Easier API Interoperability

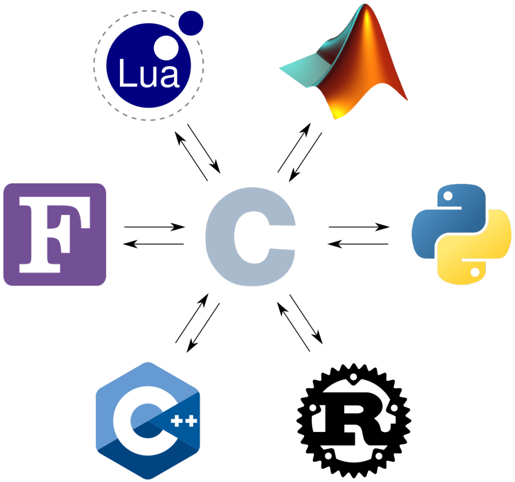
Writing a bindings generator to C/C++ with Coccinelle

Ivan PRIBEC, Michele MARTONE

Leibniz Supercomputing Centre of the Bavarian Academy of Sciences

02.02.2025

C as lingua franca



Source: Sebastian Ehlert, Using objects across language boundaries

C

```
...
void bli_dgemm(
           trans_t transa,
           trans_t transb,
             dim_t m,
             dim_t n,
             dim_t k,
     const double* alpha,
     const double* a, inc_t rsa, inc_t csa,
     const double* b, inc_t rsb, inc_t csb,
     const double* beta,
           double* c, inc_t rsc, inc_t csc );
...
```

Fortran

```fortran
interface
  ...
  subroutine bli_dgemm(transa,transb,m,n,k,alpha,a,rsa,csa,b,rsb,csb,&
                       beta,c,rsc,csc) bind(c,name="bli_dgemm")
    use, intrinsic :: iso_c_binding, only: c_double
    use bli_kinds, only: trans_t, dim_t, inc_t
    integer(trans_t), value :: transa,transb
    integer(dim_t), value :: m,n,k
    real(c_double), intent(in) :: alpha,beta
    integer(inc_t), value :: rsa,csa
    real(c_double), intent(in) :: a(csa,*)
    ...
  end subroutine
end interface
```

BLIS

Level-1v

amaxv
addv
axpyv
scal2v
axpybv
dotxv
subv
copyv
swapv
dotv
invertv
setv
xpbyv
scalv
invscalv

Level-2

gemv
ger
her
hemv
symv
syr2
syr
trsv
trmv

Level-3

gemm
hemm
herk2k
herk
symm
syr2k
syrk
trmm
trmm3
trsm

https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md

BLIS

Level-1v

amaxv
addv
axpyv
scal2v
axpybv
dotxv
subv
copyv
swapv
dotv
setv
invertv
xpbyv
scalv
invscalv

Level-2

gemv
ger
her
hemv
symv
syr2
syr
trsv
trmv

Level-3

gemm
herk2k
hemm
herk
symm
syr2k
syrk
trmm
trmm3
trsm

Level-1f

axpy2v
axpyf
dotaxpyv
dotxaxpyf
dotxf

Level-1d

addd
copyd
axpyd
invertd
invscald
scal2d
scald
setd
shiftd
setid
subd
xpbyd

Level-1m

addm
axpym
copym
scal2m
setm
subm
invscalm

https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md

BLIS

Level-1v

amaxv
addv axpyv scal2v
axpybv dotxv subv
copyv swapv
dotv setv
invertv
xpbyv
scalv
invscalv

Level-2

gemv
ger her
hemv
symv syr2
syr
trsv
trmv

Level-3

gemm
hemm herk2k
herk
symm syr2k
syrk
trmm trmm3
trsm

Level-1f

axpy2v
axpyf
dotaxpyv
dotxaxpyf
dotxf

Level-1d

addd
copyd
axpyd
invertd
invscald
scal2d
scald
setd shiftd
setid
subd
xpbyd

Level-1m

addm
axpym
copym
scal2m
setm
subm
invscalm

× {s, d, c, z}

https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md

BLIS

Level-1v

amaxv
addv
axpyv
scal2v
axpybv
dotxv
subv
copyv
swapv
dotv
setv
invertv
xpbyv
scalv
invscalv

Level-2

gemv
ger
her
hemv
symv
syr2
syr
trsv
trmv

Level-3

gemm
herk2k
hemm
herk
symm
syr2k
syrk
trmm
trmm3
trsm

Level-1f

axpy2v
axpyf
dotaxpyv
dotxaxpyf
dotxf

Level-1d

addd
copyd
axpyd
invertd
invscald
scal2d
scald
setd
shiftd
setid
subd
xpbyd

Level-1m

addm
axpym
copym
scal2m
setm
subm
invscalm

× {s, d, c, z}
× {basic, expert}

https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md

#include <blis.h>

- 34 kLOC
- heavy use of macros (templates)

```
__attribute__ ((visibility ("default"))) void bli_sgemm ( trans_t transa, trans_t transb, dim_t m, dim_t
      n, dim_t k, const float* alpha, const float* a, inc_t rs_a, inc_t cs_a, const float* b, inc_t
     rs_b, inc_t cs_b, const float* beta, float* c, inc_t rs_c, inc_t cs_c ); __attribute__ ((
     visibility ("default"))) void bli_dgemm ( trans_t transa, trans_t transb, dim_t m, dim_t n, dim_t
     k, const double* alpha, const double* a, inc_t rs_a, inc_t cs_a, const double* b, inc_t rs_b,
     inc_t cs_b, const double* beta, double* c, inc_t rs_c, inc_t cs_c ); __attribute__ ((visibility ("
     default"))) void bli_cgemm ( trans_t transa, trans_t transb, dim_t m, dim_t n, dim_t k, const
     scomplex* alpha, const scomplex* a, inc_t rs_a, inc_t cs_a, const scomplex* b, inc_t rs_b, inc_t
     cs_b, const scomplex* beta, scomplex* c, inc_t rs_c, inc_t cs_c ); __attribute__ ((visibility ("
     default"))) void bli_zgemm ( trans_t transa, trans_t transb, dim_t m, dim_t n, dim_t k, const
     dcomplex* alpha, const dcomplex* a, inc_t rs_a, inc_t cs_a, const dcomplex* b, inc_t rs_b, inc_t
     cs_b, const dcomplex* beta, dcomplex* c, inc_t rs_c, inc_t cs_c );
__attribute__ ((visibility ("default"))) void bli_sgemmt ( uplo_t uploc, trans_t transa, trans_t transb,
      dim_t m, dim_t k, const float* alpha, const float* a, inc_t rs_a, inc_t cs_a, const float* b,
     inc_t rs_b, inc_t cs_b, const float* beta, float* c, inc_t rs_c, inc_t cs_c ); __attribute__ ((
     visibility ("default"))) void bli_dgemmt ( uplo_t uploc, trans_t transa, trans_t transb, dim_t m,
     dim_t k, const double* alpha, const double* a, inc_t rs_a, inc_t cs_a, const double* b, inc_t rs_b
     , inc_t cs_b, const double* beta, double* c, inc_t rs_c, inc_t cs_c ); __attribute__ ((visibility
     ("default"))) void bli_cgemmt ( uplo_t uploc, trans_t transa, trans_t transb, dim_t m, dim_t k,
     const scomplex* alpha, const scomplex* a, inc_t rs_a, inc_t cs_a, const scomplex* b, inc_t rs_b,
     inc_t cs_b, const scomplex* beta, scomplex* c, inc_t rs_c, inc_t cs_c ); __attribute__ ((
     visibility ("default"))) void bli_zgemmt ( uplo_t uploc, trans_t transa, trans_t transb, dim_t m,
     dim_t k, const dcomplex* alpha, const dcomplex* a, inc_t rs_a, inc_t cs_a, const dcomplex* b,
     inc_t rs_b, inc_t cs_b, const dcomplex* beta, dcomplex* c, inc_t rs_c, inc_t cs_c );
```

Semantic Matching and Patching Engine

Pattern matching

```
/* blis.cocci */
@match_void@
identifier F =~ "bli_.*";
parameter list PL;
@@

  void F( PL );



/* continues on next page */
```

```
/* ... blis.h ... */

__attribute__ ((visibility ("
    default"))) void bli_dgemm
    ( trans_t transa, trans_t
    transb, dim_t m, dim_t n,
    dim_t k, const float* alpha
    , const float* a, inc_t
    rs_a, inc_t cs_a, const
    float* b, inc_t rs_b, inc_t
     cs_b, const float* beta,
    float* c, inc_t rs_c, inc_t
     cs_c );

/* ... */
```

Binding generation

```
@script: python@
pl << match.PL;
proc_name << match.F;
@@

def convert_to_fortran_args(param_list):
    # ...
    return args, arg_stmts

args, arg_stmts = convert_to_fortran_args(pl)

print(f"""
interface
  subroutine {proc_name}({args}) bind(c,name="{name}")
    {arg_stmts}
  end subroutine
end interface
""")
```

Matching and generating string functions

```
@match_string@
identifier F =~ "bli_.*";
parameter list PL;
@@

  char* F( PL );

@script: python@
pl << match_string.PL;
f << match_string.F;
@@

# ... generate wrapper ...
```

Limitations

- `union`
- callback functions
- missing semantic information

Summary

- automate large-scale code transformations

Full code for this presentation

https://github.com/ivan-pi/blis-fortran

Acknowledgements

- dealii-X
- SiVeGCS

Learn Coccinelle

Coccinelle Cheat Sheet ⇒

https://doi.org/10.5281/zenodo.14728558

Coccinelle Tutorial

1-day tutorial slides:

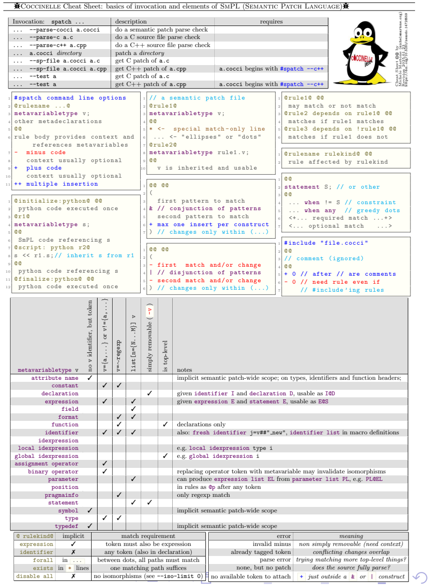
https://doi.org/10.5281/zenodo.14728519

Stay tuned for 2025 trainings:

https://tiny.badw.de/zc5D9e

Coccinelle Website

https://coccinelle.gitlabpages.inria.fr/website/



Coccinelle Cheat Sheet: basics of invocation and elements of SmPL (Semantic Patch Language)

Invocation: `spatch ...`	description	requires
`... --parse-cocci a.cocci`	do a semantic patch parse check	
`... --parse-c a.c`	do a C source file parse check	
`... --parse-c++ a.cpp`	do a C++ source file parse check	
`... a.cocci` *directory*	patch a *directory*	
`... --sp-file a.cocci a.c`	get C patch of `a.c`	
`... --sp-file a.cocci a.cpp`	get C++ patch of `a.cpp`	`a.cocci` begins with `#spatch --c++`
`... --test a`	get C patch of `a.c`	
`... --test a`	get C++ patch of `a.cpp`	`a.cocci` begins with `#spatch --c++`

Cheat Sheet 🅭🅯 by Michele Martone (michelemartone.org) Revision 2025.01.23 https://doi.org/10.5281/zenodo.14728559

```
#spatch command line options
@rulename ...@
metavariabletype v;
other metadeclarations
@@
rule body provides context and
     references metavariables
-  minus code
   context usually optional
+  plus code
   context usually optional
++ multiple insertion
```

```
@initialize:python@ @@
 python code executed once
@r1@
metavariabletype s;
@@
 SmPL code referencing s
@script: python r2@
s << r1.s;// inherit s from r1
@@
 python code referencing s
@finalize:python@ @@
 python code executed once
```

```
// a semantic patch file
@rule1@
metavariabletype v;
@@
* <-  special match-only line
 ... <- "ellipses" or "dots"
@rule2@
metavariabletype rule1.v;
@@
  v is inherited and usable
```

```
@@ @@
(
  first pattern to match
& // conjunction of patterns
  second pattern to match
+ max one insert per construct
) // changes only within (...)
```

```
@@ @@
(
- first  match and/or change
| // disjunction of patterns
- second match and/or change
) // changes only within (...)
```

```
@rule1@ @@
 may match or not match
@rule2 depends on rule1@ @@
 matches if rule1 matches
@rule3 depends on !rule1@ @@
 matches if rule1 does not
```

```
@rulename rulekind@ @@
 rule affected by rulekind
```

```
@@
statement S; // or other
@@
 ... when != S // constraint
 ... when any  // greedy dots
 <+... required match ...+>
 <... optional match   ...>
```

```
#include "file.cocci"
@@
// comment (ignored)
@@
+ 0 // after // are comments
- 0 // need rule even if
    // #include'ing rules
```

metavariabletype v	no v identifier, but token	v={a,...} or v!={a,...}	v=~regexp	list [n={N...M}] v	simply removable (-v)	is top-level	notes
attribute name	✓						implicit semantic patch-wide scope; on types, identifiers and function headers;
constant		✓	✓				
declaration					✓		given `identifier I` and `declaration D`, usable as `I@D`
expression		✓		✓			given `expression E` and `statement E`, usable as `E@S`
field				✓			
format			✓	✓			
function			✓			✓	declarations only
identifier		✓	✓	✓			also: `fresh identifier j=v##"_new"`, `identifier list` in macro definitions
idexpression							
local idexpression							e.g. `local idexpression type i`
global idexpression						✓	e.g. `global idexpression i`
assignment operator		✓					
binary operator		✓					replacing operator token with metavariable may invalidate isomorphisms
parameter				✓			can produce `expression list EL` from `parameter list PL`, e.g. `PL@EL`
position							in rules as `@p` after any token
pragmainfo			✓				only regexp match
statement				✓	✓		
symbol	✓						implicit semantic patch-wide scope
type		✓	✓				
typedef	✓						implicit semantic patch-wide scope

@ rulekind@	implicit	match requirement
expression	✓	token must also be expression
identifier	✗	any token (also in declaration)
forall	in `...`	between dots, all paths must match
exists	in `*` lines	one matching path suffices
disable all	✗	no isomorphisms (see `--iso-limit 0`)

error	*meaning*
invalid minus	*non simply removable (need context)*
already tagged token	*conflicting changes overlap*
parse error	*trying matching more top-level things?*
none, but no patch	*does the source fully parse?*
no available token to attach	`+` *just outside a* `&` *or* `|` *construct*