Friot: Functional Reactive Abstraction for IoT Programming

Computer Science Research Week @ Soc
07/08/2019
Yahui Song
Advisor: A/P Wei-Ngan Chin
Internet of Things (IoT)
Internet of Things (IoT)
Motivation Example — LCD

Requirement: to show the current temperature and a clock.
Requirement: to show the current temperature and a clock.

```c
#include <wiringPi.h>
#include <time.h>
#include <string.h>

int main() {
    if (wiringPiSetup() == -1) {
        return -1;
    }

    setup();
    while(1) loop();
    return 0;
}
```

GPIO library
(General-purpose input/output)
#define TempSensor 0
#define LCD 1

#include <wiringPi.h>
#include <time.h>
#include <string.h>

int main() {
    if (wiringPiSetup() == -1) {
        return -1;
    }
    setup();
    while(1) loop();
    return 0;
}

To initialize GPIO

void setup() {
    pinMode(TempSensor, INPUT);
    pinMode(LCD, OUTPUT); // actually needs more parameters
}

void loop() {
    int temp = digitalRead(TempSensor);

    // ... initialize the timer ...
    tm_info = localtime(&timer);
    strftime(buffer_time, 26, "Time: %H:%M:%S", tm_info);

    string data = strcat(to_string(temp), buffer_time);
    LcdPuts(LCD, data);
}

Requirement: to show the current temperature and a clock.

1. Read temperature
2. Get current time
3. Show on the lcd


**Requirement:** to show the current temperature and a clock.

```
#include <wiringPi.h>
#include <time.h>
#include <string.h>

int main() {
    if (wiringPiSetup() == -1) {
        return -1;
    }
    setup();
    while(1) loop();
    return 0;
}
```

1. High reliance on global values
2. Global delay
3. Long running data flow (callbacks)
#define TempSensor 0
#define LCD 1

void setup() {
    pinMode(TempSensor, INPUT);
    pinMode(LCD, OUTPUT); // actually needs more parameters
}

void loop() {
    int temp = digitalRead(TempSensor);
    // ... initialize the timer ...
    tm_info = localtime(&timer);
    strftime(buffer_time, 26, "Time: %H:%M:%S", tm_info);
    string data = strcat(to_string(temp), buffer_time);
    lcdPuts(LCD, data);
}

#include <wiringPi.h>
#include <time.h>
#include <string.h>

int main() {
    if (wiringPiSetup() == -1) {
        return -1;
    }
    setup();
    while(1) loop();
    return 0;
}

1. High reliance on global values
2. Global delay
3. Long running data flow (callbacks)
Signal Stream Graphs — Core Design

Signals:
- Continuously changing
- Connect to the world
- Infinite

Signals graphs:
- Static
- Multi-threaded
- Asynchronous by default

Pushing the side effect to the edge of the system!
Input / Output are Signals

- Evn.temperature :: Int → Signal Float
  - Temperature

- clock :: Signal (Int, Int, Int)
  - Day : Hour : Second

- lcd :: Int → Signal String → IO ()
  - LCD display

Higher-order functions!
High-order functions — Transformation

- \( \text{lift} :: (a \rightarrow b) \rightarrow \text{Signal } a \rightarrow \text{Signal } b \)

```haskell
data :: Signal String

data = lift floatToStr (Evn.temperature 0)
```

port number
High-order functions — Transformation

- \( \text{lift} :: (a \rightarrow b) \rightarrow \text{Signal } a \rightarrow \text{Signal } b \)
- \( \text{lift}_2 :: (a \rightarrow b \rightarrow c) \rightarrow \text{Signal } a \rightarrow \text{Signal } b \rightarrow \text{Signal } c \)
- \( \text{lift}_n \ldots \)

\[ \text{data} :: \text{Signal } \text{String} \]
\[ \text{data} = \text{lift } \text{floatToStr} \ (\text{Evn}.\text{temperature } 0) \]
High-order functions — State

- **Fold from the past**

- \( \text{foldP} :: (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow \text{Signal } a \rightarrow \text{Signal } b \)

- \( \text{totalStep} :: \text{Signal } \text{Int} \)

- \( \text{totalStep} = \text{foldP} \ (\\text{\_step count } \rightarrow \text{count } + \ 1) \ 0 \ (\text{Evn.motion } 0) \)
If the temperature rose too high, the air conditioner (AC) would be turned on automatically.

**Functional Reactive IoT Programming**
Functional **Reactive** IoT Programming

Passive

Reactive
**Functional Reactive IoT Programming**

- Update method is defined in the **Sensor** module
- Remote setters and updates
- **AC** has no awareness on the dependence
Functional **Reactive** IoT Programming

- Update method is defined in the **Sensor** module
- Remote setters and updates
- **AC** has no awareness on the dependence

- Update method is defined in the **AC** module
- Events, observations and self-updates
- Easy to track/add dependencies on **AC** module
**Functional Reactive IoT Programming**

- Update method is defined in the **Sensor** module
- Remote setters and updates
- **AC** has no awareness on the dependence

- Update method is defined in the **AC** module
- Events, observations and self-updates
- Easy to track/add dependencies on **AC** module

"How does this module work?"
Revisit: Requirement:
to show the current temperature and a clock. **Use Friot!**

```haskell
import Rpi
import Env
import Time

show :: Signal String
show = lift_2 (,) (Env.temperature 0) Time.everySec

main :: IO ()
main = Rpi.bPlus [(lcd 2 show)]
```

More concise code (LOC: 40 VS 9)
easy to read and easy to write
Revisit: Requirement:

to show the current temperature and a clock. Use Friot!

```haskell
import Rpi
import Env
import Time

show :: Signal String
show = lift_2 (,) (Env.temperature 0) Time.everySec

blink :: Signal Bool
blink = foldP (\a state -> not state) False Time.everySec

main :: IO ()
main = Rpi.bPlus [lcd 2 show]
    ,led 3 blink

More concise code (LOC: 40 (+12) VS 9 (+3))
easy to read and easy to write
```
import Rpi
import Env

helper :: Signal Bool
helper a = if a then True else False

isPeopleIn :: Signal Bool
isPeopleIn = lift helper (Env.motion 0)

mode_LCD :: Signal Bool
mode_LCD = lift helper isPeopleIn

lcd_show :: Signal String
lcd_show = lift_2 (\a b -> if a then toStr b else "null")
          mode_LCD (Env.temprature 1)

main :: IO ()
main = Rpi.bPlus [ (lcd 2 lcd_show)
                ,(led 3 isPeopleIn) ]
Signal Graph Transformation — multithreads

Input signals

isPeopleIn

mode_LCD

lcd_show

Output signals

Global Signal dispatcher
Signal Graph Transformation — multithreads

Input signals

- isPeopleIn
- mode_LCD
- lcd_show

Output signals

- isPeopleIn
- mode_LCD

Global Signal dispatcher
Signal Graph Transformation — multithreads

Input signals

isPeopleIn

mode_LCD

lcd_show

Output signals

Global Signal dispatcher

mode_LCD

lcd_show
Signal Graph Transformation — multithreads

Input signals

isPeopleIn

mode_LCD

lcd_show

Output signals

Global Signal dispatcher

isPeopleIn

mode_LCD

lcd_show
Signal Stream Graphs — Benefits

1. High reliance on global values
2. Global delay
3. Long running data flow (callbacks), passive

1. Immutability of functional language
2. Efficient (Asynchronous and multithreads)
3. Reactive programming style
Conclusion

• Friot is a new FRP language designed for IoT control systems

• Has many functional programming features

• Embedded in Haskell Compiles to C (multithreads)

• Shows clear benefits for logic re-use; specifically with time dependent behaviors, being able to be formal verified.

• Efficient (Asynchronous)
Conclusion

• Friot is a new FRP language designed for IoT control systems

• Has many functional programming features

• Embedded in Haskell Compiles to C (multithreads)

• Shows clear benefits for logic re-use; specifically with time dependent behaviors, being able to be formal verified.

• Efficient (Asynchronous)

https://www.comp.nus.edu.sg/~yahuis/
Conclusion

- Friot is a new FRP language designed for IoT control systems
- Has many functional programming features
- Embedded in Haskell, compiles to C (multithreads)
- Shows clear benefits for logic reuse; specifically with time-dependent behaviors, being able to be formally verified.
- Efficient (Asynchronous)

https://www.comp.nus.edu.sg/~yahuis/

Thanks a lot for your attention! 😊
Formal Verification — Refinement Types

• n :: \{ v : \text{Int} \mid v \geq 0 \} \quad \text{basic type}

• max :: x: \text{Int} \rightarrow y: \text{Int} \rightarrow \{ v : \text{Int} \mid x \leq v \land y \leq v \} \quad \text{function type}

• xs :: \{ v: \text{List Nat} \} \quad \text{polymorphic datatype}

Evn.temperature :: \text{Signal Int} \rightarrow \text{Signal Float}

Evn.temperature :: \text{Signal} \{ v: \text{Int} \mid v \leq 20 \land v \geq 0 \} \rightarrow \text{Signal} \{ t: \text{Float} \mid t > 10 \}