
```

Conclusion

- Public domain code available
- Stack digests as useful debugging tool
- Pros and cons of recursion
- Performance of the solver

Q&A