Star - A Modern Programming Language

And a splash of logic
Change is the norm

Real software engineering is about change, and responding to it. However, our main tools for software - programming languages - are not very good at supporting change.

Change is hard:

A typical programmer's lifetime's output is about 500K loc.

A moderately sized system.

Impossible to migrate.
1. Three ‘R’s

➔ **Re-use**
New roles for old software

➔ **Re-purpose**
React to changing circumstances

➔ **Refactor**
Improvement

Huh?

Compilers can generate better code than humans.

So, why do we try to compete?
A tale of three loops

Ok?
Let's start simple.
This looks easy...

```java
int total = 0;
for (Integer ix : L) {
    total += ix;
}
```

- we had to fix on the type of the number being totaled;
- we had to know about Java’s boxed v.s. unboxed types;
- we had to construct an explicit loop, with the result that we sequentialized the process of adding up the numbers.

Well?

Still better than C-style for loop.
Functional Programming

Mathematical properties of functions

Tractable reasoning

Very expressive
Functions are better?

let{
  total(nil) => 0.
  total(cons(E,L)) => total(L)+E
} in total(L)

- We had to fix on the type of the collection;
- Explicit about state management; and
- Explicit about order of processing
Contracts

```
contract arith[e] ::= {
    (+): (e,e)=>e.
    (*): (e,e)=>e.
    ...
    zero:e.
    one:e.
}
```

- Separate specification from implementation
- Can implement contracts for types you do not own.

Wow!

Arithmetic is not special in Star
Stream total

let{

  total: all c, e ~~

  arith[e], stream[c->>e] |:

    (c) => e.

  total([]) => zero.

  total([E, ..Ls]) => total(Ls) + E.

} in total(L)
Better by folding

leftFold( (+), 0, L)

- No explicit iteration
- Still rely on arithmetic and stream abstractions
- Order dependent on implementation of leftFold

See!
Recursion is buried in standard function
Even better with Logic

{ fold $X$ with (+) | $X$ in $L$ }

- No commitment to ordering
- No commitment to types

Deep!

Declarative specification of desired computation
Grandparents

```java
for(Pair<Person,Person> P:parents){
    for(Pair<Person,Person> Q:parents){
        if(P.child==Q.parent)
            emit(P.parent,Q.child)
    }
}
```

- Search -> satisfaction semantics
- `emit` hides some nastinesses

Look:
Java can be verbose
Folding Grandparents

foldLeft(
    (SoFar,(X,Z)) => foldLeft(
        let {
            acc(gp1,(ZZ,Y)) where Z==ZZ => [gp1..,(X,Y)].
            acc(gp1,_) => gp1.
        } in acc,
        SoFar,parents),
        []
        parents)

- Fold expressions have a tendency to explode when there is any complication
- Uses some powerful features:
  a. Tuple patterns
  b. let expressions
  c. Conditional equations
Grandparents’ Logic

\{ (GP,GC) \mid (GP,P) \text{ in parents} \land (P,GC) \text{ in parents} \}

Wow!
Similar in spirit to list abstractions

- Easy to extend with different connectives
- Easy to integrate ‘logic’ semantics with more conventional programming
Don’t need rules

Using functions to define queries is a very powerful technique. No need for a separate syntax of ‘logic’ rule.

```prolog
yourGPs(GC) => { GP | (GP,P) in parents && (P,GC) in parents }
```

Semantics is not the same as Prolog!
2. Modules

We need to be able to program in the large as well as in the small

➔ **What**
   A module is simply a record with functions

➔ **But**
   Modules often expose types as well as functions

➔ **And so**
   We can systematize many engineering patterns
In the large

Modules, packages, libraries

Few languages make much effort into *computing* libraries.

**Huh?**

Are modules ways of breaking up large programs?

Or are large programs constructed from smaller pieces.
Records of functions

{ find:all k,v ~~ (dict[k,v],k)=>option[v].
  update:all k,v~~(dict[k,v],k,v)=>dict[k,v].
}

- Record contains functions.
- Full power of language available at large scale
- But, ...
Modules export types

Embed types in records

Existentially quantified types
Existential Dictionaries

dictModTp ::= exists dict/2 ~~ {  
  find:all k,v ~~ (dict[k,v],k)=>option[v].  
  update:all k,v~~(dict[k,v],k,v)=>dict[k,v].  
  new:all k,v ~~ ()=>dict[k,v].  
}
Dictionary Defined

MyDictM:() => dictModTp.

MyDictM() => {
  dict[k,v] <-- fancyMap[k,v].
  find(D,K) => findInFancyMap(D,K).
  update(D,K,V) => updateFancyMap(D,K,V).
  new() => ...
}

- Existentials need evidence
- Can be different inside vs outside
Using modules

MD = MyDictM()
D: MD.dict[string,integer].
D.find("fred")

- Split module import into separate architectural elements
- Can write module-valued functions

See...
The type of D depends on a dynamically computed expression
A platform is a foundation where others can develop and promulgate solutions.
3. Platforms

A platform needs

➔ **Construction**
  Bigger structures from smaller pieces, in a type safe way.

➔ **Marketplace**
  Modules type signature means better reusability

➔ **Frameworks**
  Testing, deployment, scenarios; sometimes independent of modules being used.
A Simple Application

Huh?
Connections are well typed

- Many simple applications look like this
A Complete (Simple) Scenario

- Create an application from the use case

Wow!
The whole application is statically typed
An Abstract Scenario

- Can re-use this template by plugging in different modules for different roles
- Existential types respect ownership

Deep!
We have variables in place of modules
Road map

November 2018
- Bootstrap compiler
- Interpreter
- Core Run-time

2018

Spring 2019
- Self-hosting compiler
  (~25Kloc Star)
- Draft programming guide
- Implement fiber-based threading model
- Speech-actions

2019

2020
- Implement platform
- Reference manual
Takeaways

Star has strong foundations to support ‘good’ architecture

Logic can be very expressive, with a small but critical role

Sound semantics for modules leads to a greater role for PLs in large systems

Conclusions for illustration purposes only
Not a solo effort

Michael Sperber, David Frese, Andreas Bernauer, Steve Banauch, Bob Riemenschneider, Chris Gray, Kevin Twidle, Utkarsh Lath, Keith Clark, Michael Weber and ... you?
Any questions?

For more, go to
https://github.com/fgmccabe/star

In progress...