Department of Electrical Engineering and Computer Science, Northwestern University

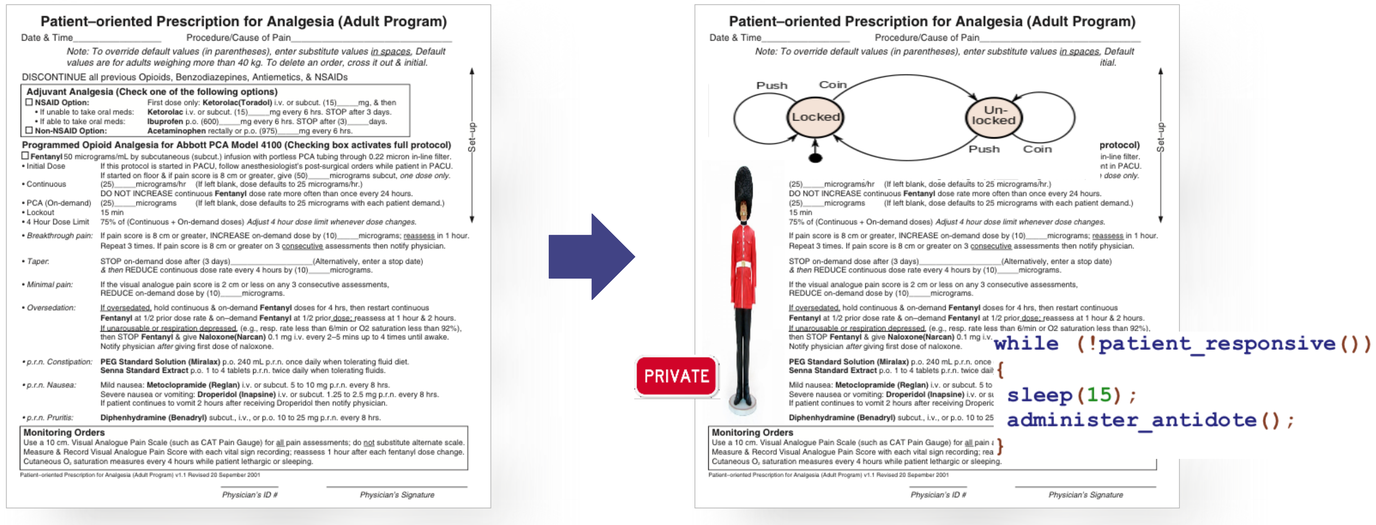
Programming Prescriptions: A Clinician Usable Patient-Oriented Prescription Programming Language

Spencer P. Florence[1] Burke Fetscher[1] Matthew Flatt[2] William H. Temps[1] Tina Kiguradze[1] Dennis P. West[1] Charlotte Niznik[1] Paul R. Yarnold[3] Robert Bruce Findler[1] Steven M. Belknap[1]

[1]Northwestern University [2]University of Utah [3]Optimal Data Analysis LLC

Key Idea: Prescriptions are programs

Prescribers are akin to programmers without a programming language. As demonstrated below, many of constructs in natural language prescriptions are directly analogous to programming constructs. By creating a creating a prescription programming language that can represent prescriptions and is usable by clinicians, we hope to lift the purely mechanistic tasks from the mental load of health care workers. We hope this will free clinicians' time to truly care for their patients.



An Example Program

```
#lang pop-pl

used by JessieBrownVA

initially
   giveBolus 80 units/kg of: HEParin by: iv
   start 18 units/kg/hour of: HEParin by: iv

infusion:
  whenever new aPTTResult
    aPTT < 45           | giveBolus 80 units/kg of: HEParin by: iv
                        | increase HEParin by: 3 units/kg/hour

    aPTT in 45 to 59    | giveBolus 40 units/kg of: HEParin by: iv
                        | increase HEParin by: 1 unit/kg/hour

 // aPTT in 59 to 101   | Continue current HEParin dose

    aPTT in 101 to 123  | decrease HEParin by: 1 unit/kg/hour

    aPTT > 123          | hold HEParin
                        | after 1 hour
                        |     restart HEParin
                        |     decrease HEParin by: 3 units/kg/hour

aPTTChecking:
  every 6 hours checkaPTT whenever aPTTResult outside of 59 to 101, x2
  every 24 hours checkaPTT whenever aPTTResult in range 59 to 101, x2
```

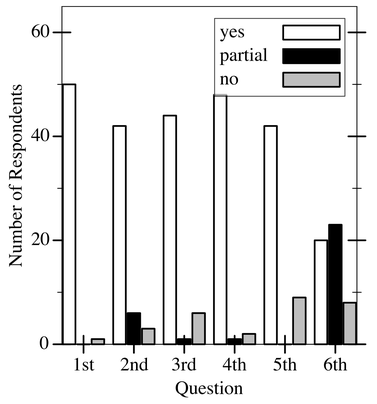
POP-PL programs coordinate with the hospital via message passing. The program on the left implements a prescription for the blood thinner Heparin, used at Jesse Brown VA in Chicago. It begins by messaging the nurse to give a one time large dose of the drug, and begin a continuous infusion. It then begin messaging the nurse to draw blood for the aPTT assay to assess the drugs effect (the **`aPTTChecking:`** section). This blood draw is scheduled for every 6 hours when the patient is not stable, and every 24 hours otherwise. It then titrates the dosage based on the result of that test (the **`infusion:`** section), possibly by giving another bolus or by stopping the I.V. entirely for a time.

Example Simulated Interaction

In our prototype implementation, when a program is run it starts a simulation of prescription which can be interacted with via a read-eval-print loop (REPL). The REPL first prints out the messages that would be sent from the prescription in a deployed version. It then prints a prompt where the user can submit messages to simulate responses from the world. This simulation does not display timestamps with the messages, as it has no real notion of time. Instead time is advanced manually with the special command **`wait`**. Message sent form the program are marked with **`[]`** s. Messages sent to the program begin with a **`>`**, the REPLs prompt.

```
[givebolus 80 units/kg HEParin I.V.]
[start 18 units/kg/hour HEParin I.V.]
[checkaptt]
> aPTTResult 46 seconds
[givebolus 40 units/kg HEParin I.V.]
[increase HEParin 1 unit/kg/hour]
> wait 6 hours
[checkaptt]
> wait 1 hour
> aPTTResult 70 seconds
> wait 6 hours
[checkaptt]
> wait 1 hour
> aPTTResult 74 seconds
> wait 6 hours
```

User Survey



We evaluated POP-PL's usability by giving a user server to 51 physicians, pharmacists, and advanced practice nurses at varying levels of training. The user survey had them read the heparin prescription to the left, and asked them to identify specific parts of a prescription and to make minor modifications to it. The results were graded by three independent raters, with the possiblities: "yes" meaning a totally incorrect response, "partial" meaning a partially correct response, and "no" for a totally incorrect response. UniODA was performed to assess inter-rater agreement for all 18 pairings of three independent raters scoring six separate test questions. Due to absence of variability in scores assigned by at least one rater, no model was possible for 7 pairings: for these combinations of raters and questions inter- rater agreement was 91.5% or greater. For the remaining 11 pairings, 5 indicated perfect agreement, 2 indicated strong agreement, and 4 indicated moderate agreement: inter-rater agreement was 75.5% or greater for these analyses, and all results were statistically significant with Bonferroni adjusted $p<0.05$. For all questions except the question about programming comfort, inter-rater agreement was 90.2% or greater. All instances of rater disagreement were resolved to consensus in post-rating discussion.