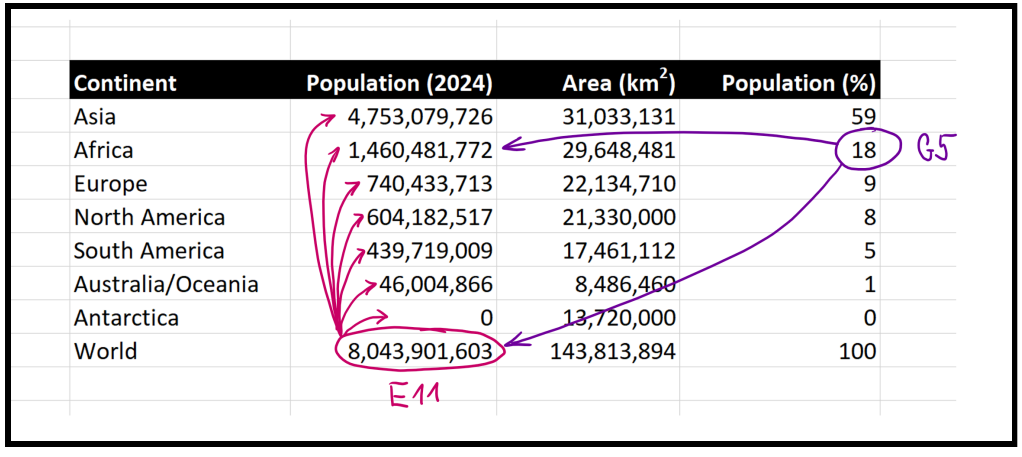
TinyExcel: Tiny spreadsheet system

Architecture and F# events

Tomas Petricek, Charles University

tomas@tomasp.net
@tomasp.net
https://tomasp.net
https://d3s.mff.cuni.cz/teaching/nprg077



Inter-cell dependencies

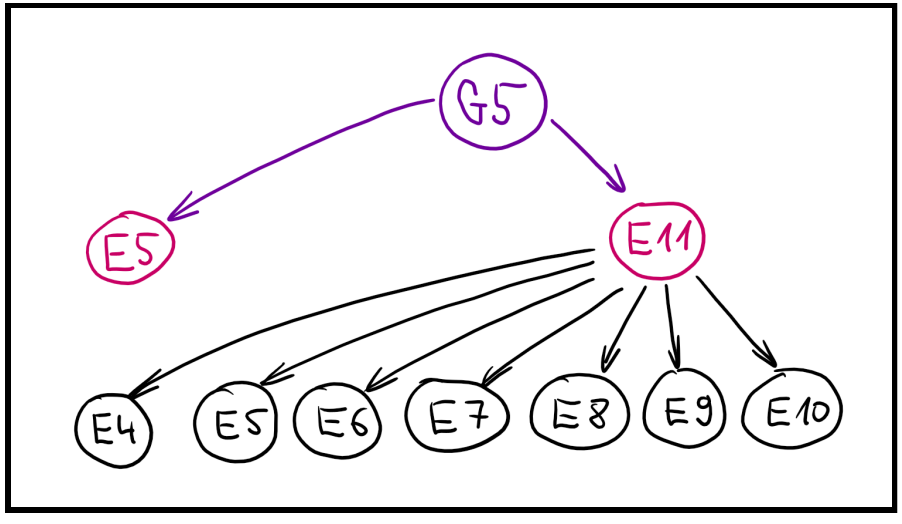
In what order to evaluate sheet?

Avoid evaluating a cell repeatedly!

What to re-evaluate when cells change?

Charles University

Dependency graphs



Dependencies via cell and range references

Cyclic dependencies

Excel does a fixed maximal number of iterations

Explicit or implicit in code

Graph data structure vs. event listeners

Charles University

Reactive programming

Different implementations

- Functional Reactive Programming
- ReactiveX (rxjs, RxJava, Rx.Net)
- Elm software architecture

Implementation techniques

- Push-based - Changes propagated from source
- Pull-based - Update required by the consumer
- Builder-based - Computation to be instantiated

TinyExcel

Implementation techniques


Naive non-cached recursive starting point


Cell is as graph node with "Updated" event


Depending nodes listen, recompute & notify


Tricky error and update handling...

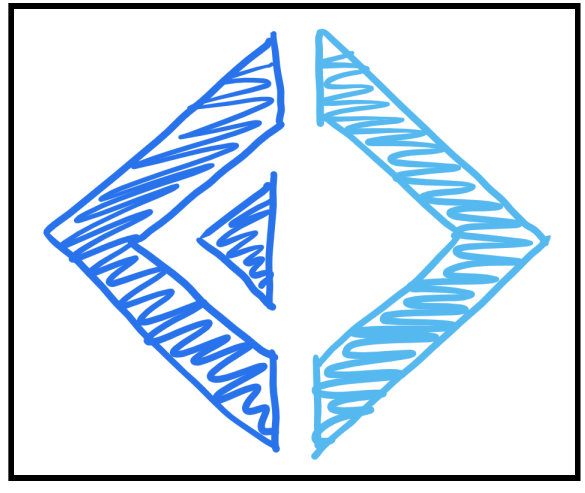
The F# language

What we need for Excel

What we need to write Excel

Event handling

- F# events are objects (values)
- Can trigger & register handlers



More tips & tricks

- Collection processing
- Fancy patterns and active patterns

Finally a user interface?

- Would be nice, but setup costs high...
- Write sheet as HTML document & open

Charles University

Generating lists

List comprehensions with the yield keyword

```
let worldInfo =
  [ yield addr "A1", Const(String "Continent")
    yield addr "B1", Const(String "Population (thousands)")
    for i, (cont, pop) in Seq.indexed continents do
      yield addr ("A"+string(i+2)), Const(String cont)
      yield addr ("B"+string(i+2)), Const(Number pop) ]
```

- `yield` adds another item to the list
- `for` and other constructs to write generators
- `Seq.indexed` trick to get item index

Charles University

Demo

Extending the List module

```
// Decares event value
let evt = Event<int>()

// Trigger event
evt.Trigger(1)
evt.Trigger(2)
evt.Trigger(3)

// Object for listening
evt.Publish

// Listen and print
evt.Publish.Add(fun n ->
  printfn "Got: %d" n)
```

F# Events

Regular F# objects
Not special constructs

Corresond to
`IObservable` in C#

Add and remove
handlers using
`AddHandler` and
`RemoveHandler`

Demo

Working with F# events

Writing and opening HTML files

If you know C#, you can use other options too!

```
let demo () =
  let f = Path.GetTempFileName() + ".html"
  use wr = new StreamWriter(File.OpenWrite(f))
  wr.Write("""<html><body><h1>Hello world!</h1></body></html>""")
  wr.Close()
  Process.Start(f)
```

- `GetTempFileName` gives you a file in TEMP folder
- `use` to make sure stream gets closed on error
- `Process.Start` can (sometimes) open files too