Functional Programming At Work

A personal perspective

Magnus Carlsson | 2012-12-05

My background

- Chalmers Datalogi PhD 1998 – with Thomas Hallgren: **Fudgets**

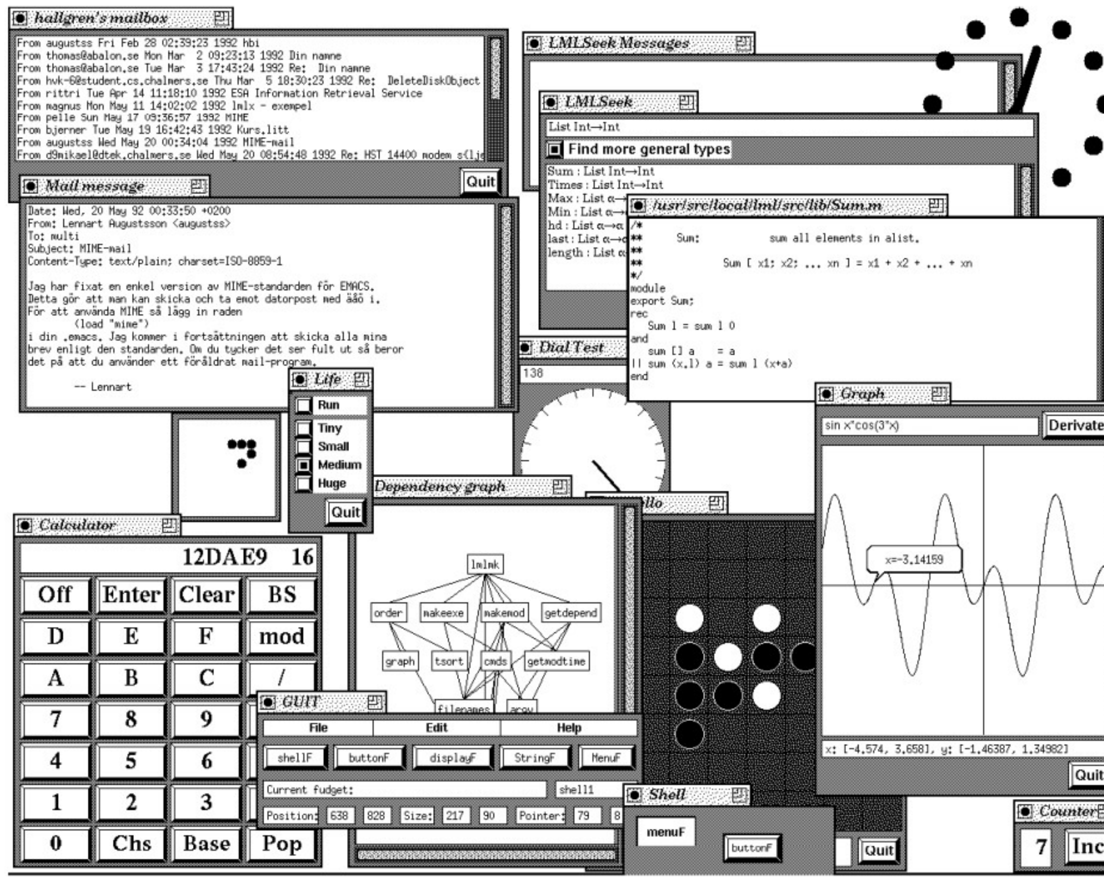


Figure 7: A collage of fudget applications. All windows belong to programs developed with the FUDGETS library.

Fudgets – the counter example (1993)


```
import Fudgets

main = fudlogue (shellF "Up/Down Counter" counterF)

counterF = intDispF >==<
           mapstateF count 0 >==<
           (buttonF "Up" >+< buttonF "Down")

count n (Left Click) = (n+1,[n+1])
count n (Right Click) = (n-1,[n-1])
```

Athena Widget “Hello World” example in C

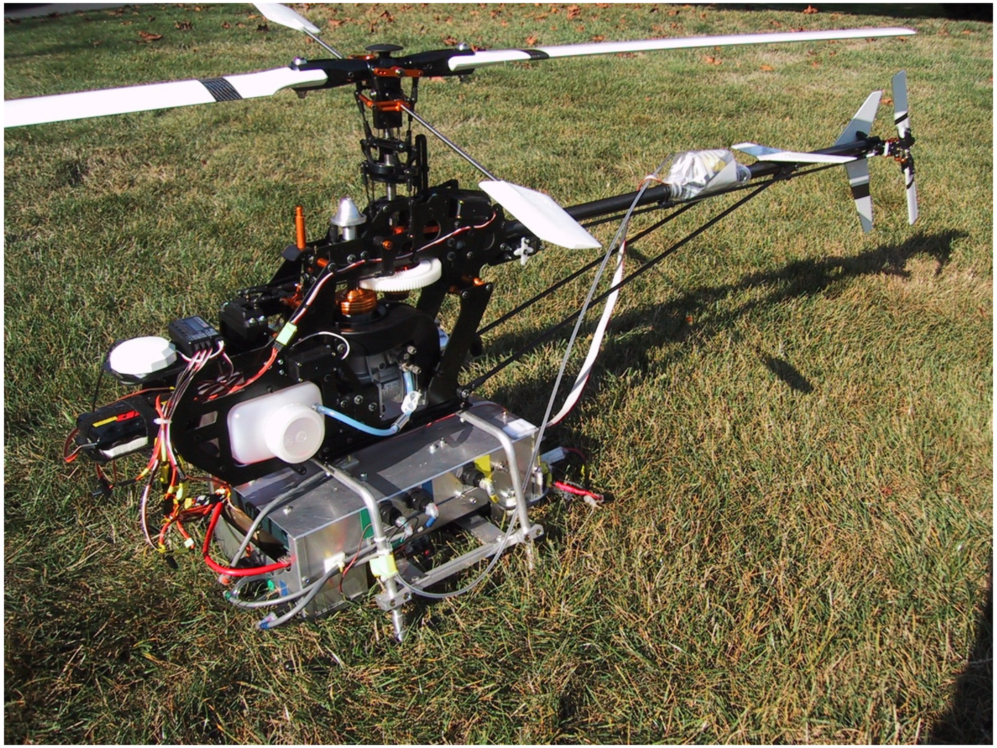
```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Label.h>

main(int argc,char **argv) {
  XtAppContext app_context;
  Widget toplevel,hello;

  toplevel = XtVaAppInitialize(&app_context,"XHello",NULL,0,&argc,argv,NULL,NULL);
  hello = XtVaCreateManagedWidget("Hello World!",labelWidgetClass,toplevel,(void*)0);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
  return 0;
}
```

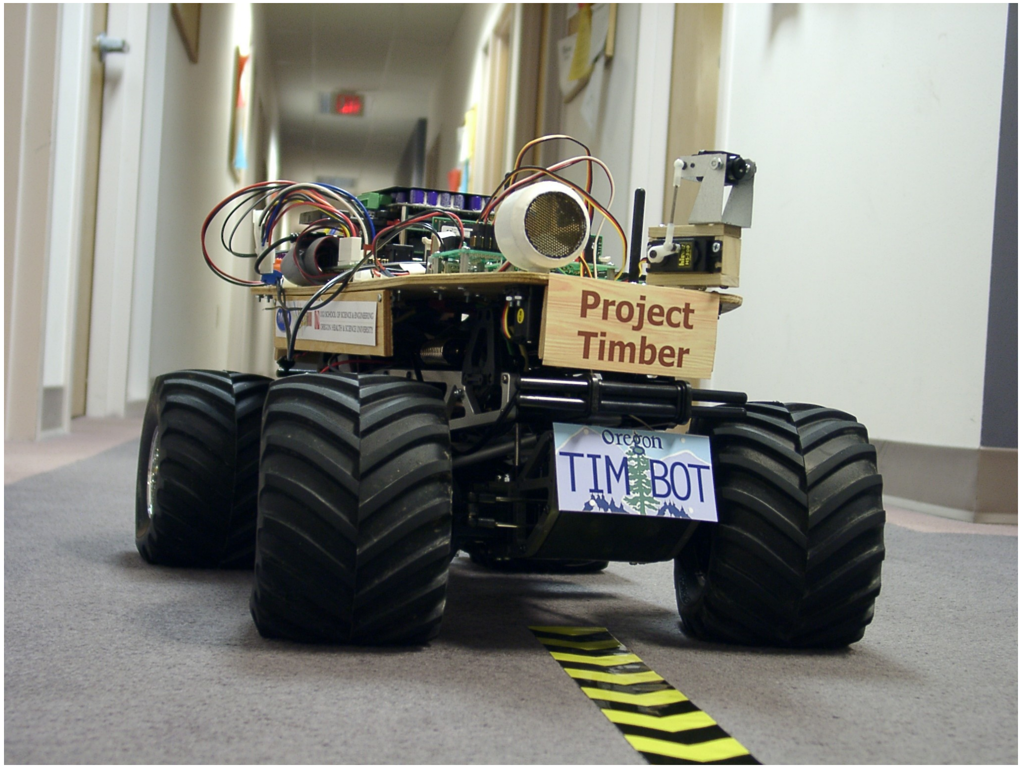
My encounters with Functional Programming

- Oregon Graduate Institute – 2000-2003
  - Haskell-embedded real-time control language → C++
  - Helicopter control - Alex Bogdanov, Geoff Harvey, Dick Kieburtz John Hunt, Erik Wan (also with Antonio Baptista)



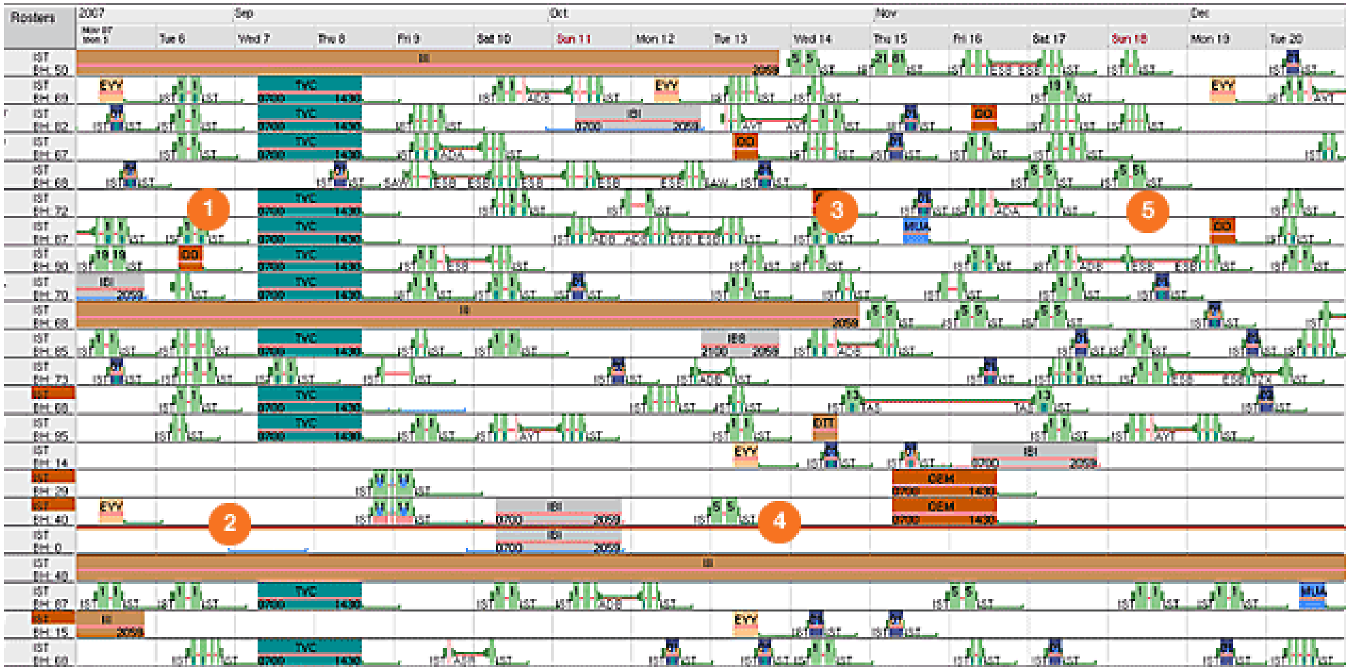
My encounters with Functional Programming

- Oregon Graduate Institute – 2000-2003
  - Timber – specialised OO/FP-language for embedded systems
  - Dick Kieburtz, Mark Jones, Johan Nordlander, Björn von Sydow



My encounters with Functional Programming

- Carmen Systems (Jeppesen today) – 2004-2006
  - Optimize schedules for flight crews, aircraft, trains
  - Dag Wedelin/Erik Andersson (from Chalmers) among founders
  - RAVE – specialized functional language developed for capturing union rules



My encounters with Functional Programming

- Galois, Inc. – 2006-2010
  - Portland, Oregon

My encounters with Functional Programming

- Galois, Inc. – 2006-2010
  - Portland, Oregon – sometimes strange clouds over Mt Hood!

What does Galois do?

- “Designs dependable software to meet mission-critical security and safety challenges in government and industry”
- Building systems that are trustworthy and secure
- Mixture of government and industry clients
- R&D with the favorite tools:
  - Formal methods
  - Typed functional languages
  - Languages, compilers, domain-specific languages
- Kernels, file systems, analysis tools, …
- Haskell for pretty much everything

Benefits of Haskell for Galois (and others)

- Expressive type system & static type checking
  - QuickCheck for testing
  - Documentation
  - Fast integration – one example project:
    - Six engineers
    - 50k lines of code, in 5 components, developed over a number of months
    - Integrated, tested, demo performed in only a week, two months ahead of schedule, significantly above performance spec.
    - 1 space leak, spotted and fixed on first day of testing via the heap profiler
    - 2 bugs found (typos from spec)
  - Find expensive bugs at compile time!

```
map :: (a->b)->[a]->[b]
```

Benefits of Haskell for Galois (and others)

- Good for prototyping new languages
  - Embedded (library-based) domain-specific languages (eDSLs)
  - Examples compiled to C:
    - Feldspar (Mary Sheeran) – signal processing
    - Atom (Tom Hawkins) – hard real-time application compiler
    - Copilot (led by Lee Pike from Galois), based on Atom – adds runtime monitoring to real-time applications

David Monniaux

Atom by Tom Hawkins

- Programmer declares **rules** that fire when conditions become true
- **Compiled to C** when Haskell program is run
- **Scheduler** is also produced
- Example: computing greatest common divisor using Atom rules

```
-- External reference to value A.
let a = word32' "a"

-- External reference to value B.
let b = word32' "b"

-- The external running flag.
let running = bool' "running"

-- A rule to modify A.
atom "a_minus_b" $ do
  cond $ value a >. value b
  a <== value a - value b

-- A rule to modify B.
atom "b_minus_a" $ do
  cond $ value b >. value a
  b <== value b - value a

-- A rule to clear the running flag.
atom "stop" $ do
  cond $ value a ==. value b
  running <== false
```

Atom – generated C code

- Scheduler that fire rules is generated for us

```c
#include <stdbool.h>
#include <stdint.h>


#include <stdlib.h>
#include <stdio.h>
unsigned long int a;
unsigned long int b;
unsigned long int x;
unsigned char running = 1;


static uint64_t __global_clock = 0;
static const uint32_t __coverage_len = 1;
static uint32_t __coverage[1] = {0};
static uint32_t __coverage_index = 0;
struct {  /* state */
  struct {  /* example */
  } example;
} state =
{  /* state */
  {  /* example */

  }
};

/* example.a_minus_b */
static void __r0() {
  uint32_t __0 = b;
  uint32_t __1 = a;
  bool __2 = __0 < __1;
  uint32_t __3 = __1 - __0;
  uint32_t __4 = __2 ? __3 : __1;
  if (__2) {
    __coverage[0] = __coverage[0] | (1 << 0);
  }
  a = __4;
}

/* example.b_minus_a */
static void __r1() {
  uint32_t __0 = a;
  uint32_t __1 = b;
  bool __2 = __0 < __1;
  uint32_t __3 = __1 - __0;
  uint32_t __4 = __2 ? __3 : __1;
  if (__2) {
    __coverage[0] = __coverage[0] | (1 << 1);
  }
  b = __4;
}

/* example.stop */
static void __r2() {
  uint32_t __0 = a;
  uint32_t __1 = b;
  bool __2 = __0 == __1;
  bool __3 = running;
  bool __4 = ! __2;
  bool __5 = __3 && __4;
  if (__2) {
    __coverage[0] = __coverage[0] | (1 << 2);
  }
  running = __5;
}


static void __assertion_checks() {
}


void example() {
  {
    static uint8_t __scheduling_clock = 0;
    if (__scheduling_clock == 0) {
      __assertion_checks(); __r0();  /* example.a_minus_b */
      __assertion_checks(); __r1();  /* example.b_minus_a */
      __assertion_checks(); __r2();  /* example.stop */
      __scheduling_clock = 0;
    }
    else {
      __scheduling_clock = __scheduling_clock - 1;
    }
  }

  __global_clock = __global_clock + 1;

}

int main(int argc, char* argv[]) {
  if (argc < 3) {
    printf("usage: gcd <num1> <num2>\n");
  }
  else {
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    printf("Computing the GCD of %lu and %lu...\n", a, b);
    while(running) {
      example();
      printf("iteration:  a = %lu  b = %lu\n", a, b);
    }
    printf("GCD result: %lu\n", a);
  }
  return 0;
}
```

Domain-specific languages at Galois

- Why domain-specific languages?
  - Small domains enables more opportunities for **verification**, **optimization**, **compilation**
- Example – specifying crypto algorithms in **Cryptol**
  - Started out as embedded DSL in Haskell
  - Now stand-alone DSL with its own compiler and type checker (written in Haskell)

Background – specifying crypto algorithms

- The official specification of AES (“Advanced Encryption Standard”): a mix of English, pseudo code and English

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])            // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                        // See Sec. 5.1.1
      ShiftRows(state)                       // See Sec. 5.1.2
      MixColumns(state)                      // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figure 5. Pseudo Code for the Cipher.[1]

Background – specifying crypto algorithms

From the Advanced Encryption Standard definition†

3.1 Inputs and Outputs

The **input** and **output** for the AES algorithm each consist of **sequences of 128 bits** (digits with values of 0 or 1). These sequences will sometimes be referred to as **blocks** and the number of bits they contain will be referred to as their length. The **Cipher Key** for the AES algorithm is a **sequence of 128, 192 or 256 bits**. Other input, output and Cipher Key lengths are not permitted by this standard.

- Cryptol can capture this precisely in a **type signature**

†http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Background – specifying crypto algorithms

From the Advanced Encryption Standard definition[†]

3.1 Inputs and Outputs

The **input** and **output** for the AES algorithm each consist of **sequences of 128 bits** (digits with values of 0 or 1). These sequences will sometimes be referred to as **blocks** and the number of bits they contain will be referred to as their length. The **Cipher Key** for the AES algorithm is a **sequence of 128, 192 or 256 bits**. Other input, output and Cipher Key lengths are not permitted by this standard.

blockEncrypt : {k} (k >= 2, 4 >= k) => ([128], [64*k]) -> [128]

For all k

...between 2 and 4

First input is a sequence of 128 bits

Second input is a sequence of 128, 192, or 256 bits

Output is a sequence of 128 bits

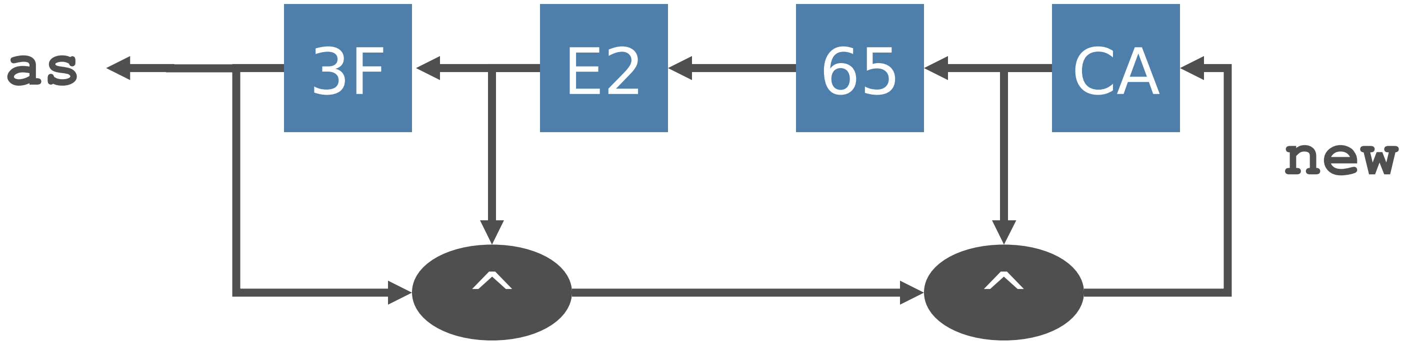
†http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Stream programming in Cryptol

- Recursive stream equations (similar to Haskell)

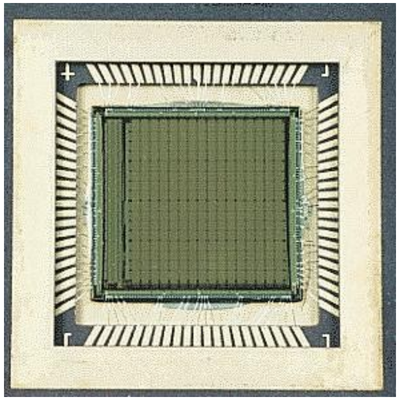
```
as  = [0x3F 0xE2 0x65 0xCA] # new;

new = [| a ^ b ^ c || a <- as
                   || b <- drop(1,as)
                   || c <- drop(3,as) |];
```



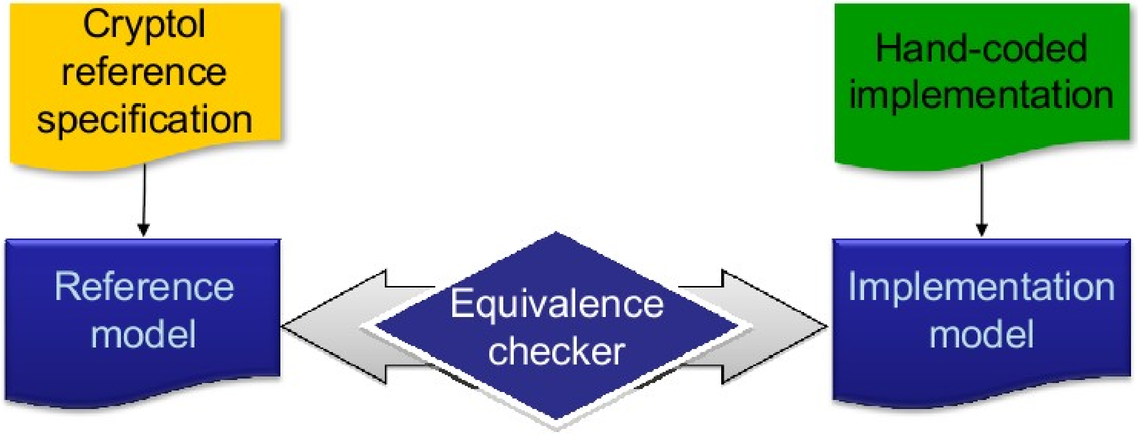
Cryptol – FPGA – a natural fit

- Data: bit streams processed by small blocks of code
- For fast crypto implementations: compile to FPGA (*field-programmable gate array*)
- Stream processing happens in parallel
- FPGAs are traditionally programmed in hardware-description languages (VHDL, Verilog)
- Other DSLs target VHDL/Verilog, e.g. Lava



Verification of crypto implementations

- Specification in Cryptol
- Optimized implementation in C or VHDL
- Challenge: prove that specification and implementation are equivalent **for all input**
- Solution: translate both specification and implementation into **boolean formulas** ("and-inverter graphs")
- Use a satisfiability solver to check for equivalence



Satisfiability in Cryptol

- Given a Cryptol predicate, e.g.

```
f : (Bit,Bit) → Bit;
f(x,y) = x & ~y;
```

- Is there x and y such that f(x,y) is true?
- Trivial in this case, but for large input widths and compex functions, not so!
- The Cryptol interpreter can translate f into a Boolean formula and feed to satisfiability solver

Equivalence checking in Cryptol

- Given two Cryptol functions f and g, the predicate

  ```
  \x -> f(x) == g(x)
  ```

  and a satisfiability solver can be used to check if f and g are equivalent, or find a counter example for x

Solving Sudoku in Cryptol with satisfiability solving

- By Levent Erkök at Galois
- Express problem as a predicate with the empty squares as variables
- Predicate is true if variables are assigned so that puzzle is solved

	9		7			8	6	
	3	1			5		2	
8		6						
		7		5				6
			3		7			
5				1		7		
						1		9
	2		6			3	5	
	5	4			8		7	

Solving Sudoku in Cryptol with satisfiability solving

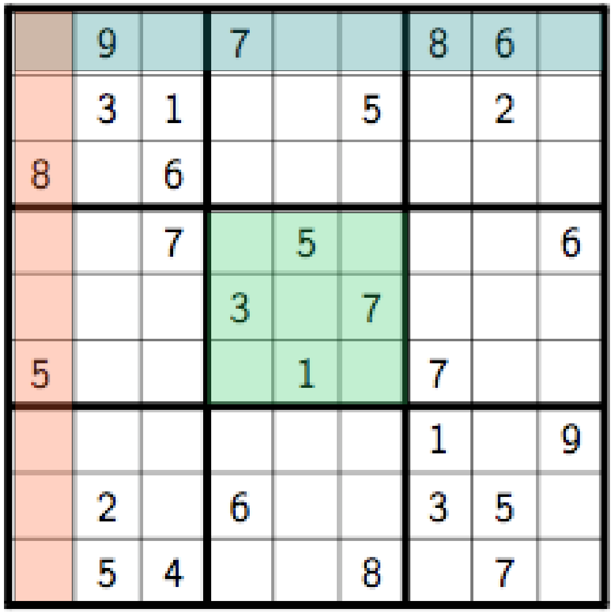
```
// Check that a sequence contains all digits 1 through 9
check : [9][4] -> Bit;
check group =
  [| contains x || x <- [1 .. 9] |] == ~zero
   where
  contains x = [| x == y || y <- group |] != zero;
```

```
// group is row, column or block
```

	9		7			8	6	
	3	1			5		2	
8		6						
		7		5				6
			3		7			
5				1		7		
						1		9
	2		6			3	5	
	5	4			8		7	

Solving Sudoku in Cryptol with satisfiability solving

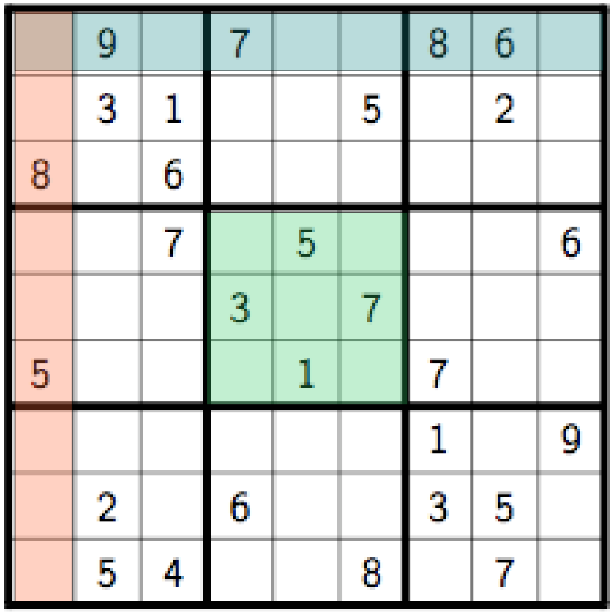
```
// Check that all groups in a board are valid
valid : [9][9][4] -> Bit;
valid rows =
  [| check group
  || group <- rows # columns # squares |] == ~zero
 where
  columns = transpose rows;
  regions = transpose [| groupBy (3, row)
                      || row <- rows |];
  squares = [| join sq
            || sq <- groupBy(3, join regions)
            |];
```



```
// Express a particular puzzle as a Cryptol predicate
puzzle : [53][4] -> Bit;

puzzle   [ a1    a3    a5 a6       a9
           b1       b4 b5    b7    b9
              c2    c4 c5 c6 c7 c8 c9
           d1 d2    d4    d6 d7 d8
           e1 e2 e3    e5    e7 e8 e9
              f2 f3 f4    f6    f8 f9
           g1 g2 g3 g4 g5 g6    g8
           h1    h3    h5 h6       h9
           i1       i4 i5    i7    i9 ]

 = valid [[a1  9 a3  7 a5 a6  8  6 a9]
          [b1  3  1 b4 b5  5 b7  2 b9]
          [ 8  c2  6 c4 c5 c6 c7 c8 c9]
          [d1 d2  7 d4  5 d6 d7 d8  6]
          [e1 e2 e3  3 e5  7 e7 e8 e9]
          [ 5 f2 f3 f4  1 f6  7 f8 f9]
          [g1 g2 g3 g4 g5 g6  1 g8  9]
          [h1  2 h3  6 h5 h6  3  5 h9]
          [i1  5  4 i4 i5  8 i7  7 i9]];
```



Solving Sudoku in Cryptol with satisfiability solving

- `puzzle : [53][4] -> Bit;`
- Input is a number with 53 digits – large search space!
- But by translating puzzle to boolean formula and using a satisfiability solver, we find a solution in < 2 seconds.
- This puzzle translates into about 5000 AND nodes
- Realistic crypto algorithms translate into > 500,000 nodes, may take > 1 hour to equivalence check

	9		7			8	6	
	3	1			5		2	
8		6						
		7		5				6
			3		7			
5				1		7		
						1		9
	2		6			3	5	
	5	4			8		7	

Domain-specific languages at Galois

- Why domain-specific languages?
  - Small domains enables more opportunities for verification, optimization, compilation
- Example – specifying crypto algorithms in Cryptol

The Cryptol team, past and present include:
Sally Browning, Magnus Carlsson, Ledah Casburn, Jonathan Daugherty, Iavor Diatchki, Trevor Elliott, Levent Erkök, Sigbjorn Finne, Andy Gill, Fergus Henderson, Joe Hendrix, Joe Hurd, John Launchbury, Jeff Lewis, Lee Pike, John Matthews, Thomas Nordin, Mark Shields, Joel Stanley, Frank Seaton Taylor, Jim Teisher, Philip Weaver, Adam Wick

Kathy Kopacek

My encounters with Functional Programming

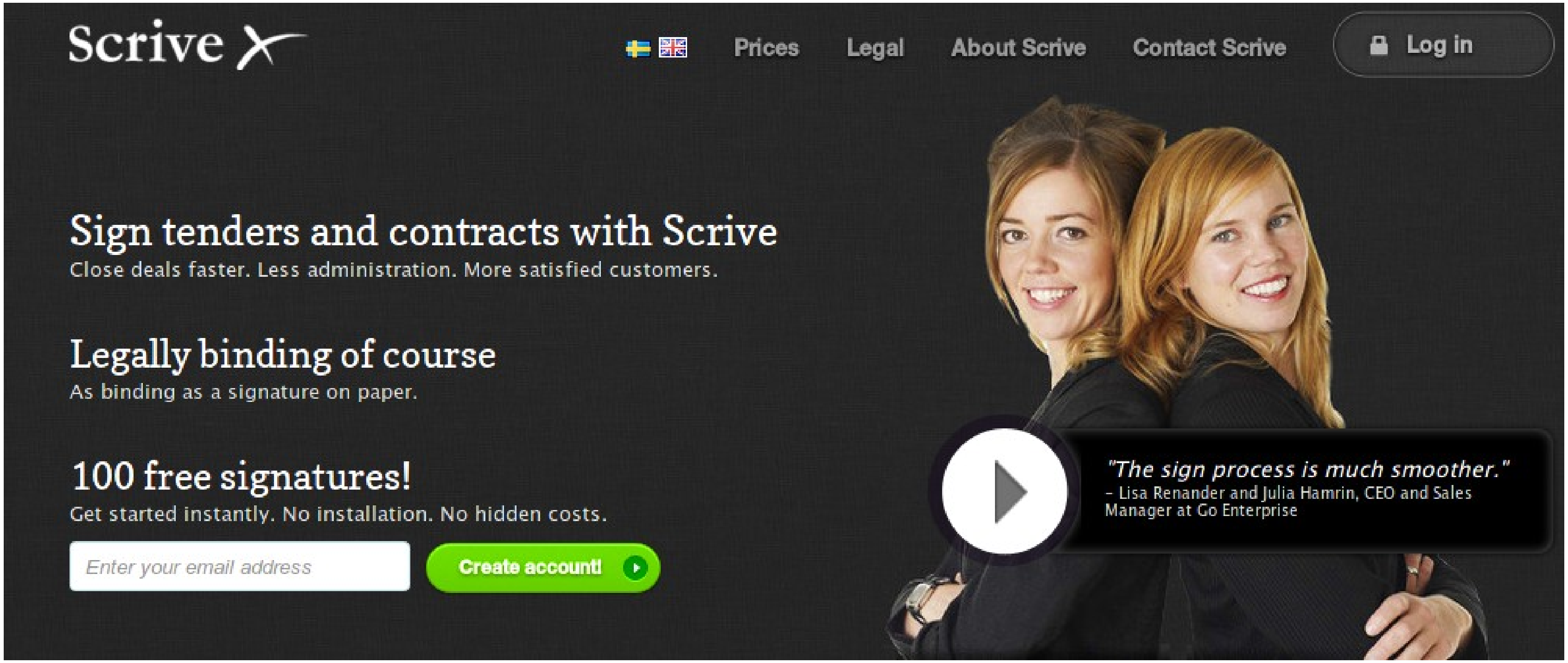
- ATC Solutions AB, 2011 –
  - Smart temperature control of residential homes
  - Central control developed in Haskell

ATC's Cyber Physical Systems

ATC has developed self-learning smart control systems that reduce the energy consumption in homes and buildings. The systems have been tested in various types of buildings both in Sweden and abroad.As the systems are integrated with the energy meters of the building, other possibilities are related to Advanced Metering Infrastructure but also Smart Grid (load management) and Intelligent Demand Response.Unique properties include that the systems are independent of the type of heating system (and cooling system for that matter) and that it automatically adjusts to the building and how it is being used.The systems optimize when to turn on and off heating when set-back control is used to lower the temperature in case a building isn't occupied. The calculation will be correct independent of the current indoor and outdoor temperature. ATC is referring to this technology as **"Just In Time Comfort"**. The specification for commissioning of the ATC systems is inspired by the best examples of effective solutions used during the Scandinavian smart metering roll-outs. The installer will e.g. use a smart phone as an installation and commissioning tool. Since the systems are connected it will be possible to detect instances when components are degrading and needs preventive support or service.

My encounters with Functional Programming

- Scrive.com (former Skrivapå), 2011 –
  - Web-based signing solutions (Stockholm)
    - Founded by Lukas Duczko & Gracjan Polak
  - Service developed in Haskell

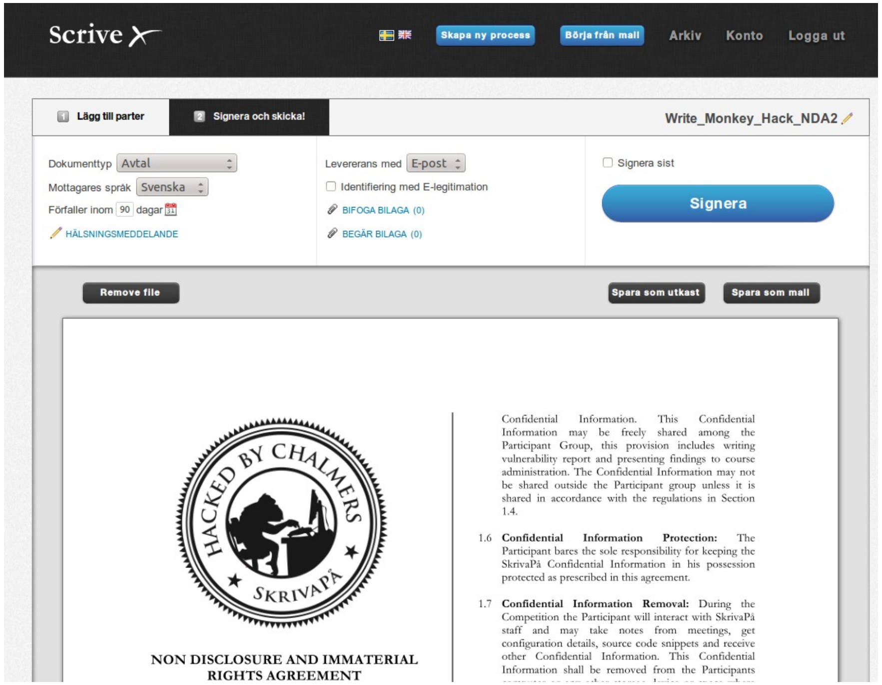


Scrive AB

- Founded 2010
- International team (about 10 people)

Scrive – sign documents on the web

- Author prepares document
- Sends invitation by email to other party (signatory)



Scrive – sign documents on the web

- Signatory reads email, follows link

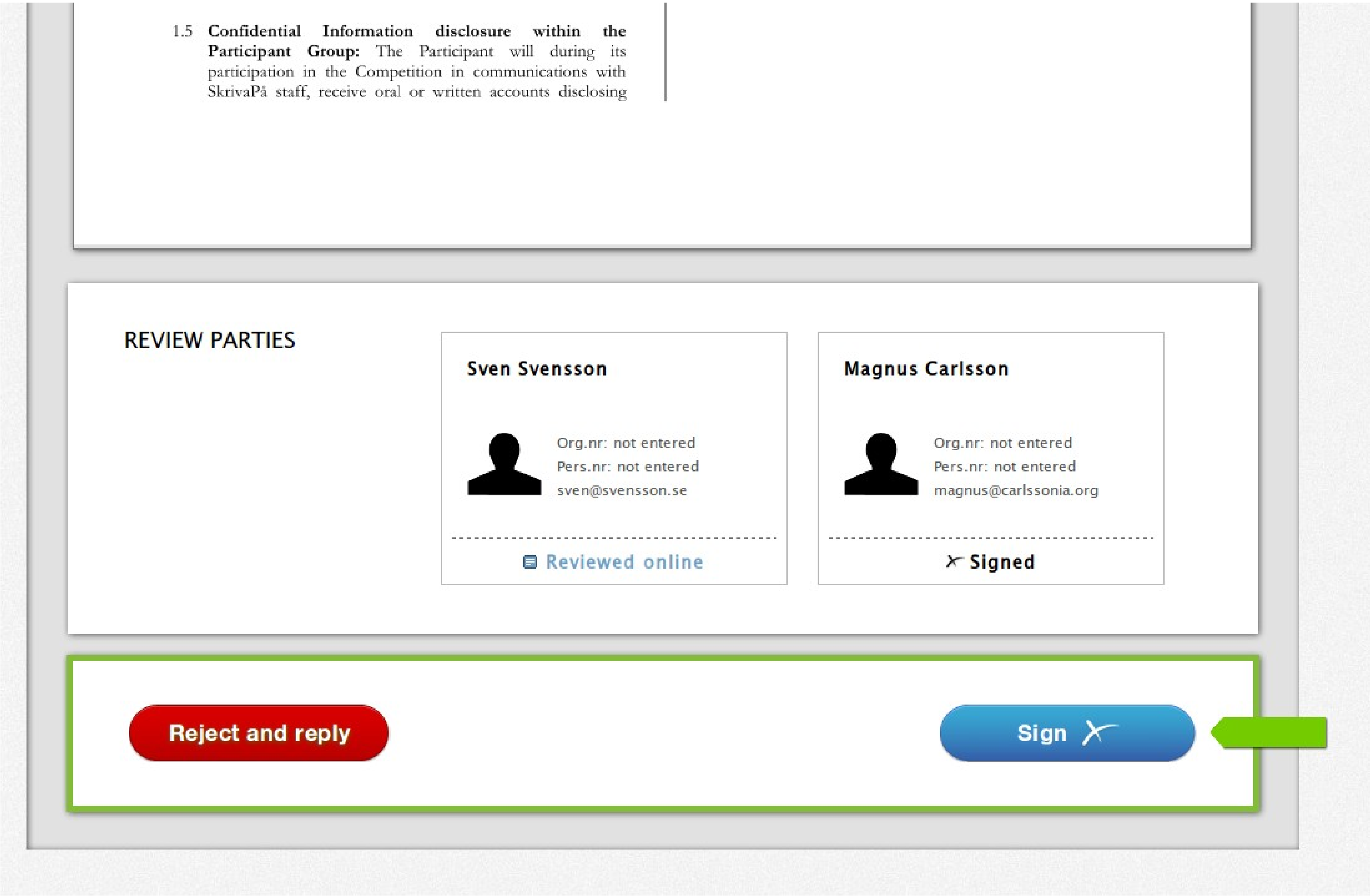
> Hi Sven Svensson,
>
> **Magnus Carlsson** has invited you to sign the document **Write_Monkey_Hack_NDA2**.
>
> *To continue:*
> 1. Click this link
> 2. Review online
> 3. Sign (or return)
> 4. Done

Scrive – sign documents on the web

- Signatory reviews agreement online

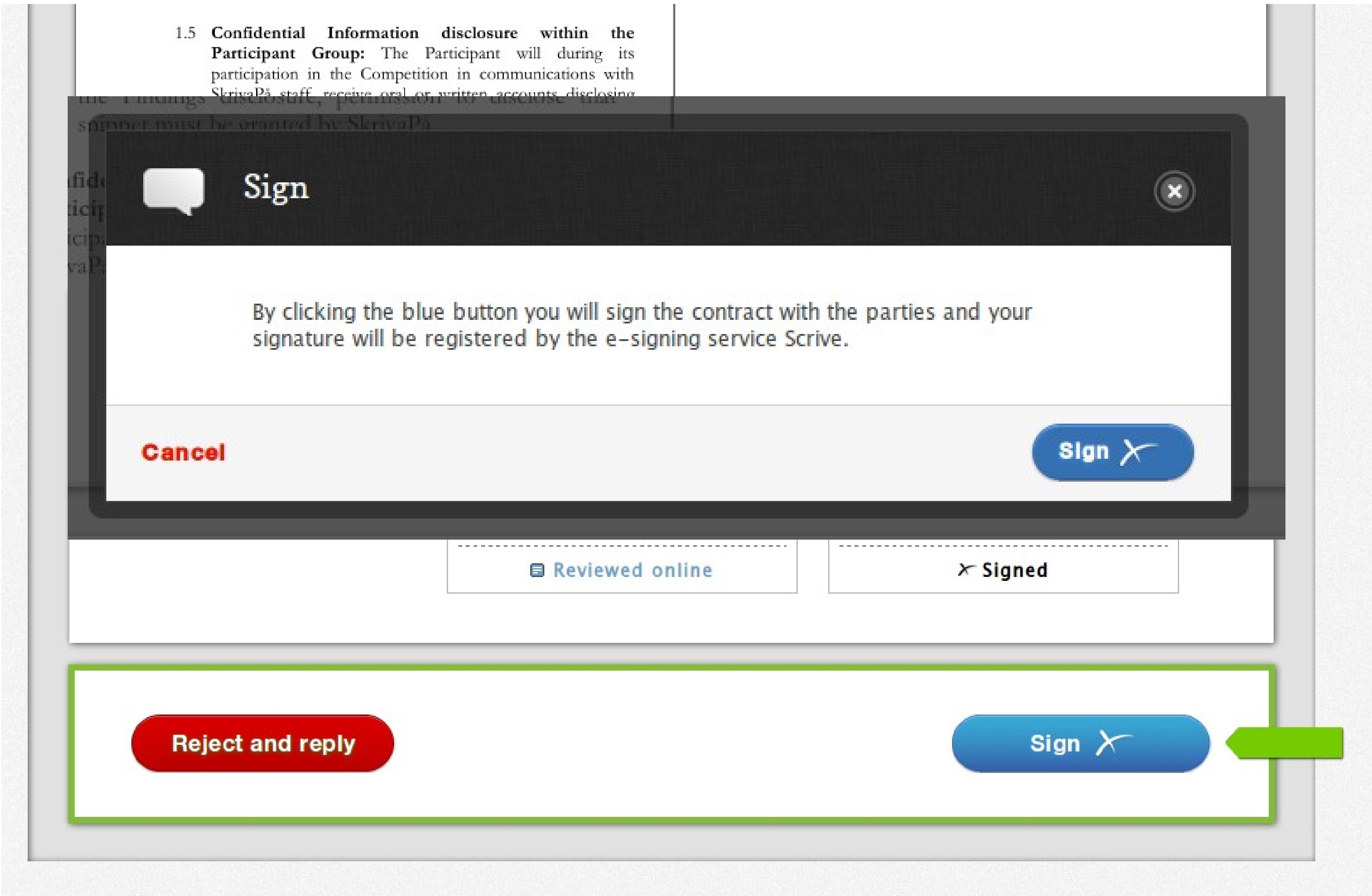
Scrive – sign documents on the web

- Signatory may reject or sign the agreement



Scrive – sign documents on the web

- Signatory may reject or sign the agreement



Scrive – sign documents on the web

- After signing, a sealed PDF with added verification page is mailed to all parties

Verification

Transaction 00000000000000000014

Document

Write_Monkey_Hack_NDA2
Main document

1 page
Sent by Magnus Carlsson

Signing parties

Magnus Carlsson	**Sven Svensson**
magnus@carlssonia.org	*sven@svensson.se*

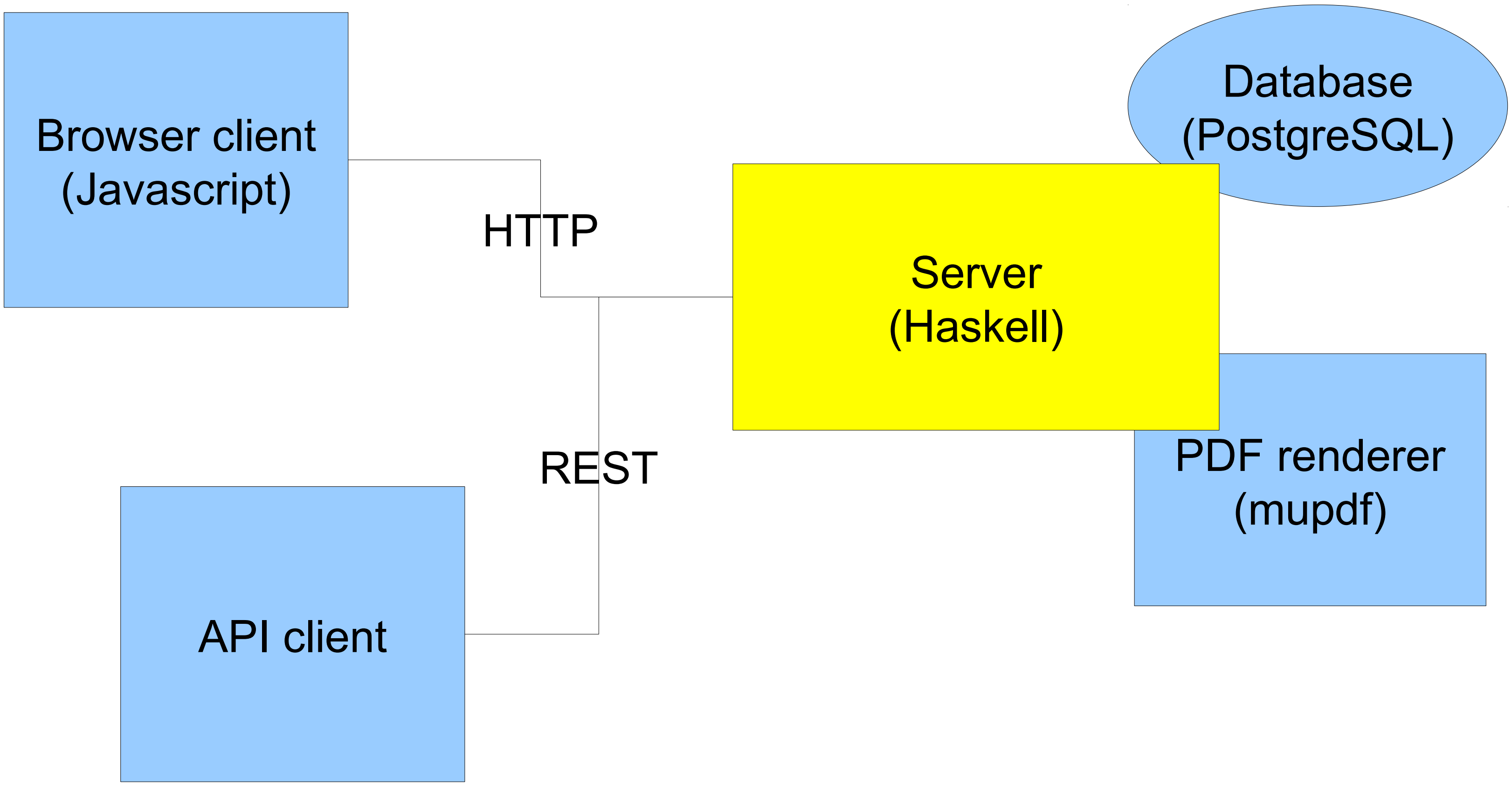
Registered events

2012-12-05, 12:00:43 CET *IP: 127.0.0.1*	*Magnus Carlsson signs the document online with email as verification method.*
2012-12-05, 12:00:43 CET *IP: 127.0.0.1*	*Scrive sends an invitation to sign to Sven Svensson.*
2012-12-05, 12:03:59 CET *IP: 127.0.0.1*	*Sven Svensson reviews the document online.*
2012-12-05, 12:11:49 CET *IP: 127.0.0.1*	*Sven Svensson signs the document online with email as verification method.*
2012-12-05, 12:11:49 CET	*The document is sealed and digitally signed by Scrive.*

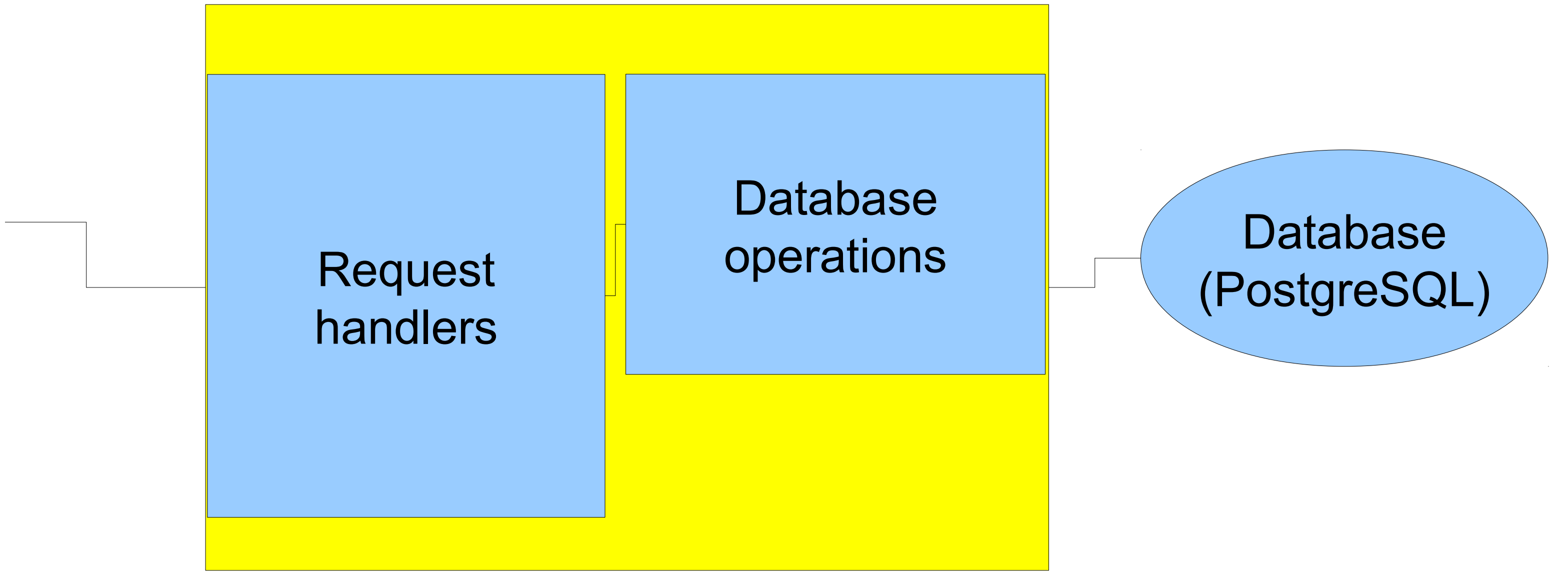
This verification was issued by Scrive. Information in italics has been safely verified by Scrive. The time stamp ensures that the originality of this document can be proven mathematically and independently of Scrive. For more information see the legal attachment (use a PDF-reader that can show concealed attachments). For your convenience Scrive also provides a service that enables you to automatically verify the documents originality at: https://scrive.com/verify

1/1

Scrive – architecture



Scrive – inside the server



Access control – putting Haskell to use at Scrive (original idea by Eric Normand)

- **Who** should be able to do **what**?
- Who:
  - Document author
  - Signatory
  - Company administrator
  - System administrator
  - Client on behalf of user through the API
  - ...
- What
  - View, edit, sign, reject document, ...

Access control – who is "who"?

- "who": an **actor** that initiates an operation
- Captured by a type class representing evidence about who initiated something

```
class Actor a where
  actorTime :: Time
  actorIPAddress :: Maybe IPAddress
  actorUserID :: Maybe UserID
  ...
```

Access control – actors

- Instances represent verified actors

```
instance Actor UserActor
  -- a logged-in user, e.g. for creating
  -- and sending documents

instance Actor SignatoryActor
   -- a signatory who received an
   -- a document to sign

instance Actor CompanyAdminActor
   -- a logged-in user with special powers
```

Access control – actors

- Actor instances are **abstract types**, with trusted functions for creating values
- Example of a web request handler:

```
handleSendDocument docid = do
  author ← mkUserActor
  dbUpdate author (SendDocumentInvitation docid)
```

- No need to check permissions in the request handler
- Good because we have many request handlers
- Where is permission checked?

Access control – what is "what"?

- "what": an **operation on the database**
- Captured by data types

```
data SendDocumentInvitation =
     SendDocumentInvitation DocumentID
 -- specific operation

data EditDocument DocumentID =
     EditDocument DocumentID
 -- describes whole set of operations
```

Access control – who can do what?

- Captured by a class **HasPermission**
- **Static check**: 'instance HasPermission a o' means an actor of type 'a' may have permission to do operations of type 'o'

```
class Actor a => HasPermission a o

instance HasPermission UserActor SendDocumentInvitation

instance HasPermission SignatoryActor SignDocument
```

Access control – who can do what?

- Captured by a class **HasPermission**
- **Static check**: 'instance HasPermission a o' means an actor of type 'a' may have permission to do operations of type 'o'
- **Dynamic check**: if all checks in 'permissionChecks a o' pass, the actor value 'a' can do the operation 'o'

```
class Actor a => HasPermission a o where
    permissionChecks :: a → o → [SQLPredicate]

instance HasPermission UserActor SendDocumentInvitation
  where
    permissionChecks u (SendDocumentInvitation docid) =
      ... -- check that 'u' is owner of 'docid'
```

Access control – summary

- Benefits of 'HasPermission'
  - **Declarative specification of access control**
    - First (conservative) approximation of access can be understood by looking at what instances of 'HasPermission' we have

      Example: no instance
      'HasPermission SystemAdminActor SignDocument'
  - **Separation of concerns**
    - Request handlers can deal with particular actors and parsing requests, but not worry about permissions
    - Database operations can deal with correct implementation of SQL queries, without worrying about who the actor is, or permissions

Thank you!