Using Structured Spreadsheets to Develop Smart Contracts

Colin González

CIFRE Ph.D. Student
Nomadic Labs - Irif, Université de Paris

Under the supervision of

Ralf Treinen (IRIF, Université de Paris) Benjamin Canou (Nomadic Labs)

Adrien Guatto (IRIF, Université de Paris) Yann Régis-Gianas (Nomadic Labs)

2021-11-26

Smart Contracts or Programmable Transactions

Programmable Transactions

- How to make a **secure** transaction **without** a trusted third-party?
- You use a blockchain, a **distributed** and **non forgeable** ledger!
- Transactions are chosen and performed using a **consensus protocol**.

Programmable Transactions

- How to make a **secure** transaction **without** a trusted third-party?
- You use a blockchain, a **distributed** and **non forgeable** ledger!
- Transactions are chosen and performed using a **consensus protocol**.

Bonus: they are **programmable**.

You Said Programmable?

- Yes! We call them **smart contracts**.
- Smart contracts are publicly hosted and executed by the blockchain.
- The code of is not modifiable.
- Hence the contract trait.

How Can End-Users Develop Smart Contracts?

A Simple Accounting Task

- Our accountant Bill,
  wants to develop, deploy and monitor a smart contract.

Informal spec

This contract **collects deposits**; **only** Alice can withdraw the collected coins.

- Bill can model it using a spreadsheet.

The Spreadsheet

- Two inputs, user and deposit
- A state to compute the collected amount.
- One output, the operations to commit.
- Bill pulls down the last line to fill the spreadsheet.

	A	B	C	D
1	user	deposit	collected	operations
2			=IF(A2 = "Alice", 0, B2)	=IF(A2 = "Alice", SEND("Alice", B2), EMPTY())
3			=IF(A3 = "Alice", 0, B3 + C2)	=IF(A3 = "Alice", SEND("Alice", B3+C3), EMPTY())

Bill's Spreadsheet

The Spreadsheet

- Two inputs, **user** and **deposit**
- A state to compute the collected amount.
- One output, the operations to commit.
- Bill pulls down the last line to fill the spreadsheet.

	A	B	C	D
1	**user**	**deposit**	collected	operations
2			=IF(A2 = “Alice”, 0, B2)	=IF(A2 = “Alice”, SEND(“Alice”, B2), EMPTY())
3			=IF(A3 = “Alice”, 0, B3 + C2)	=IF(A3 = “Alice”, SEND(“Alice”, B3+C3), EMPTY())

Bill’s Spreadsheet

The Spreadsheet

- Two inputs, **user** and **deposit**
- A state to compute the **collected** amount.
- One output, the operations to commit.
- Bill pulls down the last line to fill the spreadsheet.

	A	B	C	D
1	**user**	**deposit**	**collected**	operations
2			=IF(A2 = “Alice”, 0, B2)	=IF(A2 = “Alice”, SEND(“Alice”, B2), EMPTY())
3			=IF(A3 = “Alice”, 0, B3 + C2)	=IF(A3 = “Alice”, SEND(“Alice”, B3+C3), EMPTY())

Bill’s Spreadsheet

The Spreadsheet

- Two inputs, **user** and **deposit**
- A state to compute the **collected** amount.
- One output, the **operations** to commit.
- Bill pulls down the last line to fill the spreadsheet.

	A	B	C	D
1	**user**	**deposit**	**collected**	**operations**
2			=IF(A2 = “Alice”, 0, B2)	=IF(A2 = “Alice”, SEND(“Alice”, B2), EMPTY())
3			=IF(A3 = “Alice”, 0, B3 + C2)	=IF(A3 = “Alice”, SEND(“Alice”, B3+C3), EMPTY())

Bill’s Spreadsheet

This What Bill's Contract Looks Like

```
{ parameter unit ;
  storage address ;
  code { CDR ;
         SENDER ;
         CONTRACT unit ;
         IF_NONE
             { NIL operation ; PAIR }
             { SWAP ;
               DUP ;
               DUG 2 ;
               SENDER ;
               COMPARE ;
               EQ ;
               IF { SWAP ;
                    NIL operation ;
                    DIG 2 ;
                    BALANCE ;
                    PUSH unit Unit ;
                    TRANSFER_TOKENS ;
                    CONS ;
                    PAIR }
                  { DROP ; NIL operation ; PAIR } } } }
```

Bill's Contract in Michelson

The High-Level Version

```
type storage = address

type parameter = unit

let result (op: operation list)
           : operation list * storage = (op, alice)

let main ((), alice : parameter * storage) :
                           operation list * storage =
let some_contract : unit contract option  =
Tezos.get_contract_opt Tezos.sender in
match some_contract with
 | Some(sender_contract) ->
   if sender = alice then
     result [Tezos.transaction
              () Tezos.balance  sender_contract]
   else
     result ([]:operation list)
 | None ->
     result ([]:operation list)
```

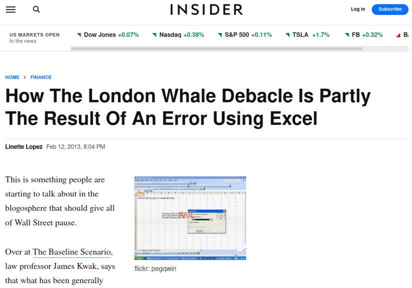
Bill's Contract in Ligo

Spreadsheets Are Easy To Use

- Spreadsheets is the most used computation platform.
- With the correct formulas, users can program autonomously.
- But, accessiblity may be an illusion.

Spreadsheets Programming is Error Prone

- Until very recently, there where no tools for static analysis.
- In 2012, JP Morgan lost about 6 billion dollars because of a spreadsheet mistake.
- In 2010, the US federal budget "The Path to Prosperity" proposal was based on the flawed economic study Growth in a Time of Debt by Reinhart and Rogoff due to a spreadsheet error.



INSIDER

Log In Subscribe

US MARKETS OPEN
In the news

Dow Jones +0.07% Nasdaq +0.38% S&P 500 +0.11% TSLA +1.7% FB +0.32%

HOME > FINANCE

How The London Whale Debacle Is Partly The Result Of An Error Using Excel

Linette Lopez Feb 12, 2013, 8:04 PM

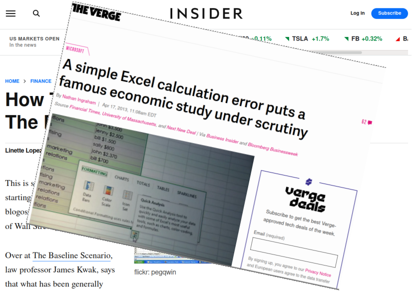
This is something people are starting to talk about in the blogosphere that should give all of Wall Street pause.

Over at The Baseline Scenario, law professor James Kwak, says that what has been generally

flickr: pegqwin

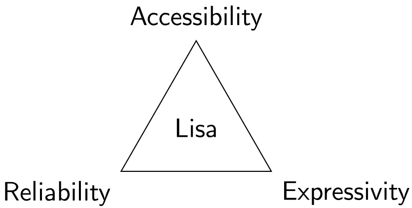
Spreadsheets Programming is Error Prone

- Until very recently, there where no tools for static analysis.
- In 2012, JP Morgan lost about 6 billion dollars because of a spreadsheet mistake.
- In 2010, the US federal budget “The Path to Prosperity” proposal was based on the flawed economic study Growth in a Time of Debt by Reinhart and Rogoff due to a spreadsheet error.



Finding a Good Tool

We want a balance between accessibility, reliability and expressivity.



Question

How to provide end users a smart contract design interface?

This Presentation

How Users Work on Spreadsheets

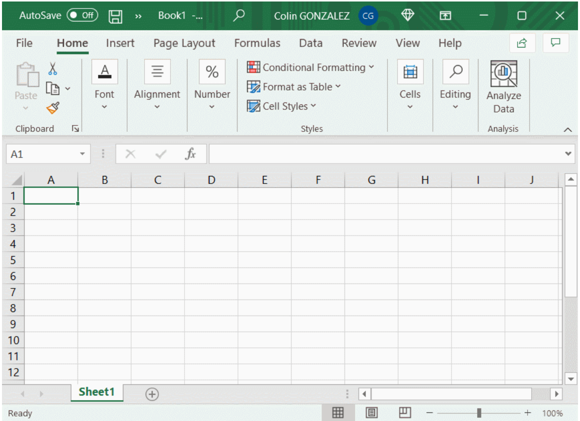
Spreadsheets 101

Core constructs

- Spreadsheets define a matrix of cells.
- Cells can contain data or an expression.
- Expressions may depend on other cells using a coordinate system.

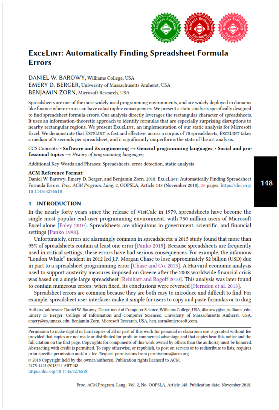
Spreadsheets 101

This what users could do to define natural numbers.



Spreadsheet Facts

- In majority spreadsheets are rectangular and grow vertically.
- Instead of defining each cell, they pull them down.
- Such spreadsheets can be seen as mutually recursive streams.



ExceLint: Automatically Finding Spreadsheet Formula Errors

DANIEL W. BAROWY, Williams College, USA
EMERY D. BERGER, University of Massachusetts Amherst, USA
BENJAMIN ZORN, Microsoft Research, USA

Spreadsheets are one of the most widely used programming environments, and are widely deployed in domains like finance where errors can have catastrophic consequences. We present a static analysis specifically designed to find spreadsheet formula errors. Our analysis directly leverages the rectangular character of spreadsheets. It uses an information-theoretic approach to identify formulas that are especially surprising disruptions to nearby rectangular regions. We present ExceLint, an implementation of our static analysis for Microsoft Excel. We demonstrate that ExceLint is fast and effective: across a corpus of 70 spreadsheets, ExceLint takes a median of 5 seconds per spreadsheet, and it significantly outperforms the state of the art analysis.

CCS Concepts: • **Software and its engineering → General programming languages**; • **Social and professional topics** → *History of programming languages*;

Additional Key Words and Phrases: Spreadsheets, error detection, static analysis

ACM Reference Format:
Daniel W. Barowy, Emery D. Berger, and Benjamin Zorn. 2018. ExceLint: Automatically Finding Spreadsheet Formula Errors. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 148 (November 2018), 26 pages. https://doi.org/10.1145/3276518

148

1 INTRODUCTION

In the nearly forty years since the release of VisiCalc in 1979, spreadsheets have become the single most popular end-user programming environment, with 750 million users of Microsoft Excel alone [Foley 2010]. Spreadsheets are ubiquitous in government, scientific, and financial settings [Panko 1998].

Unfortunately, errors are alarmingly common in spreadsheets: a 2015 study found that more than 95% of spreadsheets contain at least one error [Panko 2015]. Because spreadsheets are frequently used in critical settings, these errors have had serious consequences. For example, the infamous "London Whale" incident in 2012 led J.P. Morgan Chase to lose approximately $2 billion (USD) due in part to a spreadsheet programming error [Chase and Co. 2013]. A Harvard economic analysis used to support austerity measures imposed on Greece after the 2008 worldwide financial crisis was based on a single large spreadsheet [Reinhart and Rogoff 2010]. This analysis was later found to contain numerous errors; when fixed, its conclusions were reversed [Herndon et al. 2013].

Spreadsheet errors are common because they are both easy to introduce and difficult to find. For example, spreadsheet user interfaces make it simple for users to copy and paste formulas or to drag

Authors' addresses: Daniel W. Barowy, Department of Computer Science, Williams College, USA, dbarowy@cs.williams.edu; Emery D. Berger, College of Information and Computer Sciences, University of Massachusetts Amherst, USA, emery@cs.umass.edu; Benjamin Zorn, Microsoft Research, USA, ben.zorn@microsoft.com.

2475-1421/2018/11-ART148
https://doi.org/10.1145/3276518

Daniel W. Barrowy et al., OOPSLA SPLASH'18

Question

Is it possible to formalize structured spreadsheets are **reactive programs**?

A Reactive Interpretation of Spreadsheets

A Correspondence with Streams

In Haskell and Lustre you can have similar definitions for natural numbers.

- In Haskell

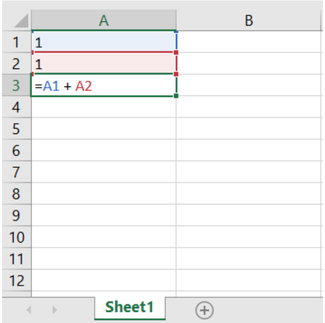
```
nat = 0 : map (+1) nat
```

- In Lustre

```
nat = 0 fby (nat + 1);
```

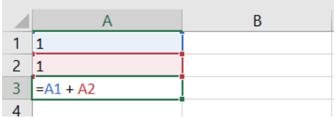
Why a New Tool?

- However a correspondence is not always clear.
- Consider the following definition of Fibonacci.



Spreadsheet Definition of Fibonacci

Corresponding Streams



Spreadsheet Definition of Fibonacci

The correspondence begins to be more difficult explain to Bill.

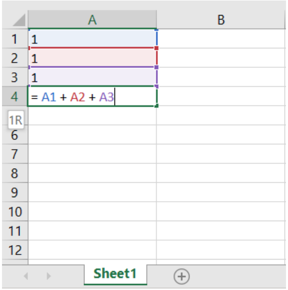
- In Haskell

```
fib = 1 : 1 : zipWith (+) fib (tail fib)
```

- In Lustre

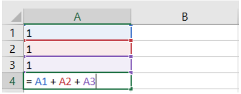
```
fib = 1 fby f;
f   = 1 -> fib + (pre fib);
```

What About This One?



A More Complicated Spreadsheet

Corresponding Streams



A More Complicated Spreadsheet

- In Haskell we have to define a mapping function and use the *right* tail.

```
map3 _ _ _ _ = []
map3 f (x:xs) (y:ys) (z:zs) = f x y z : map3 xs ys zs
plus3 x y z = x + y + z
a = 1 : 1 : 1 : map3 plus3 a (tail a) (tail (tail a))
```

- In Lustre we need more equations to add the *right* number of delay

```
a = 1 fby b;
b = 1 fby c;
c = 1 fby (1 -> b + (pre b) + (pre a));
```

Question

Is the accessiblity of spreadsheets due to explicit coordinates?

About Existing Languages

- Haskell is expressive but lacks of a notion of time.
- Lustre is safe but lacks expressivity and time is implicit.

Questions

1. How to interpret references to cells?
2. How to define the *right* amounts of delay or tails?
3. How can we formalize the semantics of structured sheets?

Lisa an Explicit Time Language

Lisa 101

- At its core Lisa is a typed functional stream-processing language.
- In OCaml this program diverges:

```
let rec from x = x::from (x + 1)
let nat = from 0
```

- In Lisa it doesn't:

```
let rec from =
  fun x ->
    (x :: thunk ((force from) (x+1)))
let nat = from 0
```

How Structured Spreadsheets correspond with Lisa

Correspondence Using the At Operator

- The coordinates are implemented with the At operator.
- It is used to reference values of streams.
- Natural numbers:

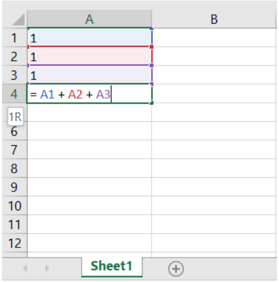
```
(** Syntactic sugar
    x :* y = x :: thunk y **)

rec nat = 0 :*
  rec nat_exp =
   ((force nat)@(fun line -> line-1) + 1)::nat_exp
```

- Fibonacci:

```
rec fib = 1 :* 1 :*
  rec fib_exp =
    (force fib)@(fun line -> line - 1)
    + (force fib)@(fun line -> line - 2)::fib_exp)
```

And the Trickier One



A Complicated Spreadsheet

```
rec a = 1 :* 1 :* 1 :*
    rec a_exp =
       (force a)@(fun line -> line - 1)
       + (force a)@(fun line -> line - 2)
       + (force a)@(fun line -> line - 3)
       :: a_exp
```

Implementation in Lisa

From Spreadsheets to Lisa

- Referencing a cell *n*, is getting the *n*-th value of a column.
- To fetch the *n*-th value of a stream, we use the At operator.
- It **forces** the stream to be at least of length *n*.
- The progression of time is implicit and accessible when needed.
- At, uses the contextual time to compute the desired observation.

Bill's Contract in Lisa

```
(* Prelude *)

let id = fun x -> x

let pred = fun x -> x-1

(* Time observation operations *)
let previous = fun s -> (force s)@pred

let current = fun s -> (force s)@id

(** Syntactic sugar
    x :* y = x :: thunk y **)

(* Blockchain operations *)
let send =
  fun who ->
    fun what -> [(who, what)]

let empty = []
```

```
(* Mutually recursive streams
   for users, deposits, collected coins
   and operations to commit *)

rec user = input "user" :: user

and deposit = input "deposit" :: deposit

and collected =
  (if (current user) = "Alice" then 0
   else (current deposit))
  :* (rec next_collected =
        (if (current user) = "Alice" then 0
         else (current deposit)
              + (previous collected))
        :: next_collected)

and operations =
  (if (current user) = "Alice" then
    send "Alice" (current deposit)
   else empty)
  :* (rec next_operations =
        (if (current user) = "Alice" then
           send "Alice"
             ((current deposit)
             + (previous collected))
         else empty):: next_operations
```

Formalization of Lisa

Lisa's Syntax

t	$::=$	x	Variable
	$\mid$	$\lambda x.t$	Abstraction
	$\mid$	$t\ t$	Application
	$\mid$	$\texttt{force}\ t$	Force
	$\mid$	$\texttt{thunk}\ t$	Thunk
	$\mid$	$t :: t$	Stream Constructor
	$\mid$	$t\texttt{@}t$	Observation
	$\mid$	$(\overline{t})$	Tuples
	$\mid$	$\mu\overline{x}.(\overline{t})$	Recursive Tuple
	$\mid$	$\delta(\overline{t})$	Primitive Application
	$\mid$	c	Constants

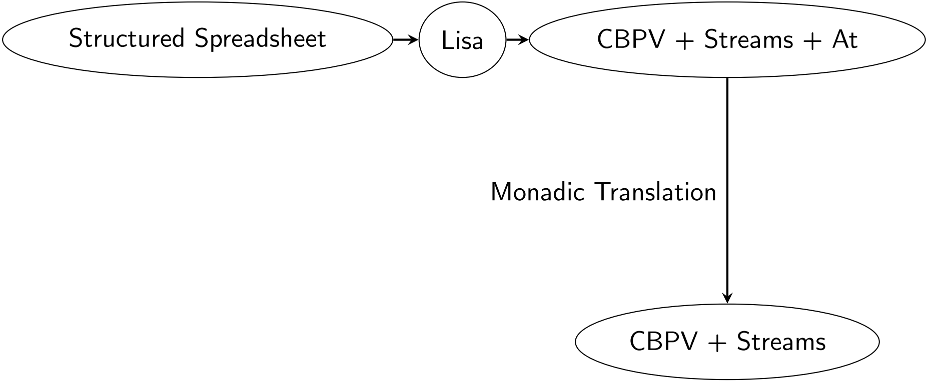
Lisa Syntax

Call-By-Push-Value: Computations Do and Values Are!

- Call-By-Push-Value (CBPV) distinguishes values and computations **syntactically**.
- Computations can be **suspended** as values using *thunks*.
- Thunked computations can be **resumed** by *forcing* them.
- Functions are **computations**, not values.

From Lisa to CBPV

- Lisa is translated to CBPV with Streams and the At operator.
- CBPV with Streams and At is formalized using a monadic translation.
- The monad abstracts the progress of time.



Compilation Scheme

What is Done

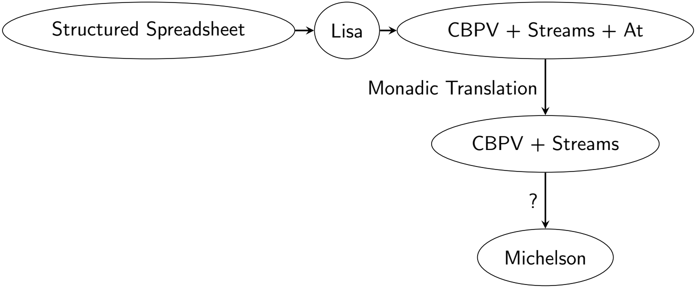
Front-End of an Interpreter for Structured Spreadsheets

- Formalized a spreadsheet system to design smart contracts.
- An implementation of a Lisa interpreter.
- A proof of concept compiler to translate a structured spreadsheet to CBPV.
- Example contracts written as structured spreadsheets.
- Code, examples and work in progress:
  `https://gitlab.com/cg-thesis/lisa`
- Started the formalization of Lisa and its compilation to CPBV.
- Started the correctness proof of the compilation.

Future Work

Generation of Michelson Code

- Clock-directed code generation a la Lustre.
- Incrementalization of the program to get a reactive function by the derivative of the program.



Compilation Scheme

Clock-Directed Code Generation

- The compilation of higher-order reactive languages is still an open question.
- In his thesis, Adrien Guatto studied the usage of integer clocks on compilation of functional reactive programs to digital circuits (e.g. VHDL).
- We want to investigate the possibility to generate an iterative step function.

Compilation by Static Differentiation

- In ESOP'19, Yann Régis-Gianas and his coauthors have shown a novel technique to compute a derivative of a functional program, that is for all function $f : A \to B$:

$$D[\![f]\!] : A \to \Delta A \to \Delta B$$
$$f\,(x \oplus dx) = f\,x \oplus D[\![f]\!]\,x\,dx$$

- A spreadsheet defines a function f from list of calls to list of operations.
- The underlying smart contract may be an incrementalization of f, i.e. $D[\![f]\!]$

Open Questions

- What is the class of contracts that can be naturally encoded as structured spreadsheets?
- The At operator is very expressive but, can we statically verify its proper usage, i.e. only observe past or current, non-cyclic, values ?

Roadmap

- Finish the front-end with a usable proof-of-concept.
- Investigate the two possible low-level code generation.

Questions ?