# Introduction to FORM

Ben Ruijl

July 17 - 19, 2023

Ruijl Research

# A modern look at FORM

# The bad parts of FORM I

- Counterintuitive control flow
- Text-based preprocessor is used for logic
- Hard to act at the expression level due to implicit loop over terms

```
#do i=1,5
    .sort
    #do j=1,`i'
        L F`j'`i' = x`j'+x^2;
        #write "test2"
    #enddo
    Print "%t";
    #write "test3"
#enddo
```

# The bad parts of FORM II

- No namespaces
- Term length limitations (`MaxTermSize` and company)
- Lack of data structures, hashmaps etc have to be emulated
- Often workarounds required that one ``needs to know''
- No native factorisation: $(x+1)(x+2)$ will be expanded
- Bugs that will never be fixed
- IO to other languages is not great
- Not possible to use as a library

# The bad parts of FORM III

- Code written when compilers and system allocators could not be trusted
- Mixing of logic and expression representation
- Pointer and offset hacks that are no longer needed?

# Symbolica

- Symbolica is a new computer algebra system
- Focus on flexibility and ease of use in existing projects
- Should be easy to pass data to and from FORM, Mathematica, etc.
- Open development on Zulip and Github
- Blog posts and documentation on `https://symbolica.io`
- Community supported

# Pattern matching I

- FORM pattern matcher has shortcomings and inconsistencies
- `id p1?.p2?*f(p1?.p2?) = 1;` may not match
- `id f(?a,f(?b,?a,?c),?d) = f(?a,f(?b,?c),?d);` may not match
- Not possible to match subset of factors or summands with `?a`
- `id f(x?)*x? = 1;` does not match to `x*y*f(x*y)` even though it matches `x*y` in the function argument!
- Iterate through all matches without replacement (Mathemetica cannot do this either)
- Should be like `regex` in Python: separate matching and replacement
- Should match at any level

# Pattern matching II

- Internally there is only one wildcard type, `x_`, that can match *any* subexpression
- `id x_ = 1` applied to `x*y*z` gives `1`
- `id x_*y_ = f(x_,y_)` applied to `x*y*z` gives all bipartitions
- `id x = 5` applied to `f(x)` gives `f(5)`

Matching `z*x_*y_*f(z_,x_,w_)` to `x*y*z*w*f(x,y,x*y,z)` gives:

```
x_ = y,      y_ = w,      z_ = x,      w_ = (x*y, z)
x_ = y,      y_ = x * w,  z_ = x,      w_ = (x*y, z)
x_ = x * y,  y_ = w,      z_ = (x, y), w_ = z
```

# Pattern matching II

- Internally there is only one wildcard type, `x_`, that can match *any* subexpression
- `id x_ = 1` applied to `x*y*z` gives `1`
- `id x_*y_ = f(x_,y_)` applied to `x*y*z` gives all bipartitions
- `id x = 5` applied to `f(x)` gives `f(5)`

Matching `z*x_*y_*f(z_,x_,w_)` to `x*y*z*w*f(x,y,x*y,z)` gives:

```
x_ = y,      y_ = w,      z_ = x,      w_ = (x*y, z)
x_ = y,      y_ = x * w,  z_ = x,      w_ = (x*y, z)
x_ = x * y,  y_ = w,      z_ = (x, y), w_ = z
```

# Python API

- Python API is easy to install: `pip install symbolica`
- Language features such as `dict` provide new ways to code

```
from symbolica import Expression

# create a Symbolica expression
x, y, w1_, w2_ = Expression.vars('x','y','w1_','w2_')
f = Expression.fun('f')
e = f(3, x)*y**2 + 5 + Expression.parse('x^2+6')

# replace f(w1_,w2_) with f(w-1,w2^2) in e
# where w1_>=0 and w2_ is a variable
r = e.replace_all(f(w1_,w2_), f(w1_ - 1, w2_**2),
      (w1_ >= 1) & w2_.is_var())
print('Replaced:', r)
```

# Python API II

- Symbolica's polynomial arithmetic is often much faster than competitors'
- Work on integration into FIRE and KIRA underway

```
a = parse('(1+3*x1+5*x2+7*x3+9*x4+11*x5+13*x6+15*x7)^7-1')
b = parse('(1-3*x1-5*x2-7*x3+9*x4-11*x5-13*x6+15*x7)^7+1')
g = parse('(1+3*x1+5*x2+7*x3+9*x4+11*x5+13*x6-15*x7)^7+3')
ag = a * g
bg = b * g
ag.gcd(bg)
```

| Optimizer | Time |
|---|---|
| Fermat | 1241 |
| FORM | 82 |
| Mathematica | 84 |
| Symbolica | 4.4 |

# Python API II

- Symbolica's polynomial arithmetic is often much faster than competitors'
- Work on integration into FIRE and KIRA underway

```
a = parse('(1+3*x1+5*x2+7*x3+9*x4+11*x5+13*x6+15*x7)^7-1')
b = parse('(1-3*x1-5*x2-7*x3+9*x4-11*x5-13*x6+15*x7)^7+1')
g = parse('(1+3*x1+5*x2+7*x3+9*x4+11*x5+13*x6-15*x7)^7+3')
ag = a * g
bg = b * g
ag.gcd(bg)
```

| Optimizer | Time |
|---|---|
| Fermat | 1241 |
| FORM | 82 |
| Mathematica | 84 |
| Symbolica | 4.4 |

# Preprocessor

- Python is the preprocessor!
- Example trace algorithm in FORM vs Symbolica:

```
#N = 10
S p1,...,p`N';
CF f, d(s);

L F = f(p1,...,p`N');

#do l = `N',1,-1
    #if `l' % 2 == 1
        id f(p1?,...,p`l'?) = 0;
    #else
        id f(p1?,...,p`l'?) = sum_(k,1,`l',(-1)^(k+1) *
            d_(p1,p`k')*f(p2,...,p{`k'-1},p{`k'+1},...,p`l'));
    #endif
#enddo
id f = 4;
```

# Preprocessor

- Python is the preprocessor!
- Example trace algorithm in FORM vs Symbolica:

```
from symbolica import Expression
N = 10
p = Expression.vars(*['p' + str(i + 1) for i in range(N)])
p_ = Expression.vars(*['p' + str(i + 1) + '_' for i in range(N)])
f = Expression.fun('f')
d = Expression.fun('d', is_symmetric=True)

e = f(*p[:5]) # = f(p1,p2,p3,p4,p5)

for l in range(N,1,-1):
    if l % 2 == 1:
        e = e.replace_all(f(*p_[:l]), 0)
    else:
        e = e.replace_all(f(*p_[:l]), sum((-1)**(k+1) *
            d(p_[0],p_[k]) * f(*p_[1:k], *p_[k+1:l])
                    for k in range(1,l))).expand()
e = e.replace_all(f(), 4)
print(e)
```

# Computational graph

- Operations per term (like in FORM) can be executed with a `map`:

```
from symbolica import Expression, Transformer
x, x_ = Expression.vars('x', 'x_')
e = (1+x)**2
r = e.map(Transformer().expand().replace_all(x_**2, 6))
print(r)
```

# The future of Symbolica

- Goal: community funding through university licenses
- Funding will be used for FORM maintenance as well
- Let me know if you need certain features!

Join development on:

- `https://symbolica.io`
- `https:/reform.zulipchat.com`
- `https://github.com/benruijl/symbolica`