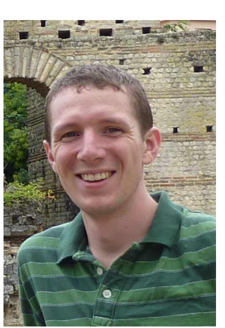
Declarative GUIs: Simple, Consistent, and Verified

Stephan Adelsberger
Vienna University
of Economics
Austria

Anton Setzer
Swansea
University
UK



Eric Walkingshaw
Oregon State
University
USA

PPDP 2018, Frankfurt, Germany
on memory stick: PPDP2018/a4-adelsberger.pdf
5 September 2018

Dependent Type Theory and GUIs

Examples

Verification of GUIs

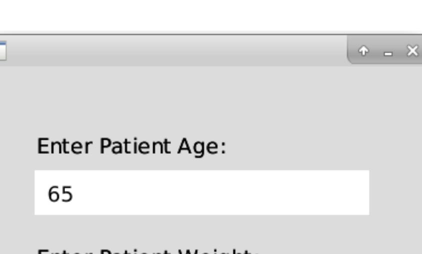
Conclusion

Dependent Type Theory and GUIs

Examples

Verification of GUIs

Conclusion


Enter Patient Age:
65
Enter Patient Weight:
70
Continue

Verified Apps in Medical Domain

- Apps are increasingly used in safety critical applications, especially medical domain.
  - Example: assistance for prescription of medicines.
- Current assumption: responsibility is with the doctor.
  - Not sustainable.
- Testing not sufficient since coverage of all cases not possible for complex apps.
- Our approach: Create verified apps running directly in the Agda (theorem prover based on dependent types).
  - Avoids translation into a different language which can be source of new errors.
- Approach goes beyond finite state machines.
  - Data aware GUIs (arbitrary inputs).
  - Arbitrary many interactions between each GUI frame allowed.

Event Handlers as Dependent Types

- Common approach for designing graphical user interfaces is observer pattern based on action listeners.
- Two parts
  - Layout of GUI (called in our setting Frame).
  - Action listeners) handling all events (such as button clicks, mouse events) (called in our setting FrameObj).
- FrameObj
  - depends on the events of Frame.
  - can modify the elements in Frame.
- Therefore its type depends on Frame.

Event Handlers as Dependent Types

- When modifying Frame, therefore FrameObj needs to be adapted.
- Generic operations for modifying GUIs cannot be expressed without dependent types in a type correct way.
- Approaches to overcome this problem:
  - Use of dynamically typed programming languages.
  - Use of GUI builders which adapt user interfaces by program transformation resulting in machine generated code.
  - Our Proposal: Use of dependent types.

GUI Data Type Using Dependent Types

```
record GUI : Set where
  gui : Frame
  obj : FrameObj gui
```

State Dependent Objects

```
record Interfaceˢ : Set₁ where
  Stateˢ   : Set
  Methodˢ  : Stateˢ → Set
  Resultˢ  : (s : Stateˢ) → Methodˢ s → Set
  nextˢ    : (s : Stateˢ) (m : Methodˢ s) → Resultˢ s m
             → Stateˢ
```

Assuming $(State, Method, Result, next)$: Interface^s

```
record IOObjectˢ (s : State) : Set where
  coinductive
  method :
    (m : Method s) → IO (Σ[ r ∈ Result s m ] IOObjectˢ (next s m r))
```

GUIInterface

$$\mathsf{GUIState} = \mathsf{Frame}$$

$$\begin{aligned}\mathsf{GUIElMethod}\ \mathit{gui} = {} & \mathsf{Fin}\ (\mathsf{guiEl2NrButtons}\ \mathsf{frameCmpStruc}\ \mathsf{frame}\ \mathit{gui}) \\ & \times\ \mathsf{Tuple}\ \mathsf{String} \\ & \quad (\mathsf{guiEl2NrTextboxes}\ \mathsf{frameCmpStruc}\ \mathsf{frame}\ \mathit{gui})\end{aligned}$$

$$\mathsf{GUIElResult}\ \mathit{gui}\ m = \mathsf{Frame}$$

$$\mathsf{nextGUI}\ \mathit{gui}\ m\ r = r$$

$$\begin{aligned}&\mathsf{GUIInterface} : \mathsf{Interface}^{\mathsf{s}} \\ &\mathsf{GUIInterface}\ .\mathsf{State}^{\mathsf{s}} = \mathsf{GUIState} \\ &\mathsf{GUIInterface}\ .\mathsf{Method}^{\mathsf{s}} = \mathsf{GUIElMethod} \\ &\mathsf{GUIInterface}\ .\mathsf{Result}^{\mathsf{s}} = \mathsf{GUIElResult} \\ &\mathsf{GUIInterface}\ .\mathsf{next}^{\mathsf{s}} = \mathsf{nextGUI}\end{aligned}$$

GUI Data

```
FrameObj : Frame → Set
FrameObj gui = IOObjectˢ GUIInterface gui
```

Used in the definition from above

```
record GUI : Set where
  gui : Frame
  obj : FrameObj gui
```

Dependent Type Theory and GUIs

Examples

Verification of GUIs

Conclusion

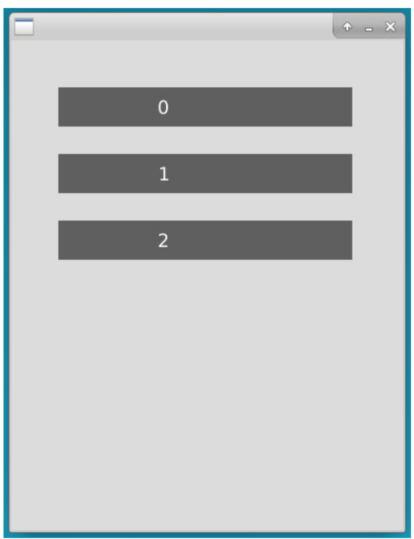
Example Infinite Buttons

$$
\begin{aligned}
&\mathsf{nFrame} : (n : \mathbb{N}) \to \mathsf{Frame}\\
&\mathsf{nFrame}\ 0 \qquad\quad = \mathsf{emptyFrame}\\
&\mathsf{nFrame}\ (\mathsf{suc}\ n) = \mathsf{addButton}\ (\mathsf{show}\ n)\ (\mathsf{nFrame}\ n)
\end{aligned}
$$

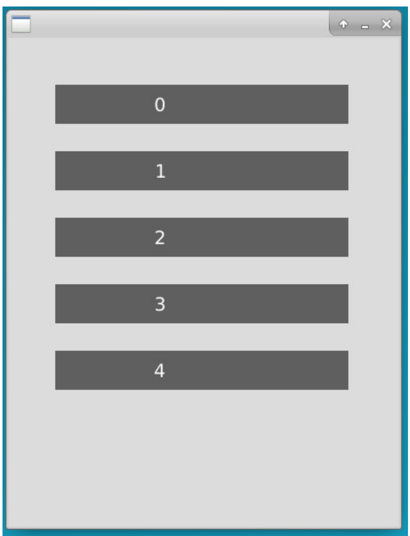
Object defined by copattern matching:

$$
\begin{aligned}
&\mathsf{infiniteBtns} : \forall\{i\} \to (n : \mathbb{N}) \to \mathsf{GUI}\ \{i\}\\
&\mathsf{infiniteBtns}\ n\ .\mathsf{gui} = \mathsf{nFrame}\ n\\
&\mathsf{infiniteBtns}\ 0\ .\mathsf{obj}\ .\mathsf{method}\ ()\\
&\mathsf{infiniteBtns}\ (\mathsf{suc}\ n)\ .\mathsf{obj}\ .\mathsf{method}\ (m\ ,\ _) =\\
&\quad \mathsf{returnGUI}\ (\mathsf{infiniteBtns}\ (n + \mathsf{finTo}\mathbb{N}\ m))
\end{aligned}
$$

Example Infinite Buttons



Example Infinite Buttons

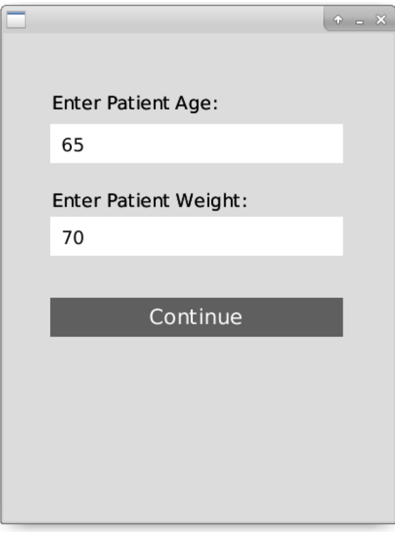


Business Processes

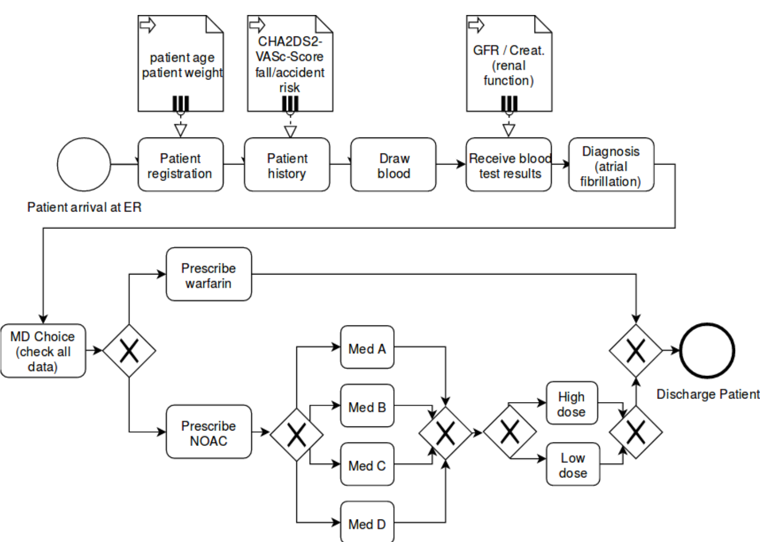
```
data BusinessModel : Set where
  terminate : String → BusinessModel
  xor       : List (String × BusinessModel)
              → BusinessModel
  input     : {n : ℕ} → Tuple String n
              → (Tuple String n → BusinessModel)
              → BusinessModel
  simple    : String → BusinessModel → BusinessModel
```

```
businessModel2Gui : BusinessModel → GUI
```

Screenshot



Medical Example



Model of Medical Process

$$
\begin{array}{lll}
\text{discharge} & = & \text{terminate}\ \texttt{"Discharge Patient"} \\
\text{lowdoseSelection} & = & \text{simple}\ \texttt{"Low Dose"}\ \ \text{discharge} \\
\text{highdoseSelection} & = & \text{simple}\ \texttt{"High Dose"}\ \text{discharge}
\end{array}
$$

$$
\begin{array}{l}
\text{doseSelectionA} : \text{WghtCat} \rightarrow \text{BusinessModel} \\
\text{doseSelectionA}\ {\leq}60 = \text{lowdoseSelection} \\
\text{doseSelectionA}\ {>}60 = \text{highdoseSelection}
\end{array}
$$

Model of Medical Process

```
bloodTestRes : FallRisk → AgeCat → WghtCat
                → BusinessModel
bloodTestRes f a w =
  input "Enter Bloodtest Result" λ str →
  diagnosis f (str2RenalCat str ) a w
```

Dependent Type Theory and GUIs

Examples

Verification of GUIs

Conclusion

Verification States of the GUI

```
data MethodStarted (g : GUI) : Set where
  notStarted : MethodStarted g
  started : (m : GUIMethod g)
            (pr : IO consoleI ∞ GUI) → MethodStarted g

data State : Set where
  state : (g : GUI) → MethodStarted g → State

Cmd : State → Set
Cmd (state g notStarted)             = GUIMethod g
Cmd (state g (started m (exec' c f))) = IOResponse c
Cmd (state g (started m (return' a))) = ⊤

guiNext : (g : State) → Cmd g → State
```

State Reached after Inputs

```
stateAfterBloodTest :
  (strAge strWght strFallR strScore strBlood : String)
  → State
stateAfterBloodTest strAge strWght strFallR strScore strBlood
  = guiNexts patientRegistrationState
               (nilCmd
                 >>> textboxInput2 strAge strWght
                 >>> textboxInput2 strFallR strScore
                 >>> btnClick
                 >>> textboxInput   strBlood)
```

Theorem 1

```
theoremWarfarin :
  (strAge strWght strFallR strScore strBlood : String)
    → str2RenalCat strBlood ≡ <25
    → stateAfterBloodTest strAge strWght strFallR
                          strScore strBlood
      -eventually-> warfarinState
```

Theorem 2

```
theoremNoLowDosisWeight>60 :
  (strAge strWght strFallR strScore strBlood : String)
  → str2WghtCat strWght ≡ >60
  → (w' : WghtCat)
  → stateAfterBloodTest strAge strWght strFallR
       strScore strBlood
     -gui-> NOACSelectionAState w'
  → (s : State)
  → NOACSelectionAState w' -gui-> s
  → ¬ (s ≡ lowdoseSelectionState)
```

Dependent Type Theory and GUIs

Examples

Verification of GUIs

Conclusion

Conclusion

- Event handlers depend on the frame, therefore are dependently typed.
- Event handler modelled by a state dependent object.
- More complex GUIs modelled by using a simple declarative data type for Business Processes.
- Model given by
  - States = States of the GUI
  - Transitions = GUI responses and IO events.

Conclusion

- Proof of correctness of GUIs.
- In medical domain
  - Conditions demanded originate from clinical studies which give negative results excluding certain medications and doses.
  - Programers write a program in a positive way which determines prescriptions.
  - Verification connects the two by showing that the program written by the user fulfils the conditions demanded by medicine.
- Program constantly changes in response to demands and changes of medicine.
- Declarative approach and dependent types support rapid adaption of program and verification.

Bibliography I

A. Abel, S. Adelsberger, and A. Setzer.
ooAgda.
Agda Library. Available from https://github.com/agda/ooAgda, 2016.

A. Abel, S. Adelsberger, and A. Setzer.
Interactive programming in Agda – Objects and graphical user interfaces.
Journal of Functional Programming, 27, Jan 2017.
doi 10.1017/S0956796816000319.

Bibliography II

S. Adelsberger, B. Igried, M. Moser, V. Savenkov, and A. Setzer.
Formal verification for feature-based composition of workflows, 2018.
To appear in proceedings of 10th International Workshop on Software Engineering for Resilient Systems SERENE 2018, 10 September 2018, Iasi, Romania. Available from http://www.cs.swan.ac.uk/~csetzer/articles/SERENE/SERENE18/SERENE18AdelsbergerIgriedMoserSavenkovSetzerfinal.pdf.

S. Adelsberger, A. Setzer, and E. Walkingshaw.
Declarative GUIs: Simple, consistent, and verified.
Git respository, 2017.

Bibliography III

S. Adelsberger, A. Setzer, and E. Walkingshaw.
Declarative GUIs: Simple, consistent, and verified, 2018.
To appear in Proceedings of PPDP'18. Available from
http://www.cs.swan.ac.uk/~csetzer/articles/PPDP18/PPDP18adelsbergerSetzerWalkingshawFinal.pdf.