A Fortran Code Transformer

Paul Madden

paul.a.madden@noaa.gov

The Scalable Modeling System

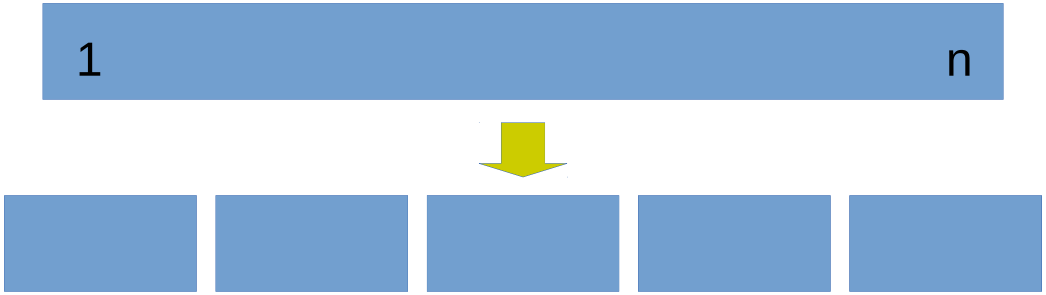
- SMS developed mainly 1990-2000
  - Directive-based parallelization of Fortran codes
  - Front-end translator (PPP)
    - Interprets directives and outputs new, parallel code
    - Fix distributed-array loop bounds and array indices, serialize IO statements, allow serial serial regions in otherwise parallel code
  - Back-end library
    - API routines, MPI interface, decomposition information

SMS Examples: sms$distribute

```
!sms$distribute begin
  real :: u(:)
!sms$distribute end

allocate (u(1:n))

do i=1,n
  u(i)=i
enddo
```



```
allocate (u(sms__local_lo:sms__local_hi))

do i=sms__local_lo,sms__local_hi
  u(i)=i
enddo
```

SMS Examples: Implicit Translation

```
read (lun) u
```

```
if (sms__i_am_root()) then
  allocate (sms__global_u(sms__global_size))
  read (lun) sms__global_u
endif
sms__scatter(sms__global_u,u)
if (sms__i_am_root()) then
  deallocate (sms__global_u)
endif
```

SMS Examples: sms$serial

```
sum=0.0
!sms$serial begin
do i=1,n
  sum=sum+u(i)
enddo
!sms$serial end
```

```
sum=0.0
sms__gather(sms__global_u,u)
if (sms__i_am_root()) then
  do i=1,n
    sum=sum+sms__global_u(i)
  enddo
endif
sms__bcast(sum)
```

Avoid floating-point order-of operation differences

SMS Weaknesses

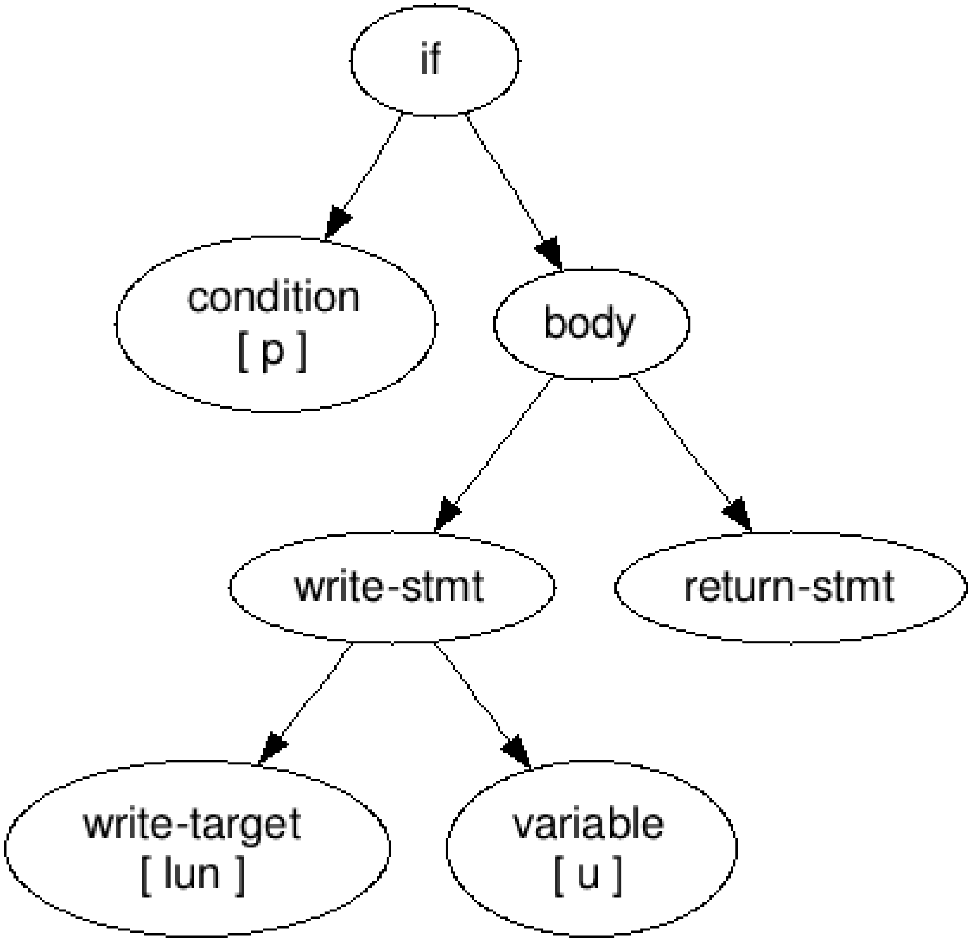
- Based on Eli compiler toolkit
  - Powerful, but complicated and unfamiliar
  - F77 parser with incomplete F90 extensions
- Issues with
  - F90 features like modules (scoping, `use,only`), array syntax (array-index correction), kinds
  - Delineation of declaration, executable sections
  - Parallel builds
- Many workaround hacks in model code

Parsing Basics: AST

Code

```
if (p)
  write (lun) u
  return
endif
```

Abstract Syntax Tree



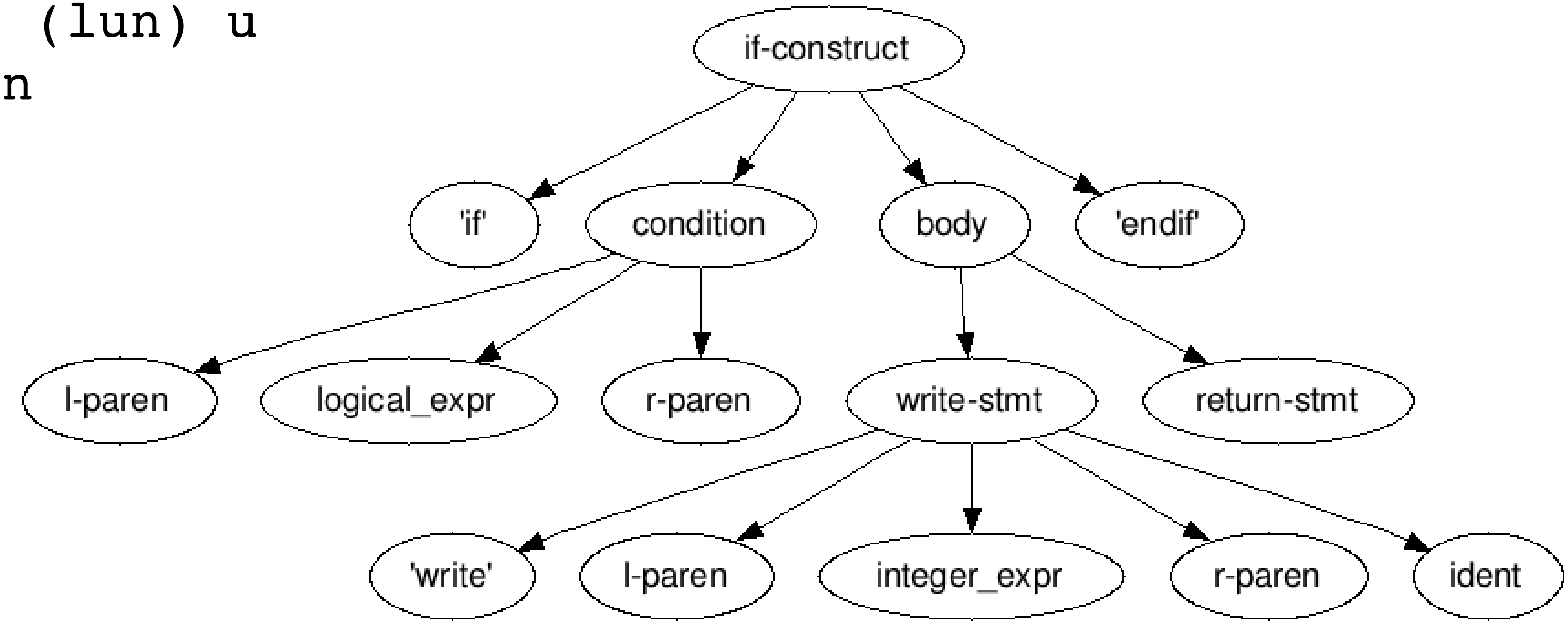
Parsing Basics: Parse Tree

Code

Parse Tree

aka Concrete Syntax Tree

```
if (p)
  write (lun) u
  return
endif
```

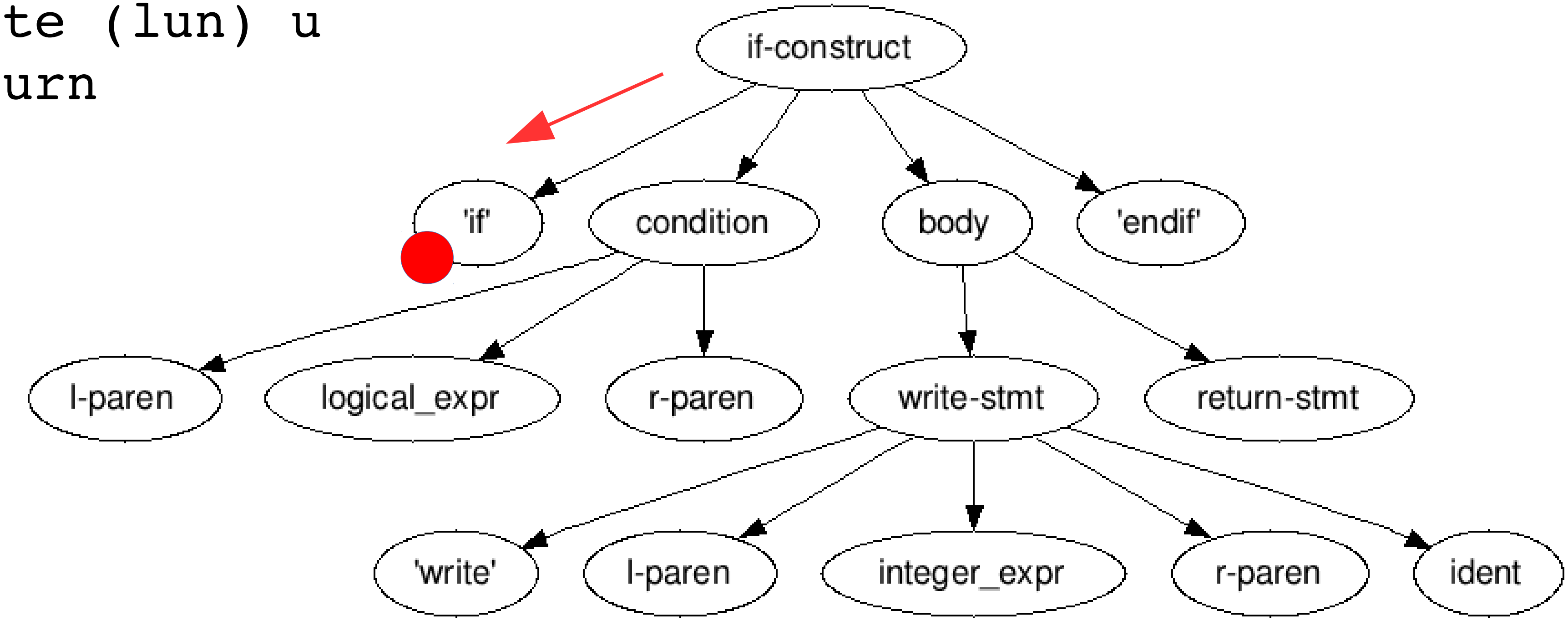


AST vs Parse Tree

- Simplified AST ideal for translation between different languages
  - e.g. Fortran to C, or C to assembly
- Parse Tree has benefits for translation within one language
  - e.g. Fortran to SMS-Fortran
- Simple to output an *identity* translation from parse tree...

Emit Text via Depth-First Walk

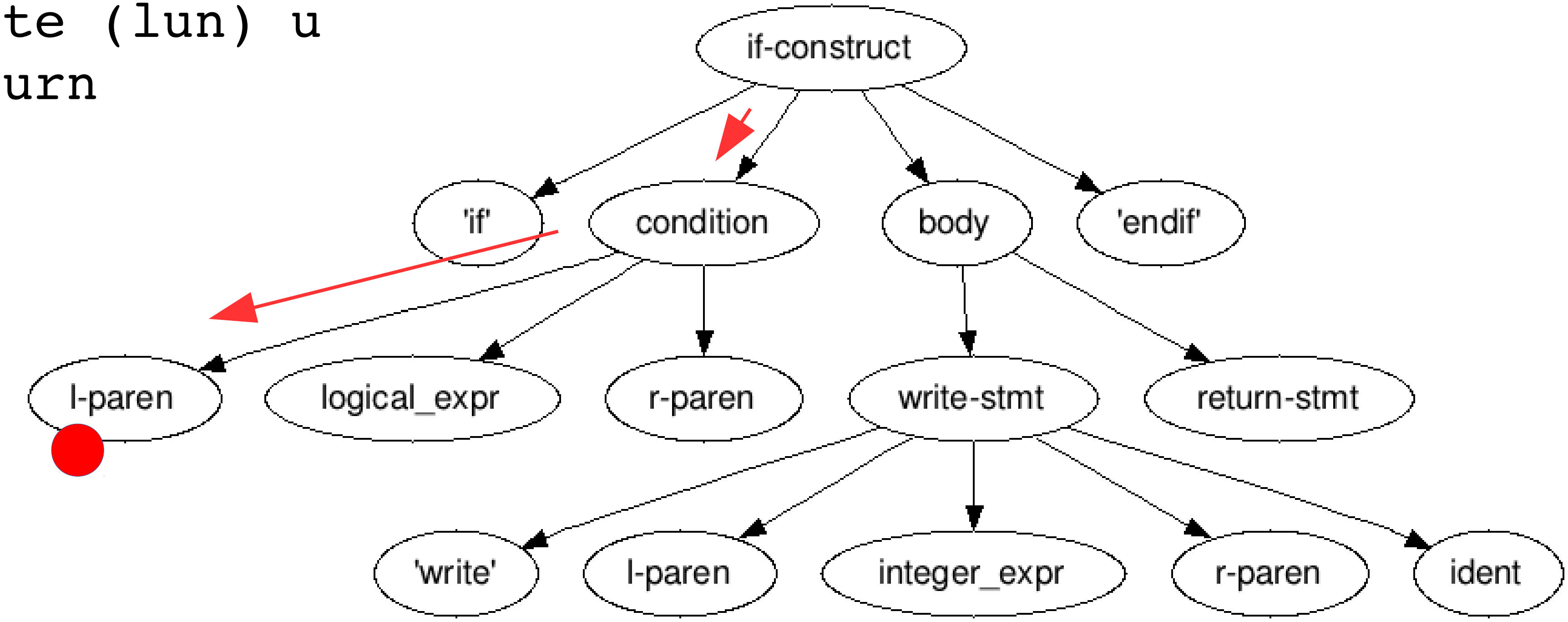
```
if (p)
  write (lun) u
  return
endif
```



```
if
```

Emit Text via Depth-First Walk

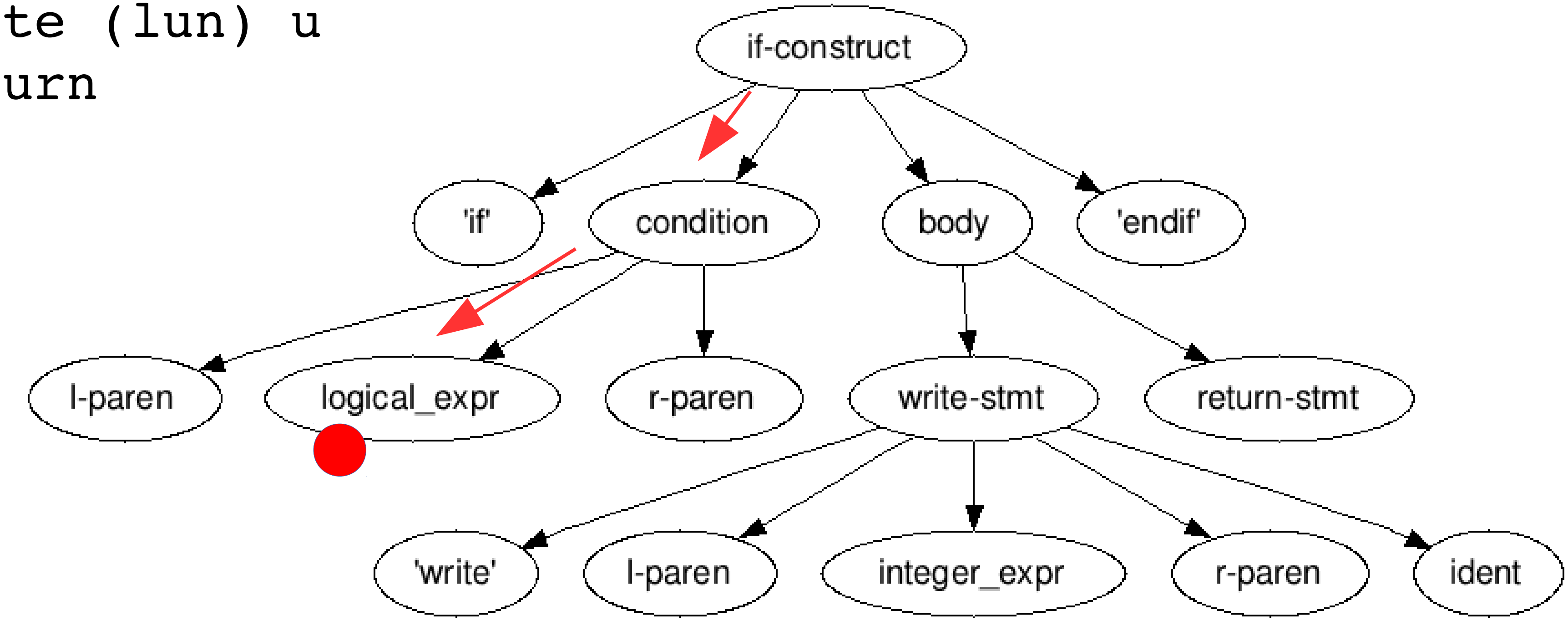
```
if (p)
  write (lun) u
  return
endif
```



```
if (
```

Emit Text via Depth-First Walk

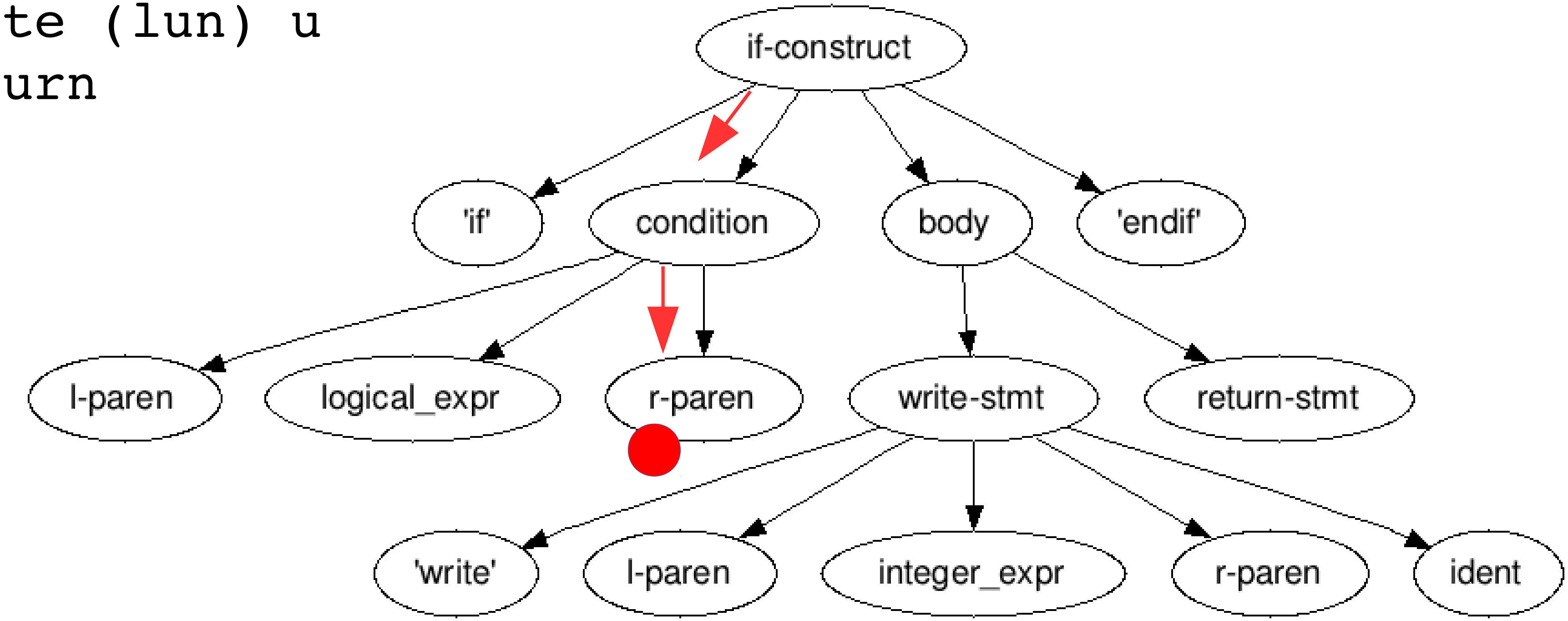
```
if (p)
  write (lun) u
  return
endif
```



```
if (p
```

Emit Text via Depth-First Walk

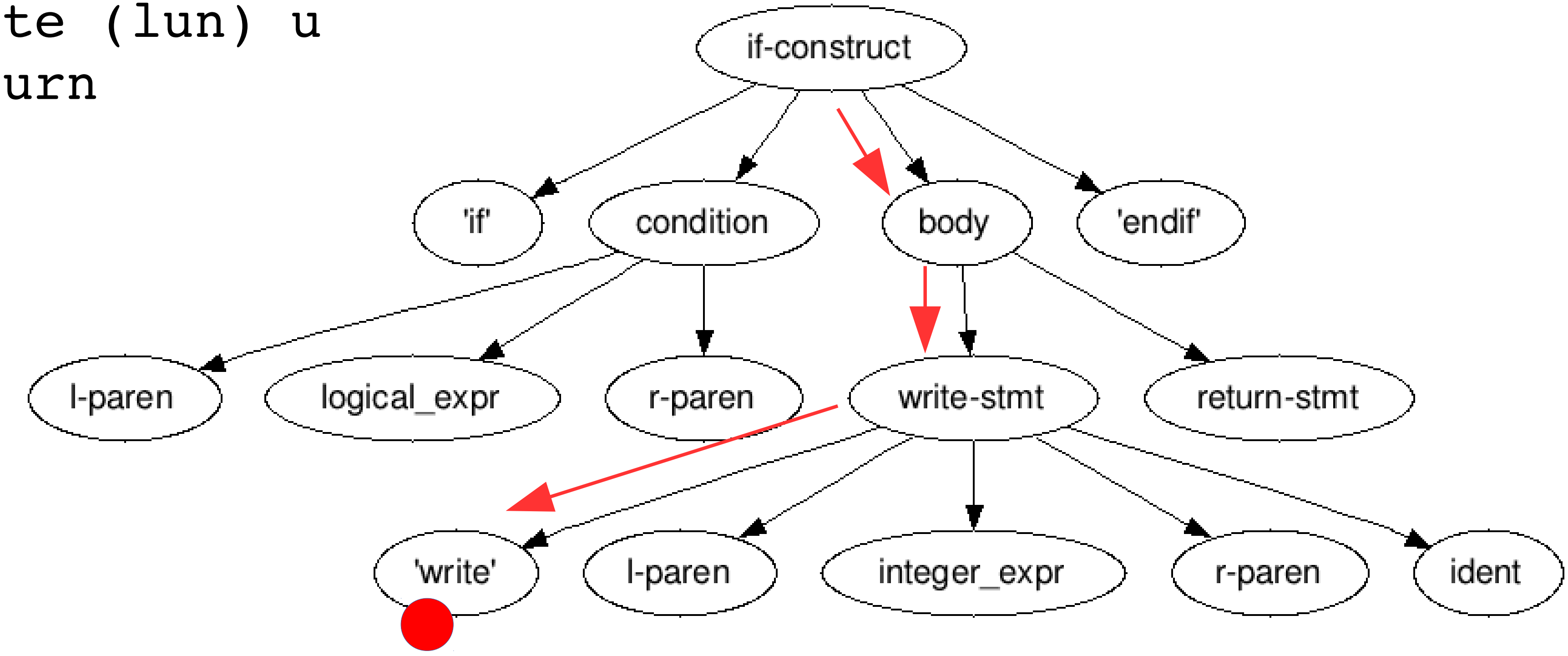
```
if (p)
  write (lun) u
  return
endif
```



```
if (p)
```

Emit Text via Depth-First Walk

```
if (p)
  write (lun) u
  return
endif
```



```
if (p)
  write
```

Parser Technologies

- Recursive Descent parsers
  - Often handwritten in a single language
  - Mutually-recursive procedures to consume and recognize language
  - Easy to write and understand, hard to maintain
- LR parsers
  - Generated from specifications (lex, yacc)
  - Must learn tool languages, understand their error messages
  - Harder to write and understand, easier to maintain

Parsing Expression Grammars

- Introduced by Bryan Ford in 2004
- Best of both worlds: A recursive-descent parser, generated by tools, from specifications
  - Ambiguities, ordering dealt with in grammar
  - Easy to understand and maintain
- Many implementations in various languages

PPP Implementation

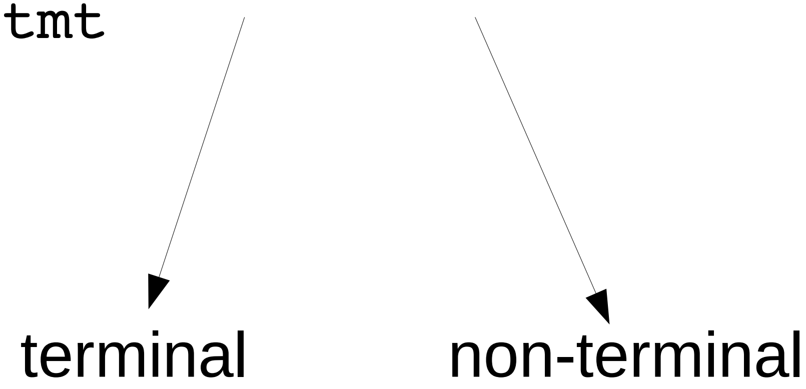
- Treetop, a Ruby-based PEG parser generator
  - Developed by Boulder's own Pivotal Labs
- Codebase
  - 6550 loc, core Fortran 90 support
  - 2475 loc, SMS extensions (incl. translation logic)
  - 884 loc, support utilities
- 3374 grammar loc → 38k generated parser loc
- 706 language-recognition tests
  - Including much cruel and unusual Fortran

Treetop PEG: Basic

```
rule if_stmt
  'if' condition 'then' stmt 'else' stmt /
  'if' condition 'then' stmt
end

rule stmt
  assign_stmt /
  do_stmt /
  if_stmt /
  ...
end

rule condition
  ...
end
```



terminal

non-terminal

Treetop PEG: Classes and Repetition

```
rule if_stmt
  if_t condition then_t stmt ( else_t stmt )? <If_Stmt>
end

rule if_t
  'if' <T>
end

rule block
  stmt+ <Block>
end
```

Ruby tree-node classes

Treetop PEG: Lookahead Assertions and Semantic Predicates

Negative lookahead assertion

```
rule assign_stmt
  var '=' val !comma_t
end
```

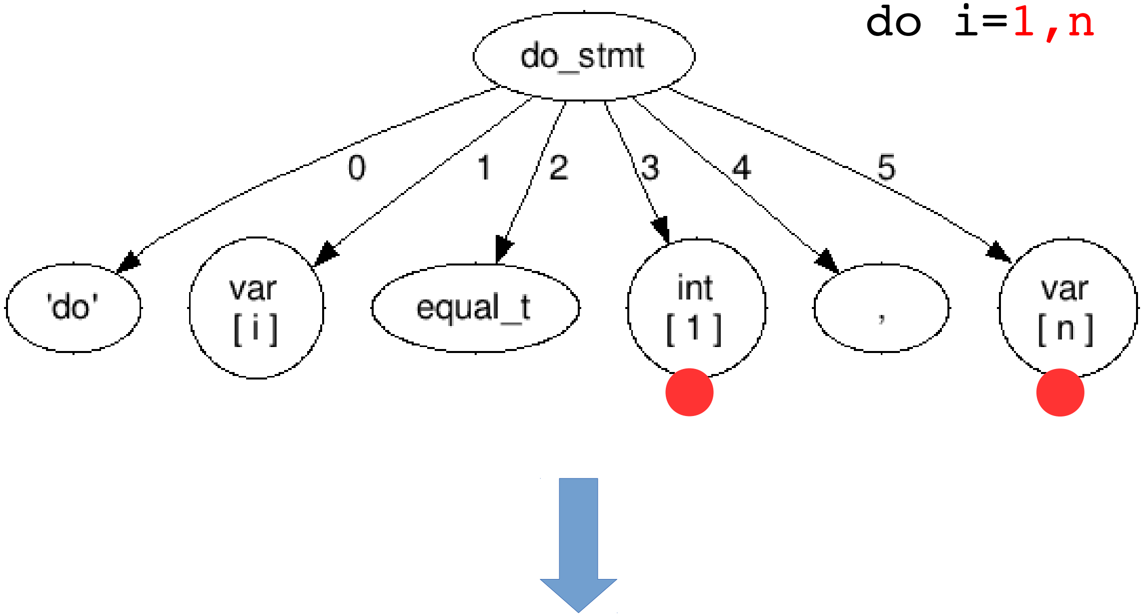
Positive semantic predicate

```
rule array_assignment
  var '=' var &{ |e| both_arrays?(e[0],e[2]) }
end
```

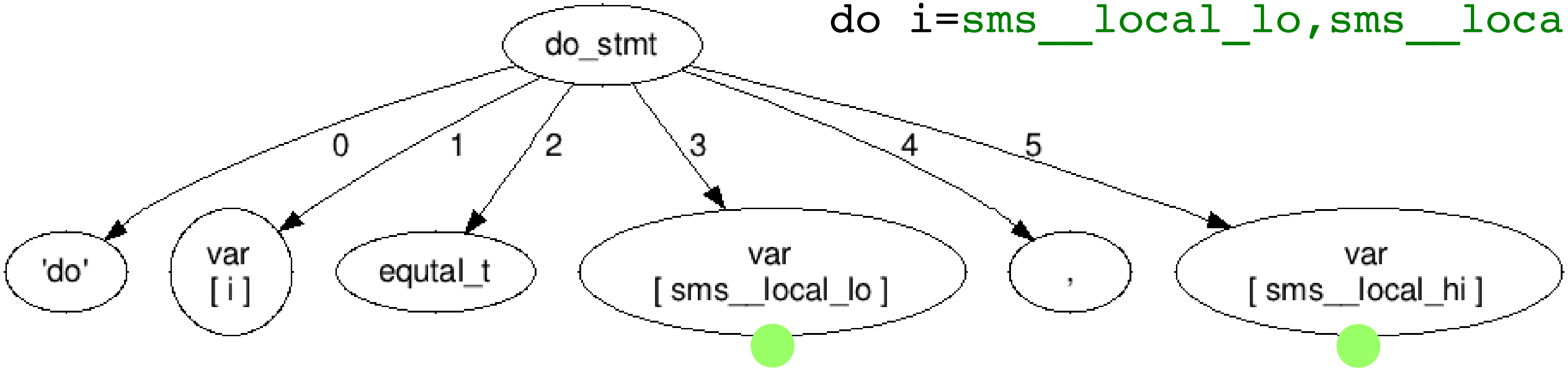
Semantic predicate only for side-effect

```
rule hollerith_with_report
  hollerith &{ |e| puts “Found one!”; true }
end
```

Translation via Tree Manipulation



```
do i=1,n
```



```
do i=sms__local_lo,sms__local_hi
```

Tree Manipulation Code

```
do i=1,n
0  12345
```

```
do i=sms__local_lo,sms__local_hi
0  123            45
```

In Ruby:

```
new_code=”do #{e[1]}=sms__local_lo,sms__local_hi”
new_tree=parse(new_code)
replace_node(this,new_tree)
```

Or, reuse even more elements from the parse tree:

```
new_code=”#{e[0]} #{e[1]}#{e[2]}sms__local_lo#{e[4]}sms__local_hi”
```

Translation

- First, obtain the parse tree.
- Second, a depth-first walk over the parse tree is performed for translation.
  - Some nodes have `translate` methods, for replacing themselves, recording, and/or error handling
  - Children are translated first, and can record information for later use by ancestor nodes.
- Then, a second depth-first walk over the transformed tree is done to emit translated source, ready for compilation.

Analysis 1

```
sum=0.0
!sms$serial begin
do i=1,n
  sum=sum+u(i)
enddo
!sms$serial end
```

During translation walk, we note that distributed array u occurs in the serial region, triggering roll-out of gather code. Similarly for scalar sum and broadcast code.

```
sum=0.0
sms__gather(sms__global_u,u)
if (sms__i_am_root()) then
  do i=1,n
    sum=sum+sms__global_u(i)
  enddo
endif
sms__bcast(sum)
```

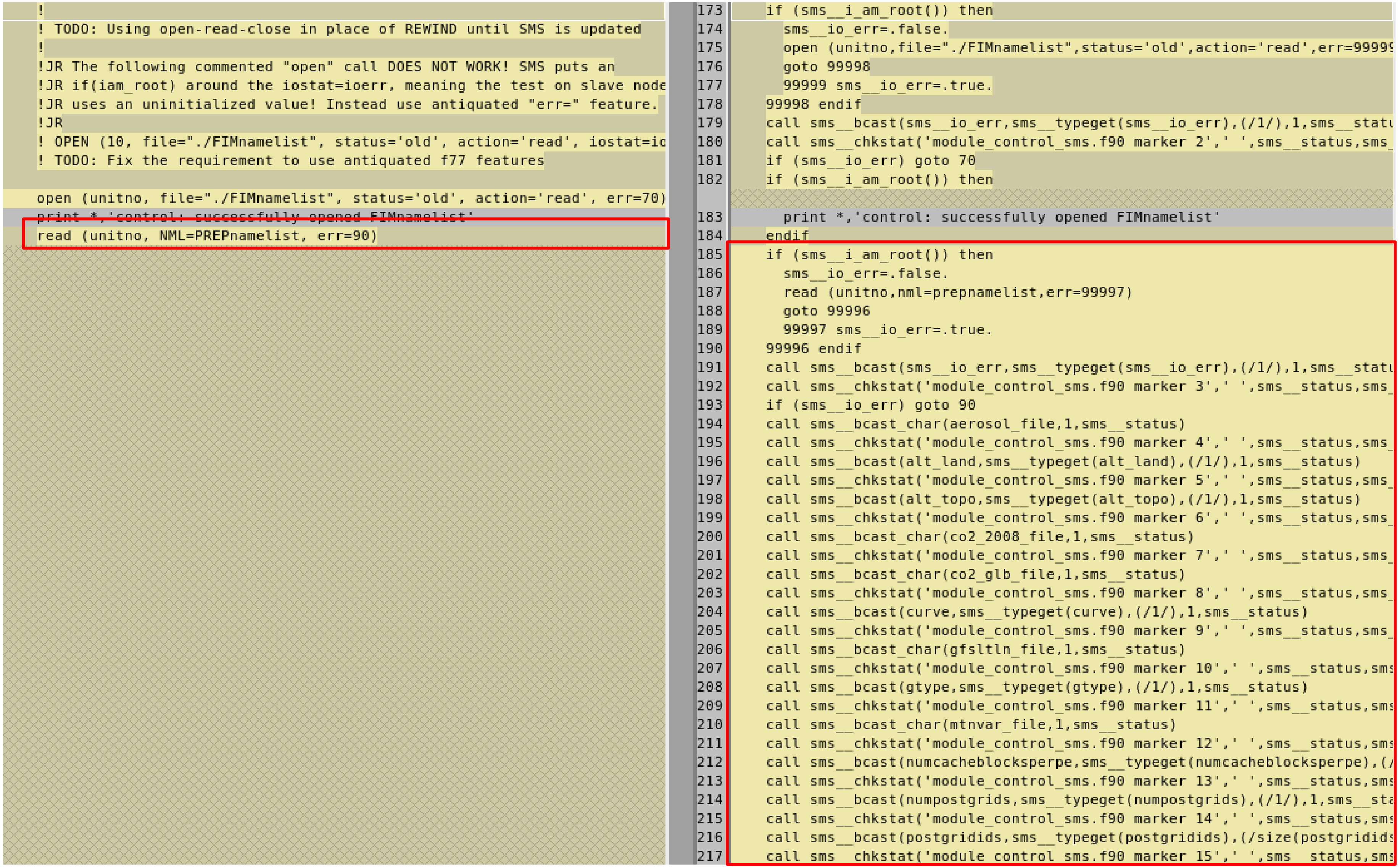
Analysis 2

```
if (p) goto 88
!sms$serial begin
do i=1,n
  if (u(i).gt.max) max=u(i)
enddo
88 print *,'hello'
!sms$serial end
```

Branch into serial region could permit some tasks from bypassing the collective gather, leading to a hang. So, branches into or out of serial regions are detected and rejected.

```
if (p) goto 88
sms__gather(sms__global_u,u)
if (sms__i_am_root()) then
  do i=1,n
    if (sms__global_u(i).gt.max) max=sms__global_u(i)
  enddo
  88 print *,'hello'
endif
```

Code Expansion



```
      !
      ! TODO: Using open-read-close in place of REWIND until SMS is updated
      !
      !JR The following commented "open" call DOES NOT WORK! SMS puts an
      !JR if(iam_root) around the iostat=ioerr, meaning the test on slave node
      !JR uses an uninitialized value! Instead use antiquated "err=" feature.
      !JR
      ! OPEN (10, file="./FIMnamelist", status='old', action='read', iostat=ic
      ! TODO: Fix the requirement to use antiquated f77 features

      open (unitno, file="./FIMnamelist", status='old', action='read', err=70)
      ~~print *,'control: successfully opened FIMnamelist'~~
      read (unitno, NML=PREPnamelist, err=90)
```

```
      if (sms__i_am_root()) then
        sms__io_err=.false.
        open (unitno,file="./FIMnamelist",status='old',action='read',err=9999
        goto 99998
        99999 sms__io_err=.true.
      99998 endif
      call sms__bcast(sms__io_err,sms__typeget(sms__io_err),(/1/),1,sms__statu
      call sms__chkstat('module_control_sms.f90 marker 2',' ',sms__status,sms_
      if (sms__io_err) goto 70
      if (sms__i_am_root()) then

        print *,'control: successfully opened FIMnamelist'
      endif
      if (sms__i_am_root()) then
        sms__io_err=.false.
        read (unitno,nml=prepnamelist,err=99997)
        goto 99996
        99997 sms__io_err=.true.
      99996 endif
      call sms__bcast(sms__io_err,sms__typeget(sms__io_err),(/1/),1,sms__statu
      call sms__chkstat('module_control_sms.f90 marker 3',' ',sms__status,sms_
      if (sms__io_err) goto 90
      call sms__bcast_char(aerosol_file,1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 4',' ',sms__status,sms_
      call sms__bcast(alt_land,sms__typeget(alt_land),(/1/),1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 5',' ',sms__status,sms_
      call sms__bcast(alt_topo,sms__typeget(alt_topo),(/1/),1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 6',' ',sms__status,sms_
      call sms__bcast_char(co2_2008_file,1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 7',' ',sms__status,sms_
      call sms__bcast_char(co2_glb_file,1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 8',' ',sms__status,sms_
      call sms__bcast(curve,sms__typeget(curve),(/1/),1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 9',' ',sms__status,sms_
      call sms__bcast_char(gfsltln_file,1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 10',' ',sms__status,sms
      call sms__bcast(gtype,sms__typeget(gtype),(/1/),1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 11',' ',sms__status,sms
      call sms__bcast_char(mtnvar_file,1,sms__status)
      call sms__chkstat('module_control_sms.f90 marker 12',' ',sms__status,sms
      call sms__bcast(numcacheblocksperpe,sms__typeget(numcacheblocksperpe),(/
      call sms__chkstat('module_control_sms.f90 marker 13',' ',sms__status,sms
      call sms__bcast(numpostgrids,sms__typeget(numpostgrids),(/1/),1,sms__sta
      call sms__chkstat('module_control_sms.f90 marker 14',' ',sms__status,sms
      call sms__bcast(postgridids,sms__typeget(postgridids),(/size(postgridids
      call sms__chkstat('module_control_sms.f90 marker 15',' ',sms__status,sms
```

Status

- Outcomes
  - FIM and NIM models updated to use new PPP
  - Many workaround hacks eliminated, better implicit translation of IO statements, further development is now tractable (lower bar to entry with Ruby)
- Issues
  - Some parses technically wrong, need derived-type and F95+ support, better support for other directive families, need unit tests
  - Can't broadcast pointer assignments with MPI

Future Work, Etc.

- Modular for other translator plug-ins
- Produce AST for easier re-use
- Open Fortran Parser / Rose
- Code at https://github.com/maddenp/ppp

Thanks!