

## Implementing a DSL with Stratego

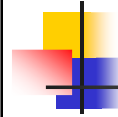


Leonard G. C. Hamey and Shirley N. Goldrei  
Department of Computing  
Macquarie University  
Sydney, Australia

10:25-10:45am



## Background



- Re-implement DSL
  - Original implementation:
    - 20 years ago
    - Domain expert (first author)
- Re-implemented today using Stratego/XT
  - Same domain expert
  - Language essentially the same
    - The compiler design goals are a little different
  - Diary of development experience
    - Basis of LDTA paper



## Domain

- Low Level (pixel level) Computer Vision
  - Edge Detection
  - Detection of corners, ridges or blobs
  - Used to help identify objects or track moving objects
- Algorithms which compute new pixel values based on neighbouring values
  - Kernel computation including Convolution



## Example: Sobel operator for Edge detection



## Example: Sobel operator for Edge detection



[[plrg]]  
macquarie

## Apply Language

- Subset of Ada
  - Arithmetic and boolean expressions
  - control flow structures
  - primitive data types: byte, real and integer
  - multidimensional array types with index ranges
  - procedures (no functions)

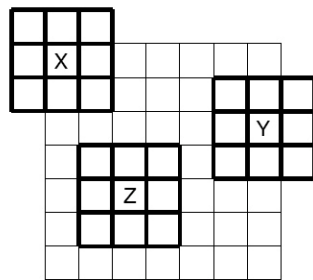
[[plrg]]  
macquarie

## Apply Language: features

- Abstract Data Type **window**
  - represents a rectangular region of the image on which the kernel operation will be performed
- Procedure special formal parameter declaration:
  - window of *Type*
  - window (*Range, Range*) of *Type* border *expr*



## Kernel Operations



- 3x3 kernel
- three example window locations
  - x top left corner
  - y right edge
  - z clear of all borders





## Kernel Operations

- In general, 9 regions
- E has no border considerations
- Others have differing border considerations
- Small image considerations

A	B	C
D	E	F
G	H	I

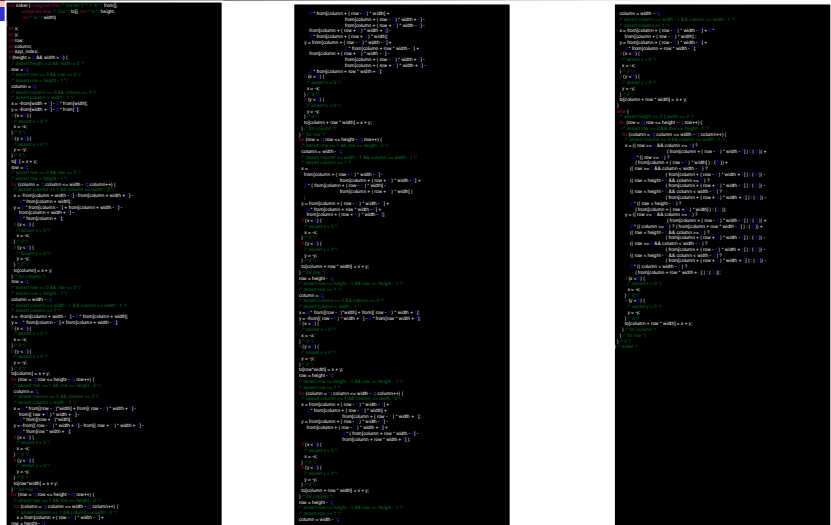


## Sobel in *Apply*


```
procedure sobel(from: in window (-1..1, -1..1)
               of byte border 0,
               to: out window of byte)
is
  x, y : integer;
begin
  x := from(-1,-1) + 2*from(-1, 0) + from(-1,1) -
        from( 1,-1) - 2*from( 1, 0) - from( 1,1);
  y := from(-1,-1) + 2*from( 0,-1) + from(1,-1) -
        from(-1, 1) - 2*from( 0, 1) - from(1, 1);
  if x < 0 then x := -x; end if;
  if y < 0 then y := -y; end if;
  x := x + y;
  if x > 255 then x := 255; end if;
  to := x;
end sobel;
```



# Sobel: Generated C Code




The image displays three panels of C code generated for the Sobel operator. The code is presented in a dark-themed editor with syntax highlighting. The first panel shows the most basic implementation with clear loops and variable declarations. The second panel shows an optimization where the inner loop is unrolled, and some variables are moved out of the loop. The third panel shows a more aggressive optimization, likely using SIMD or vectorization, with code that processes multiple pixels at once using array indexing and pointer arithmetic.



# Compiler design goals

- Easily retarget different C APIs
- Generate readable C code
  - Aid Verification
  - Build confidence to allow integration in larger applications
- Optimisation (with domain knowledge)
  - Simplification
  - Execution performance



## Implementing Apply in Stratego/XT

- Translate Apply -> AST -> C
- Extended concrete syntax:
  - AST transformations difficult to write
  - @ indicates extension keyword/syntax
  - Abstract Apply main loop: @apply
  - Bridge Ada -> C gap: e.g. @cfor => C for loop
  - Analysis and optimisation: @assert, @known



## AST example

```
FixLoop :
  ApplyLoop(stmts) ->
    For(
      Var("row"),
      Int("0"),
      Sub(Var("height"), Int(1)),
      For (
        Var("column"),
        Int("0"),
        Sub(Var("width"), Int(1)),
        stmts
      )
    )
```

Early version (day 11):  
2 for loops only





## Example

```

FixLoop2Index :
[[ @apply ~looptype window (i1..i2,j1..j2) loop ~s end loop; ]| ->
[[ app_index := 0;
   for row in 0..height-1 loop
     @cfor column @:= 0; column <= width-1; loop
       assert column >= 0 and column <= width - 1;
       assert row >= 0 and row <= height-1;
       ~s
       if column = -j1-1 and row >= -i1 and row < height - i2 then
         column := column + width - j2 + j1 + 1;
         app_index := app_index + width - j2 + j1 + 1;
       else
         column := column + 1;
         app_index := app_index + 1;
       end if;
     end loop;
   end loop;
   app_index := -i1 * width - j1;
   for row in -i1..height-i2-1 loop
     for column in -j1..width-j2-1 loop
       ~s
       app_index := app_index + 1;
     end loop;
     app_index := app_index -i1 + i2;
   end loop;
]] where(<debug> ["extreme window dimensions: ", i1, i2, j1 ,j2])

```



## Unexpected benefits

- Experimentation with language features

```

if (@known(row < 255)) then x:=row;
    else x:=row % 255 end if;

```

- loop for row 1..100 do loop

Generates:


```

@assert row >= 1
@assert row <= 100

```










## Implementation Comparison

- Original
  - Lex
  - Yacc
  - C
    - verbose code for tree traversal, matching and transformation
    - code output
- Stratego/XT
  - SDF
  - Stratego
    - Compact code more powerful
  - Pretty Printer



## Implementation Comparison

- Original
  - Explicit coding
  - No concrete syntax
  - Change very complex
- Stratego/XT
  - Simple DS syntax for matching and transformation
  - Concrete syntax
  - Reduce complexity by composing small transformations





## Relative speed up compared to reasonable hand-written code

	Core-gcc	PC-gcc	SPARC-gcc	Core-MSVC	PC-MSVC
Hand-written	5.232	3.608	4.668	5.190	4.070
Old compiler	4.429	3.438	6.366	4.450	4.600
Speedup	15%	5%	-36%	14%	-13%
New Compiler	3.283	2.462	4.596	3.940	3.970
Speedup	37%	32%	2%	24%	2%




## Quantitative comparison of development effort

- Evolution of Apply programming model and language over 6 years
  - Started as an API
  - Platform specific language
  - Ada based platform independent language
  - C tree matching and transformation added
- Re-implementation took 5 months (based on total logged elapsed time)





## Qualitative Evaluation


- The project was a success!
- Short development time
- Met design goals
- Old C code was impossible to work with (even by the original implementer!)
  - language development stalled for 16 years



## Conclusion

- Comparing implementations of a single DSL
- Implemented by a Domain Expert
  - “Excited by the fact that I could just say I want to transform this bit of code to this other bit of code”
- Implementation of a non-trivial non-embedded DSL using transformation





---

Questions?

