Aldor: Implementation I

Stephen M. Watt

University of Western Ontario

Edinburgh, September 28 2000

Aldor — Implementation

- Optimizing compiler
- Interpreted interactive environment for the same language
- Generates
  - Stand-alone executable programs
  - Object libraries in native OS formats
  - Portable byte code libraries
  - C or Lisp source

Compiler Objects

- Abstract syntax tree (AbSyn)
- Symbol meaning (Syme)
- Semantic form (Sefo)
- Type form (TForm)
- First order abstract machine code (FOAM)

Compiler Phases

1. Includer (text → text)
2. System command handler (text → text)
3. Scanner (text → token stream)
4. Linearizer (token stream → token stream)
5. Parser (token stream → parse tree)
6. Macro expander (parse tree → parse tree)
7. Abstract syntax tree normalizer (parse tree → AbSyn)
8. Abstract syntax tree checker (AbSyn → AbSyn)
9. Scope binder (AbSyn → AbSyn)
10. Type inference (AbSyn → Sefo)
11. Intermediate code generator (Sefo → FOAM)
12. Foam optimizer (FOAM → FOAM)
13. Concrete code generator (FOAM → C, Lisp)
14. Machine code generation (C, Lisp → native code)

The Aldor Compiler as a Program

- 327 files (200K loc) compiler proper
- 127 files (30K loc) in basic data structure library
- 285 files (65K loc) basic mathematics library
- 523 files (21K loc) in test suite
- 293 files (42K loc) of support code + tools

Compiler Command Line Options

```
Usage:     axiomxl [options] files ...

           axiomxl -help       Basic help.
           axiomxl -hfiles     Input file types.
           axiomxl -hoptions   Summary of options.
           axiomxl -hhelp      Help on help.
           axiomxl -hall       All help.

File types accepted:
  .as      Aldor source

  .ai      Included Aldor source
  .ax      Parsed Aldor source
  .fm      Foam source
  .ao      Machine-independent object file (interface, code, ...)
  .al      Archive of machine-independent object files
  .o       Object file
  .a       Archive of object files

           Files with no type extension are taken as Aldor source.
```

```
Option summary:
 -V        Run verbosely, giving compilation information.
 -D <id>   Add global assertion as `#assert <id>'.
 -U <id>   Remove global assertion as `#unassert <id>'.
 -A <fn>   Read command line options from response file <fn>.
 -K <n>    Compile only the first <n> files (0-9).
 --        Treat remaining arguments as input files.
 -H ...    Help.
 -B <dir>  Use <dir> as the base directory for axiomxl system files.
 -I <dir>  Search <dir> for additional include files.
 -Y <dir>  Search <dir> for additional libraries.
 -R <dir>  Put the resulting files in directory <dir>.
 -L <name> Use the given library.
 -F ...    Indicate which output files are to be generated.
 -E <fn>   Specify the main entry point.
 -G ...    Run the program.
 -O        Standard optimizations.
 -Q ...    Select code optimizations.
 -Z ...    Debug and profiling options.
 -C ...    Control C generation.
 -P ...    Control C++ generation.
 -S ...    Control Lisp generation.
 -M ...    Control compiler messages.
 -N ...    Configuration file options.
 -W ...    Developer options.
```

```
Help options:
  -H elp   Brief, general help.
  -H all   Help about ... ALL options.
  -H files            ... input file types.
  -H options          ... summary of options.
  -H info             ... product information.
  -H {A|args}         ... argument gathering options.
  -H {H|help}         ... help options.
  -H dir              ... directory and library options.
  -H {F|fout}         ... output file options.
  -H {G|go}           ... execution options.
  -H {O|Q|optimize}   ... optimization options.
  -H {Z|debug}        ... debugging options.
  -H C                ... C code generation options.
  -H {S|lisp}         ... Lisp code generation options.
  -H {M|message}      ... message options.
  -H {W|dev}          ... developer options.
```

```
Argument gathering options:
  -A <fn>  Read command line options from response file <fn>.

  -K <n>   Compile only the first <n> files (0-9).  The remaining
           arguments are given to the Aldor program, if run with `-G'.

  --       Treat remaining arguments as input files, even if they begin with `-'.
           If the environment variable `AXIOMXLARGS' is defined, its
           contents are handled as options before the command line.


Directory and library options:
  -I <dir> Search <dir> for additional include files.
           The environment variable `INCPATH' is tried for defaults.
  -Y <dir> Search <dir> for additional libraries.
           The environment variable `LIBPATH' is tried for defaults.
  -R <dir> Put the resulting files in directory <dir>.
           The default is the current directory.
  -B <dir> Use <dir> as the base directory for axiomxl system files.
           Without `-B', the environment variable `AXIOMXLROOT' is tried.

  -L <fn>  Use the library given by file name <fn>.
           An alphanumeric <fn> is a short form for `lib<fn>.a'.
  -L <id>=<fn>    Same as `-L <fn>' but the source name <id> is
                  associated with the library.
```

```
File options:
 -F <ft>[=<fn>]  Indicate which output files are to be generated.
          The parameter <ft> is one of:
            ai       (*) Source after inclusion      (from .as)
            ax       (*) Macro-expanded parse tree   (from .as)
            asy      (*) Symbol information
            ao       (*) Machine-independent object
            fm       (*) Foam code
            lsp      (*) Lisp code
            c        (*) C code
            c++      (2) C++ code and Aldor stubs
            o        (*) Object file
            x        (1) Executable file
            axlmain (1) Generate ‘axlmain.c’ with function ‘main’.

          If no ‘-F’ or ‘-G’ option is given, then ‘-Fao’ is assumed.
          Cases marked ‘(*)’, above, cause one file of the given
          type to be generated for each input file.
          Cases marked ‘(1)’ cause only one file to be generated for
          the entire compilation.
          Cases marked ‘(2)’ cause two files of the given type
          to be generated for each input file.
          If the parameter <fn> is given, it is used as the name of
          the corresponding output file.

          The compiler will not overwrite a C or Lisp file it did not generate.
```

```
Execution options:
  -G run    Run the program, as compiled to machine code.
            The executable file is removed when finished unless
            the ‘-Fx’ option is present.
  -G interp Interpret the program, as translated to intermediate Foam code.
  -G loop   Run interactively, interpreting each expression as it is
            given.  With this option, the file ‘aldorinit.as’ is used for
            initialization.

  -E <fn>   Use the input file ‘<fn>.*’ as the main entry point.
            The default is the first file.
            ‘-e’ is useful only with ‘-Fx’, ‘-Grun’ or ‘-Ginterp’.



Optimization options:
  -O              Optimize.  This is equivalent to ‘-Q2’.

  -Q <n>          Select a level of code optimization. (default ‘-Q1’)

  -Q <opt>        Turn on optimization <opt>.
  -Q no-<opt>     Turn off optimization <opt>.
```

Optimization options (Continued):

		Q0	Q1	Q2	Q3	Q4
-Q all	All supported optimizations.				X	X
-Q inline	Allow open coding of functions.			X	X	X
-Q inline-all	Allow open coding of any functions.				X	X
-Q inline-limit=<n>	Set maximum inline code size to <n>.	1	1	5	6	8
-Q cfold	Evaluate constants at compile time.		X	X	X	X
-Q ffold	Evaluate floating point constants.					X
-Q hfold	Evaluate type code computations.		X	X	X	X
-Q peep	Local ‘‘peep-hole’’ optimizations.		X	X	X	X
-Q deadvar	Eliminate unused variables and values.		X	X	X	X
-Q dassign	Eliminate assignments to dead variables.			X	X	X
-Q emerge	Eliminate lexical levels and records.			X	X	X
-Q emerge-rr	Eliminate raw records.			X	X	X
-Q cprop	Copy propagation.			X	X	X
-Q cse	Common sub-expression elimination.			X	X	X
-Q flow	Simplify computed tests and jumps.			X	X	X
-Q cast	Reduce the number of casts.			X	X	X
-Q cc	Use the C compiler’s optimizer.			X	X	X
-Q del-assert	Do not check ’assert’ statements.			X	X	X
-Q cc-fnonstd	Possibly faster but not IEEE compliant					X

Combinations can be used, e.g. ‘-Q3 -Qno-ffold’.
The option ‘-Qcc’ may exclude ‘-Z’ on some platforms.

```
Debug options:
  -Z db             Generate debugging information in object files.
  -Z prof           Generate profiling code in object files.

Configuration options:
  -N file=<file>    Specify name of config file.
  -N sys=<name>     Specify system name.

C generation options:  Control the behaviour of '-Fc' '-Fo' and '-Fx'.
  -C standard       '-Fc' generates ANSI/ISO standard C.
  -C old            '-Fc' generates old C (the default).
                    The options '-Cstandard' and '-Cold' are mutually exclusive.

  -C comp=<name>    Use <name> instead of the default C compiler.
  -C link=<name>    Use <name> instead of the default linker.
  -C cc=<name>      Use <name> instead of the default C compiler and linker.
                    Without '-Ccc=', the environment variable 'CC' is tried.
  -C go=<name>      Use <name> to run the output of the linker.
                    Without '-Cgo=', the environment variable 'CGO' is tried.

  -C args=<opts>    Pass <opts> as options to the C compiler.
  -C smax=<n>       Try to put no more than <n> statements per file,
                    if necessary splitting the generated file into:
                    <name>.h, <name>.c, <name>001.c, <name>002.c, etc.
                    Using -C smax=0 turns off file splitting.
                    (default: smax=2000)
```

```
-C idlen=<n>     Set the maximum length of C identifiers to be <n>.
                 Using -C idlen=0 turns off identifier truncation.
                 (default: idlen=32)
-C {no-}idhash   Use hash codes in global C identifiers.
                 (default: idhash)
-C {no-}lines    Preserve Aldor source line numbers in generated C.
                 (default: no-lines).

-C sys=<name>    Pass '-W sys=<name> to the C compiler and linker.
                 The unicl driver understands this option.
-C runtime=<name1>,<name2>
                 Link in lib<name1>.a lib<name2>.a as the runtime library
                 (default: foam).
-C fortran       Pass '-Wfortran' to link driver which adds fortran options
                 such as fortran runtime libraries.
-C lib=<name>    Link with the library lib<name>.a
```

```
C++ generation options:  Control the behaviour of ‘-Fc++’.
 -P basicfile=<bf> If the filename <bf> (absolute filename) is provided,
                the standard basic types correspondence between C++
                and Aldor will be overridden.
                If this option is not provided, the compiler uses
                ’basic.typ’ located in $AXIOMXLROOT/include.

 -P discrim-return Will discriminate functions on the return type by changing
                the name of the function to ’fnname_return-type’.

 -P no-discrim-return    Won’t discriminate functions on the return type.
                Names of the functions will be as the original, however
                the code generated for overloaded functions on the return
                type only won’t compile.
                This option is the default.


Lisp generation options:  Control the behaviour of ‘-Flsp’.
 -S common       ‘-Flsp’ produces Common Lisp code   (the default).
                 The options ‘-Scommon’, ‘-Sstandard’ and ‘-Sscheme’
                 are mutually exclusive.

 -S ftype=<ft>   Use <ft> as the file type for the generated lisp file
                 (default: ‘-Sftype=lsp’).
```

```
Message options:  Use ‘-M no-<word>’ to negate ‘-M <word>’.
 -M <n>           Control how much detail is given.       (default: ‘-M2’)
 -M no-<opt>      Turn off ‘-M <opt>’.
                                                          M0  M1  M2  M3
 -M warnings      Display warnings.                        N   Y   Y   Y
 -M number        Display the order of discovery number.   N   Y   Y   Y
 -M source        Display the program lines for messages.  N   N   Y   Y
 -M details       Display details.                         N   N   Y   Y
 -M notes         Display cross reference notess.          N   N   Y   Y
 -M remarks       Display remarks.                         N   N   N   Y

 -M sort          Sort messages by source position.        Y   Y   Y   Y
 -M mactext       Point to macro text, rather than use.    Y   Y   Y   Y
 -M abbrev        Abbreviate types in messages.            Y   Y   Y   Y
 -M human         Human-oriented format.                   Y   Y   Y   Y
 -M name          Show the name of each message.           N   N   N   N
 -M antiques      Display warnings for old-style code.     N   N   N   N
 -M preview       Display messages as they occur.          N   N   N   N
 -M inspect       Use interactive inspector for errors.    N   N   N   N
 -M emax=<n>      Stop after <n> error messages.          (default: 10)
 -M db=<fn>       Use <fn> as the message database.       (default: no-db)
 -M <msgname>     Turn on warning or remark named <msgname>
 -M no-<msgname>  Turn off warning or remark named <msgname>
                  Use ‘-Mname’ to find the names of messages.

                  These can be combined, e.g. ‘-M2 -Mno-source -Mpreview’
```

```
Developer options (subject to change):

 -W debug         Turn on experimental debugging code generation.
 -W depend        Print compile-time dependencies for this file.
 -W small-hcodes Turn on experimental short hashcode generation.
 -W emerge-noalias Work around for optimizer bug.
 -W check         Turn on internal safety checks.
 -W nhash         Assume all exported types have constant hashcodes.
 -W trace-cfuns  Runtime announcements of entry to generated C functions.
 -W no-keyword=<name>
                  The symbol <name> will not be treated as a keyword. This
                  may only be used to compile old files containing symbols
                  that were later made into reserved words (e.g. ref)
 -W name=<name>  Compile file as if it was called <name>.as.
 -W prefix=<prefix> Compile file as if it had <prefix> at the start of its name.
 -W runtime       Produce code suitable for the runtime system.
 -W loops         Always inline generators when possible.
 -W missing-ok   Do not stop compilation if some exports are missing.
 -W audit         Set maximum foam auditing level.
 -W dumb-import  Do not trace imported domains for their base value.
 -W rtcache=N    Set the runtime cache size for domain constructors to N.
                  (the default is N=15; use N=0 for the libfoam default)
 -W trap          Trap failure exits (for debugging Aldor).

 -W gc            Garbage collect as needed (if gc is available).
```

```
-W gcfile       Garbage collect after each file (if gc is available).

-W sexpr        Run a read-print loop.
-W seval        Run a read-eval-print loop.

-W test=<name>  Run compiler self-test <name>.
                Use ‘-W test=show’ to see a list of self-tests.
-W D+<name>     Turn on debug hook <name>.
                Use ‘-W D+show’ to see a list of debug hooks.
-W D-<name>     Turn off debug hook <name>.
                Use ‘-W D+show’ to see a list of debug hooks.
-W T[apdrgst0]+<ph>
                Phase tracing:
                  -WTa+<ph>       Announce entry to <ph>
                  -WTp+<ph>       Pretty print result of <ph>
                  -WTd+<ph>       Print debug information for <ph>
                  -WTr+<ph>       Show result of <ph>
                  -WTg+<ph>       Garbage collect after <ph>
                  -WTs+<ph>       Storage audit after <ph>
                  -WTt+<ph>       Terminate after <ph>
                  -WT0+<ph>       Ignore earlier ‘-WT’ option for <ph>
                Several phases can be given as ‘ph1+ph2+ph3’ or ‘all’.
                Several options can be given at once, e.g. ‘-WTags+all’.
                Compile a file with ‘-v’ to see the phase abbreviations.
```

Example – Source Code

```
#include "axllib.as"

SI ==> SingleInteger;
F  ==> DoubleFloat;
CF ==> Complex F;

import from CF;
inline from CF;

default xa, xb, ya, yb: F;
default xn, yn, MAX:   SI;
default draw: (x: SI, y: SI, n: SI) -> ();

drawMand(xa, xb, xn, ya, yb, yn, draw, MAX): () == {

  mandel(c: CF): SI == {
      z: CF := 0;
      n: SI := 0;
      while norm z < 4.0 for free n in 1..MAX repeat z  := z*z + c;
      n
    }
   for y in step(yn)(ya, yb) for yi in 1..yn repeat
      for x in step(xn)(xa, xb) for xi in 1..xn repeat
          draw(xi, yi, mandel complex(x, y));
}
```

Abstract Syntax

```
union abSyn {
    /* Generic views. */
    struct abHdr         abHdr;
    struct abGen         abGen;

    /* Leaves */
    struct abBlank       abBlank;
    struct abId          abId;
    struct abIdSy        abIdSy;

    struct abDocText     abDocText;
    struct abLitInteger  abLitInteger;
    struct abLitString   abLitString;
    struct abLitFloat    abLitFloat;

    /* Interior */
    struct abAdd         abAdd;
    struct abAnd         abAnd;
    ...
    struct abWith        abWith;
    struct abYield       abYield;
};
```

```
/* Generic View */

struct abHdr {
    BPack(AbSynTag)   tag;
    BPack(AbUse)      use;
    BPack(AbState)    state;

    Length            argc;
    SrcPosStack       pos;

    AbSeman           seman;

    union {
        SymeList poss;       /* during tiBottomUp */
        Syme     unique;     /* after  tiTopDown  */
    } meaning; /* ... for Ids and Literals */

    union {
        TPoss    poss;       /* during tiBottomUp */
        TForm    unique;     /* after  tiTopDown  */
    } type;
;
```

Example: Abstract Syntax Tree

```
(Sequence
  (Sequence () ())
  (Sequence () () () () () () () () () ())
  (Sequence (Import () AxlLib) (Inline () AxlLib))
  (Import () Boolean)
  (Import
    (With
      ()
      (Sequence
        (Declare string (Apply -> Literal %))
        (Declare << (Apply -> (Comma TextWriter %) TextWriter))
        (Declare << (Apply -> % (Apply -> TextWriter TextWriter)))))
    String)
  (Import
    (With
      ()
      (Sequence
        (Declare newline %)
        (Declare << (Apply -> (Comma TextWriter %) TextWriter))
        (Declare << (Apply -> % (Apply -> TextWriter TextWriter)))))
    Character)
  (Import
    (With () (Sequence (Declare print %) (Declare error %)))
    TextWriter)
```

```
(Import () FormattedOutput)
()
()
()
(Import () (Apply Complex DoubleFloat))
(Inline () (Apply Complex DoubleFloat))
(Default (Declare (Comma xa xb ya yb) DoubleFloat))
(Default (Declare (Comma xn yn MAX) SingleInteger))
(Default
  (Declare
    draw
    (Apply
      ->
      (Comma
        (Declare x SingleInteger)
        (Declare y SingleInteger)
        (Declare n SingleInteger))
      (Comma))))
(Define
  (Declare
    drawMand
    (Apply
      ->
      (Comma
        (Declare xa ())
        (Declare xb ())
```

```
      (Declare xn ())
      (Declare ya ())
      (Declare yb ())
      (Declare yn ())
      (Declare draw ())
      (Declare MAX ()))
    (Comma)))
(Lambda
  (Comma
    (Declare xa ())
    (Declare xb ())
    (Declare xn ())
    (Declare ya ())
    (Declare yb ())
    (Declare yn ())
    (Declare draw ())
    (Declare MAX ()))
  (Comma)
  (Label
    drawMand
    (Sequence
      (Define
        (Declare
          mandel
          (Apply -> (Declare c (Apply Complex DoubleFloat)) SingleInteger))
        (Lambda
```

```
    (Comma (Declare c (Apply Complex DoubleFloat)))
    SingleInteger
    (Label
      mandel
      (Sequence
        (Assign (Declare z (Apply Complex DoubleFloat)) \0)
        (Assign (Declare n SingleInteger) \0)
        (Repeat
          (Assign z (Apply + (Apply * z z) c))
          (While (Test (Apply < (Apply norm z) (LitFloat "4.0"))))
          (For (Free n) (Apply |..| \1 MAX) ()))
        n))))
(Repeat
  (Repeat
    (Apply draw xi yi (Apply mandel (Apply complex x y)))
    (For x (Apply (Apply step xn) xa xb) ())
    (For xi (Apply |..| \1 xn) ()))
  (For y (Apply (Apply step yn) ya yb) ())
  (For yi (Apply |..| \1 yn) ())))))))
```

Type Inference

- Determine meanings for all symbols in a parse tree.
- Find the *type* and *origin*.
- Two passes:
  - TiBottomUp: assemble set of possible interpretations for each node.
  - TiTopDown: impose constraint that each expression can have at most *one* interpretation.

Type Inference — Complications

- Overloading
- Mutually recursive typings

```
Type : Type

Tuple: Type -> Type
->   : (Tuple Type, Tuple Type)  -> Type
```

- Ambiguity on return type.
- Dependent types and their scopes.

Symbol Meaning

- Different kinds of symbols:

```
Label
Param   LexVar   LexConst Fluid
Import  Export
Extend  Library  Archive
Builtin Foreign  Temp
```

- Information per symbol instance:

```
struct syme {
    BPack(UByte)    fieldc;     /* Length of fieldv */
    BPack(SymeTag)  kind;       /* How the symbol is used */
    UShort          bits;       /* Bit fields */

    Symbol          id;         /* The symbol being defined */
    Lib             lib;        /* Library for lazy info */
    Hash            hash;       /* Hash code for lazy info */
    TForm           type;       /* The type of the symbol */

    ULong           mask;       /* Fields found in fieldv */
    Syme            full;       /* Syme for other fields */
    AInt *          fieldv;     /* Field values */
};
```

- Fields:

```
enum symeField {
    SYFI_Origin,          /* Origin as a pointer */
    SYFI_Exporter,        /* Exporter for import symes */
    SYFI_Comment,         /* Documentation for export symes */
    SYFI_Extendee,        /* Extendees for extend symes */
    SYFI_Library,         /* Library for library symes */
    SYFI_Archive,         /* Archive for archive symes */
    SYFI_Builtin,         /* Builtin tag for builtin symes */
    SYFI_Foreign,         /* Foreign origin for foreign symes */
    SYFI_Foam,            /* foam for temporary symes */
    SYFI_HasTest,         /* Has Expr for 'has' symes */
    SYFI_Original,        /* Unsubstituted self */
    SYFI_Extension,       /* Extend syme for extendee symes */
    SYFI_Condition,       /* Condition for conditional symes */
    SYFI_Twins,           /* Equivalent symes */
    SYFI_Depths,          /* sefoFreeVars depth numbers */
    SYFI_Mark,            /* Sefo traversal mark */
    SYFI_DefLevel,        /* Level where syme is bound */
    SYFI_LibNum,          /* Serial number in syme section */
    SYFI_VarIndex,        /* Serial number in stab level */
    SYFI_UsedDepth,       /* Max relative depth used */
    SYFI_IntStepNo,       /* Interactive step where defined */
    SYFI_FoamKind,        /* Par, Loc, Lex, Glo */
    SYFI_Closure,         /* Foam closure */
```

```
    SYFI_Inlined,           /* List of inlined symes */
    SYFI_DVMark,            /* Dead variable elimination mark */
    SYFI_SImpl,             /* Implementation information */
    SYFI_ConstLib,          /* Library for foam constant info */
    SYFI_ConstInfo,         /* Constant number and opt. flags*/
    SYFI_DefnNum,           /* Sequence number when defined */
    SYFI_HashNum,           /* Runtime hash code */
    SYFI_ExtraBits,         /* More syme bits */
};
```

Semantics at Each Node

```
struct abSeman {
    Doc      comment;     /* Comments attached to the absyn. */
    Stab     stab;        /* For lexical level. */
    int      defnIdx;     /* Definition index */
    Syme     syme;        /* Meaning of id or literal. */
    TForm    tform;       /* In type context. */
    AbSyn    implicit;    /* Implicit operator on expression. */
    AbEmbed  embed;       /* Implicit embedding for product contexts. */
    SImpl    impl;        /* Syme implementation, if any */
};
```

Example — Types of exported symbols

```
((|Declare|
    |drawMand|
    (|Apply|
      ->
      (|Comma|
        (|Declare|
          |xa|
          |DoubleFloat|
          ((|symeNameCode| . 51509931) (|symeTypeCode| . 754279279)))
        (|Declare|
          |xb|
          |DoubleFloat|
          ((|symeNameCode| . 51509932) (|symeTypeCode| . 754279279)))
        (|Declare|
          |xn|
          |SingleInteger|
          ((|symeNameCode| . 51509944) (|symeTypeCode| . 83725344)))
        (|Declare|
          |ya|
          |DoubleFloat|
          ((|symeNameCode| . 51510700) (|symeTypeCode| . 754279279)))
        (|Declare|
          |yb|
          |DoubleFloat|
```

```
  ((|symeNameCode| . 51510701) (|symeTypeCode| . 754279279)))
(|Declare|
  |yn|
  |SingleInteger|
  ((|symeNameCode| . 51510713) (|symeTypeCode| . 83725344)))
(|Declare|
  |draw|
  (|Apply|
    ->
    (|Comma|
      (|Declare|
        |x|
        |SingleInteger|
        ((|symeNameCode| . 200161) (|symeTypeCode| . 83725344)))
      (|Declare|
        |y|
        |SingleInteger|
        ((|symeNameCode| . 200162) (|symeTypeCode| . 83725344)))
      (|Declare|
            |n|
            |SingleInteger|
        ((|symeNameCode| . 200151) (|symeTypeCode| . 83725344))))
    (|Comma|))
  ((|symeNameCode| . 815338578) (|symeTypeCode| . 326007403)))
(|Declare|
  max
```

```
        |SingleInteger|
        ((|symeNameCode| . 316389153) (|symeTypeCode| . 83725344))))
     (|Comma|))
   ((|symeNameCode| . 259747446) (|symeTypeCode| . 659738015))))
(|Sequence|)
```

Foam: Intermediate Code

- First order: functions and types now handled explicitly
- Target level:
  Maps simply to register-based or stack-based
  Maps simply to Lisp, C, or assembly level
- Primitive types:

```
Nil   Char   Bool   Byte   HInt    SInt   SFlo DFlo Word Arb
Int8  Int16  Int32  Int64  Int128
Ptr   Env    Arr    Rec    Prog    Clos
```

- Data access:

```
Par[i]     Lex[i,j]   Fluid[i]
Glo[i]     Const[i]
RElt[f,i]  AElt[t,i]  EElt[i,j,k]
```

- Control:

```
NOp    Set
Seq    If       Select  Goto   Throw  Catch  Return
PCall  BCall    CCall   OCall
Cast   PushEnv  PopEnv  Values Kill   Free
```

- Data operations, e.g.:

```
SInt0  SInt1  SIntMin  SIntMax
SIntIsZero  SIntIsNeg  SIntIsPos  SIntIsEven  SIntIsOdd
SIntEQ  SIntNE  SIntLT  SIntLE
SIntNegate  SIntPrev  SIntNext
SIntPlus  SIntMinus  SIntTimes  SIntTimesPlus
SIntMod  SIntQuo  SIntRem  SIntDivide  SIntGcd
SIntPlusMod  SIntMinusMod  SIntTimesMod  SIntTimesModInv  SIntLength
SIntShiftUp  SIntShiftDn  SIntBit  SIntNot  SIntAnd  SIntOr  SIntXOr

WordTimesDouble  WordDivideDouble  WordPlusStep  WordTimesStep

DFlo0  DFlo1  DFloMin DFloMax  DFloEpsilon
DFloIsZero  DFloIsNeg  DFloIsPos
DFloEQ DFloNE  DFloLT  DFloLE
DFloNegate  DFloPrev DFloNext
DFloPlus  DFloMinus  DFloTimes  DFloTimesPlus DFloDivide
DFloRPlus  DFloRMinus  DFloRTimes  DFloRTimesPlus DFloRDivide
DFloDissemble  DFloAssemble

...
```

Example — FOAM for the Inner Loop

```
  ...
(Label 4)
 (CCall NOp (Loc 14))
 (If (Cast Bool (CCall Word (Loc 13))) 5)
 (Set (Loc 12 x) (CCall Word (Loc 15)))
 (CCall NOp (Loc 19))
 (If (Cast Bool (CCall Word (Loc 18))) 5)
 (Set (Loc 11 xi) (CCall Word (Loc 20)))
 (CCall NOp (Par 6 draw) (Loc 11 xi) (Loc 0 yi)
   (OCall Word (Const 2 mandel) (Env 0)
     (CCall Word (Lex 1 9 complex) (Loc 12 x) (Loc 1 y))))
 (Goto 4)
   ...
```

The Runtime System

- Memory allocation + GC
- Big integer arithmetic
- Stack unwinding (exceptions, dynamically scoped variables)
- Export lookup from domains
- Dynamic linking

- Written in C and Aldor

Domain Representation

- Layers of lazy evaluation to elaborate domain objects as needed.
- Ultimately an array of pointers to functions.
- Reponds to queries: give me export of name `<hash1>` and type `<hash2>`. Looks up "`add`" inheritance tree and category defaults.
- Cache imports into local, fixed-location slots.

Optimization

Details in **Implementation II**.

- Procedural integration (inlining).
- Data structure elimination.
- Constant propagation, etc.

- Certain easy optimizations delegated to concrete code back end.

Example — Optimized FOAM for the Inner Loop

```
  ...
  (Set (Loc 10 double) (DFlo 4.0000000000000000e0))
  (If (BCall DFloLE (Loc 10 double) (Loc 13 double)) 9)
  (Set (Loc 9 a) (SInt 1))
(Label 1)
  (If (BCall SIntLT (Cast SInt (Par 7 MAX)) (Loc 9 a)) 2)
  (Set (Loc 0) (Loc 9 a))
  (Set (Loc 20 double) (BCall DFloMinus (Loc 27) (Loc 28)))
  (Set (Loc 15 double) (BCall DFloTimes (Loc 17 double) (Loc 18 double)))
  (Set (Loc 14 double) (BCall DFloTimes (Loc 18 double) (Loc 17 double)))
  (Set (Loc 19 double) (BCall DFloPlus (Loc 15 double) (Loc 14 double)))
  (Set (Loc 17 double) (BCall DFloPlus (Loc 20 double) (Loc 22 double)))
  (Set (Loc 18 double) (BCall DFloPlus (Loc 19 double) (Loc 24 double)))
  (Set (Loc 27) (BCall DFloTimes (Loc 17 double) (Loc 17 double)))
  (Set (Loc 28) (BCall DFloTimes (Loc 18 double) (Loc 18 double)))
  (Set (Loc 13 double) (BCall DFloPlus (Loc 27) (Loc 28)))
  (Set (Loc 10 double) (DFlo 4.0000000000000000e0))
  (If (BCall DFloLE (Loc 10 double) (Loc 13 double)) 9)
  (Set (Loc 9 a) (BCall SIntNext (Loc 9 a)))
  (Goto 1)
(Label 9)
  (CCall NOp (Par 6 draw) (Cast Word (Loc 8 a)) (Cast Word (Loc 16 a)) (Cast Word (Loc 0)))
  ...
```

Generated C for the Inner Loop

```
        ...
        T27 = T17_double * T17_double;
        T28 = T18_double * T18_double;
        T13_double = T27 + T28;
        T10_double = 4.000000000000000;
        if (T10_double <= T13_double) goto L9;
        T9_a = 1;
L1:     if ((FiSInt) P7_MAX < T9_a) goto L2;
        T0 = T9_a;
        T20_double = T27 - T28;
        T15_double = T17_double * T18_double;
        T14_double = T18_double * T17_double;
        T19_double = T15_double + T14_double;
        T17_double = T20_double + T22_double;
        T18_double = T19_double + T24_double;
        T27 = T17_double * T17_double;
        T28 = T18_double * T18_double;
        T13_double = T27 + T28;
        T10_double = 4.000000000000000;
        if (T10_double <= T13_double) goto L9;
        T9_a = T9_a + 1;
        goto L1;
L9:     fiCCall3(void, P6_draw,
                 (FiWord) T8_a, (FiWord) T16_a, (FiWord) T0);
        ...
```

Generated Common Lisp for the Inner Loop

```
 ...
 (setq t27 (|DFloTimes| |T17-double| |T17-double|))
 (setq t28 (|DFloTimes| |T18-double| |T18-double|))
 (setq |T13-double| (|DFloPlus| t27 t28))
 (setq |T10-double| (the |DFlo| 4.0000000000000000e0))
 (when (|DFloLE| |T10-double| |T13-double|) (go |Lab9|))
 (setq |T9-a| (the |SInt| 1))
Lab1|
 (when (|SIntLT| p7-max |T9-a|) (go |Lab2|))
 (setq t0 |T9-a|)
 (setq |T20-double| (|DFloMinus| t27 t28))
 (setq |T15-double| (|DFloTimes| |T17-double| |T18-double|))
 (setq |T14-double| (|DFloTimes| |T18-double| |T17-double|))
 (setq |T19-double| (|DFloPlus| |T15-double| |T14-double|))
 (setq |T17-double| (|DFloPlus| |T20-double| |T22-double|))
 (setq |T18-double| (|DFloPlus| |T19-double| |T24-double|))
 (setq t27 (|DFloTimes| |T17-double| |T17-double|))
 (setq t28 (|DFloTimes| |T18-double| |T18-double|))
 (setq |T13-double| (|DFloPlus| t27 t28))
 (setq |T10-double| (the |DFlo| 4.0000000000000000e0))
 (when (|DFloLE| |T10-double| |T13-double|) (go |Lab9|))
 (setq |T9-a| (|SIntNext| |T9-a|))
 (go |Lab1|)
Lab9|
 (|CCall| |P6-draw| |T8-a| |T16-a| t0)
 ...
```