# A functional calculus of physiological evidence

Thomas A Nielsen[1,2], Henrik Nilsson[2], Tom Matheson[1]

[1]Department of Biology, Leicester University [2]School of Computer Science, University of Nottingham

## Introduction

There is increased recognition of the need for machine-readable experiment definitions. A universal notation would facilitate the sharing and replication of results and clear up apparent inconsistencies between studies. Whether experiments are carried out by machines or humans, they can essentially be seen as programs that can manipulate and observe the real world. There are currently two fundamental approaches to defining such programs:

- Configuration files
  - easy to write by hand or with a GUI
  - can be flexible, e.g. XML formats
  - but ultimately have a limited scope
  - little or no facility for abstraction
    - complex experiments are infeasible

```
NumRepetitions = 5
Velocity = 40
ViewingAngle = 25
Interval = 120
```

- Imperative programming language
  - much more flexible
  - more difficult to:
    - reason about the experiment or ultimate source of observations
    - read and write

```
FOR I = 1 TO 5
  WAIT(60)
  X=X+1
  SPIKES=SIMULATE(X)
NEXT I
```

- Can we do better with a declarative ("*what*, not *how*") language?

## Lambda calculus and functional programming

- Calculating exclusively with purely mathematical functions.
- Functions are first class entities, which means they can be stored in variables, passed as values to other functions and composed.
- Expressions in the (pure) lambda calculus have no side effects. In particular, there is no variable mutation, state or input/output.
- The lambda calculus forms the basis for almost all interactive proof assistants (Coq, Isabelle, HOL, Agda, ACL2 etc.) used to mechanically verify mathematical proofs.
- There are several high-performance implementations (Haskell, ML, Clean).
- Types

| | | |
|---|---|---|
| $\mathbb{R}$ | Real numbers | 3.141252... |
| $\mathbb{Z}$ | Integers | 3 |
| $\boldsymbol{Bool}$ | Booleans | $\boldsymbol{True}$, $\boldsymbol{False}$ |
| $()$ | Unit type (No information) | () |
| $[\alpha]$ | List of type $\alpha$ | [1,2,3] |
| $\alpha \times \beta$ | Pair of $\alpha$ and $\beta$ | (5,()) |
| $\alpha \rightarrow \beta$ | Function from $\alpha$ to $\beta$ | $\lambda x \rightarrow x > 5$ |

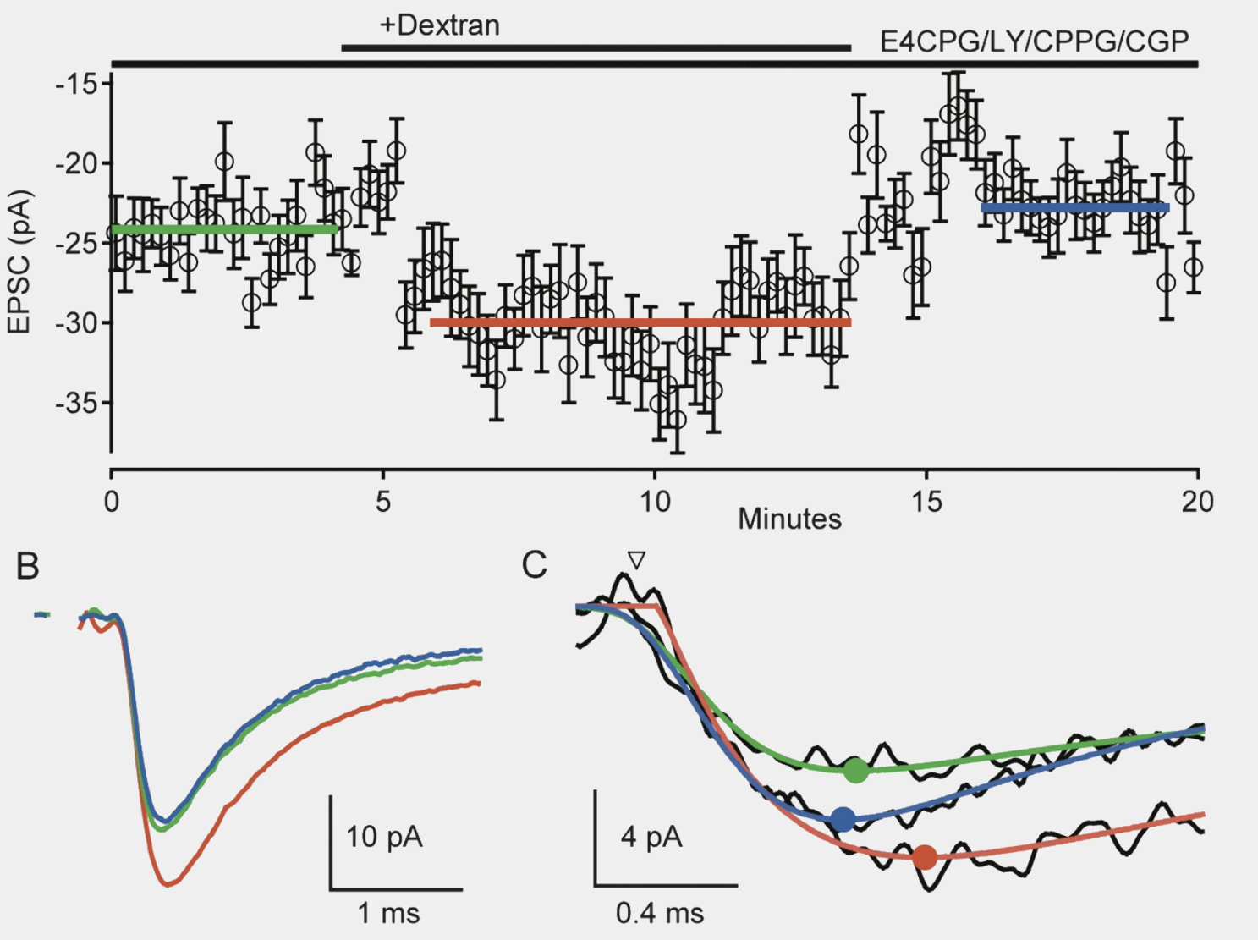
- One of the most difficult problems in purely functional programming languages is interacting with the real world. Functional reactive programming is a solution to this problem.

## Funding

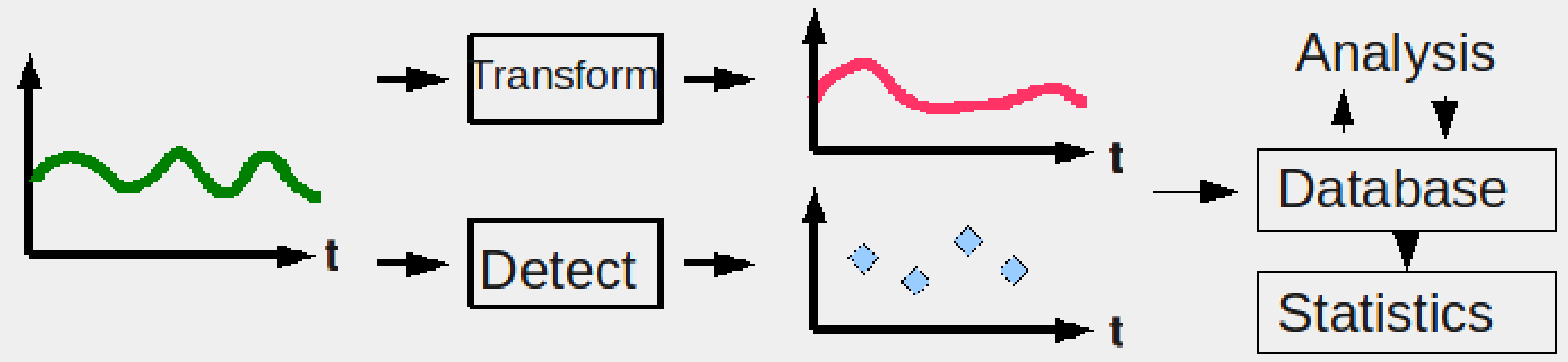
- BBSRC (TM), EPSRC (HN) and HFSP (TAN)

## Functional Reactive Programming

- Signal: for any type $\alpha$, $\boldsymbol{Signal}\ \alpha = \boldsymbol{Time} \rightarrow \alpha$
- Event: for any type $\alpha$, $\boldsymbol{Event}\ \alpha = [\,\boldsymbol{Time} \times \alpha\,]$
- Duration: for any type $\alpha$, $\boldsymbol{Duration}\ \alpha = [\,\boldsymbol{Time} \times \boldsymbol{Time} \times \alpha\,]$



| Quantity | Type |
|---|---|
| Membrane Voltage | $\boldsymbol{Signal}\ \mathbb{R}$ |
| Animal location | $\boldsymbol{Signal}\ (\mathbb{R} \times \mathbb{R})$ |
| Spike | $\boldsymbol{Event}\ ()$ |
| Spike waveforms | $\boldsymbol{Event}\ (\boldsymbol{Signal}\ \mathbb{R})$ |
| EPSC Amplitude | $\boldsymbol{Event}\ \mathbb{R}$ |
| Drug present | $\boldsymbol{Duration}\ ()$ |
| Visual stimulus | $\boldsymbol{Signal}\ \boldsymbol{Shape}$ |



## BAYSIG

- A new programming language we have built for neuroscience research
- Syntax and semantics similar to Haskell; the compiler is written in Haskell
- First-class signals
  - $\{: e :\}$ Signal that has value e at every timepoint
  - $\langle: s :\rangle$ Value of signal s at the current timepoint

$$\begin{aligned} \boldsymbol{smap} &:: (\alpha \rightarrow \beta) \rightarrow \boldsymbol{Signal}\ \alpha \rightarrow \boldsymbol{Signal}\ \beta \\ \boldsymbol{smap}\ \boldsymbol{f}\ \boldsymbol{s} &= \{: \boldsymbol{f} \langle: \boldsymbol{s} :\rangle :\} \end{aligned}$$

- First-class event streams
  - $\boldsymbol{pred}\ ??\ \boldsymbol{sig}$ An event that occurs whenever the predicate $\boldsymbol{pred}$ on signal $\boldsymbol{sig}$'s value becomes true.

$$\begin{aligned} \boldsymbol{crosses} &:: \mathbb{R} \rightarrow \boldsymbol{Signal}\ \mathbb{R} \rightarrow \boldsymbol{Event}\ () \\ \boldsymbol{crosses}\ \boldsymbol{x_{th}}\ \boldsymbol{sig} &= \boldsymbol{tag}\ ()\ ((\lambda \boldsymbol{x} \rightarrow \boldsymbol{x} > \boldsymbol{x_{th}})\ ??\ \boldsymbol{sig}) \end{aligned}$$

- Switch between different signals based on event occurrences

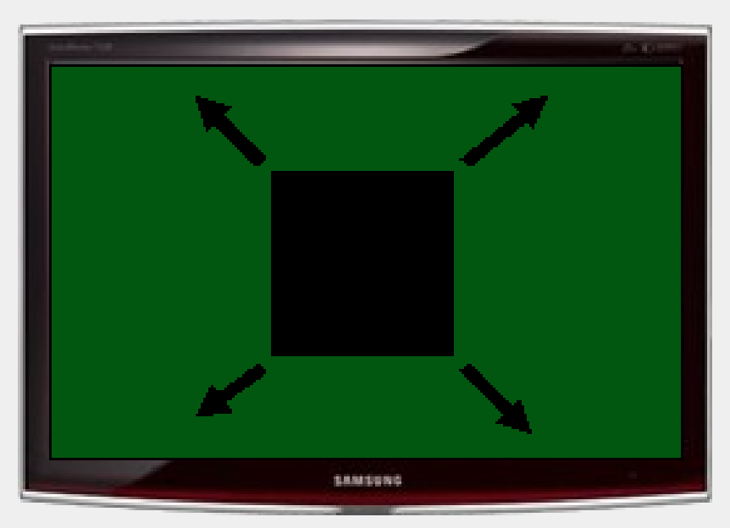
$$\begin{aligned} \boldsymbol{V_m} = \boldsymbol{switch} & \\ \_, \boldsymbol{end_{refrac}} &\rightsquigarrow \textbf{let}\ \boldsymbol{D}\ \boldsymbol{v} = \boldsymbol{cellOde}\ \boldsymbol{v}\ \textbf{in}\ \boldsymbol{v} \\ \boldsymbol{spike} &\rightsquigarrow \{: \boldsymbol{v_{rest}} :\} \end{aligned}$$

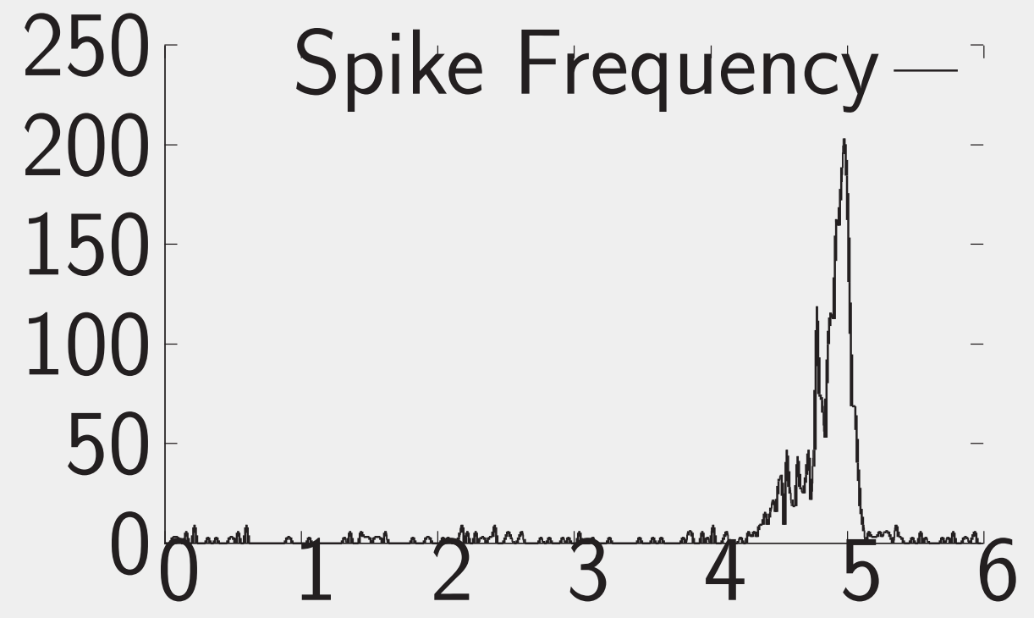
- Signals and events can be defined by mutual recursion

$$\begin{aligned} \boldsymbol{spike} &= \boldsymbol{crosses}\ (-0.05)\ \boldsymbol{V_m} \\ \boldsymbol{end_{refrac}} &= \boldsymbol{later}\ 0.002\ \boldsymbol{spike} \end{aligned}$$

- Bind *source* to variable
  $\boldsymbol{signalIn}\ \lessdot\!\ast\ \boldsymbol{source}\ \boldsymbol{parameter}$
- Send variable to *sink*
  $\boldsymbol{signalOut}\ \ast\!\gtrdot\ \boldsymbol{source}\ \boldsymbol{parameter}$
- *Remember* a value in a database for future analysis
  $\boldsymbol{x}! = \ldots$

## Looming detection in the desert Locust



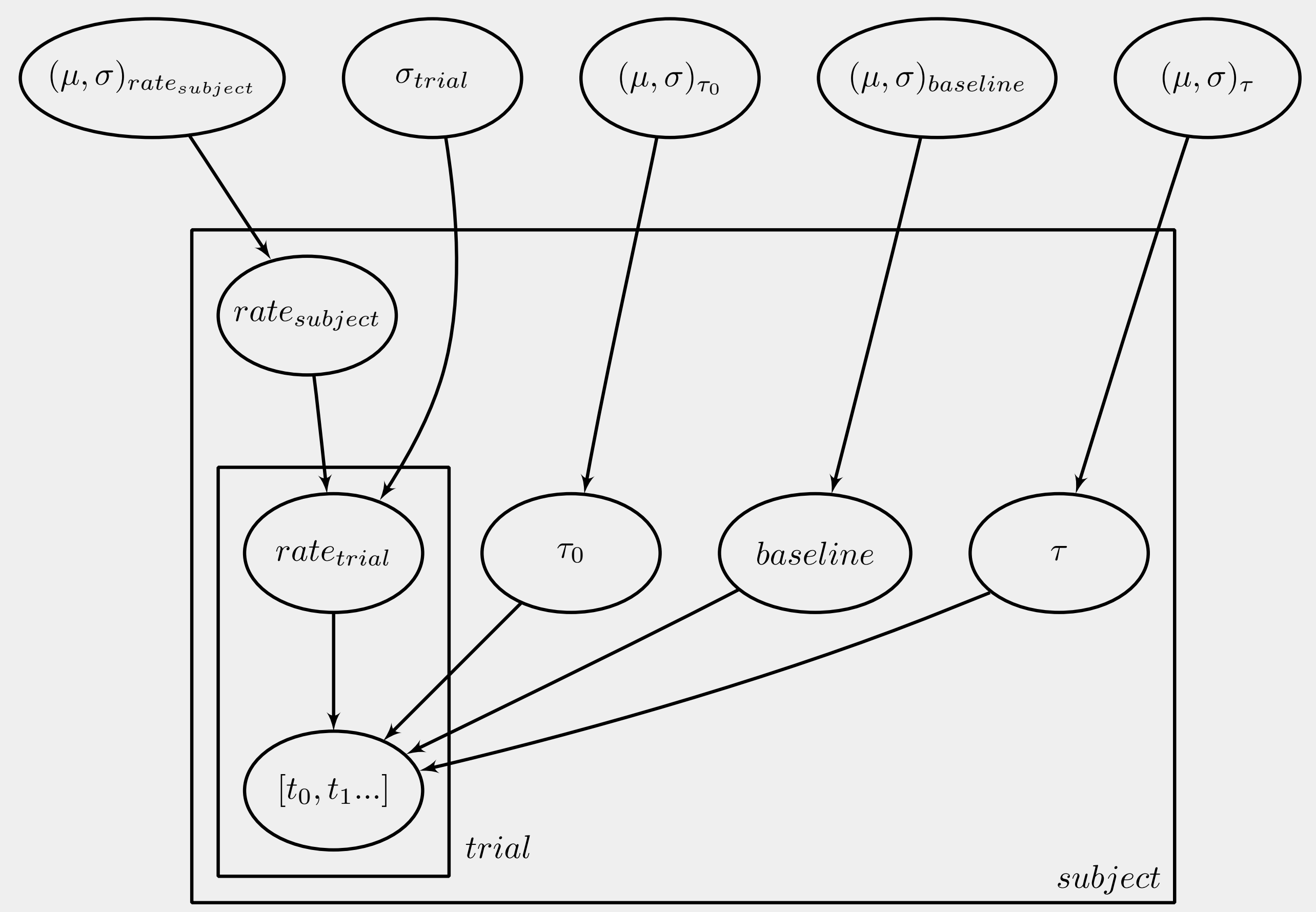


- BAYSIG experiment and analysis definition

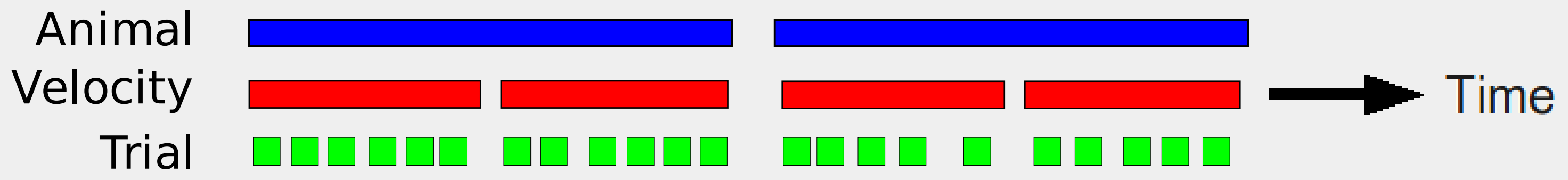
$$\begin{aligned} &\boldsymbol{ecVoltage}!\ \lessdot\!\ast\ \boldsymbol{ADC}\ (0, 20000, 6) \\ &\boldsymbol{distance} = \{: \boldsymbol{v} * \langle: \boldsymbol{seconds} :\rangle - 5 :\} \\ &\boldsymbol{loomingSquare} = \\ &\quad \{: \boldsymbol{colour}\ (0,0,0) \\ &\qquad\qquad (\boldsymbol{translate}\ (0, 0, \langle: \boldsymbol{distance} :\rangle)\ (\boldsymbol{cube}\ \boldsymbol{l}))\ :\} \\ &\boldsymbol{loomingSquare}\ \ast\!\gtrdot\ \boldsymbol{screen}\ () \\ &\boldsymbol{spikes}! = \boldsymbol{crosses}\ \boldsymbol{v_{th}}\ \boldsymbol{ecVoltage} \\ &\boldsymbol{histogram}! = \\ &\quad \{: \boldsymbol{length}\ (\boldsymbol{filter}\ (\boldsymbol{between} \langle: \boldsymbol{delay}\ \boldsymbol{seconds} :\rangle \\ &\qquad\qquad\qquad \langle: \boldsymbol{seconds} :\rangle \circ \boldsymbol{fst})\ \boldsymbol{spikes})\ :\} \end{aligned}$$

## Hierarchical Bayesian Modelling

- Statistical parameter estimation and hypothesis testing



- Bayes' theorem, hierarchically
  $\boldsymbol{p(\theta_{pop}|D) \propto p(D|\theta_{trial})p(\theta_{trial}|\theta_{subject})p(\theta_{subject}|\theta_{pop})p(\theta_{pop})}$
- Likelihood function
  $\boldsymbol{p(spikes|r(t)) = e^{-\int r(t)dt} \prod map\ (r \circ fst)\ spikes}$
- Defined in terms of *nested* durations



$\boldsymbol{running}$ ‘$\boldsymbol{within}$‘ $\boldsymbol{distinct}$ $\boldsymbol{velocity}$ ‘$\boldsymbol{within}$‘ $\boldsymbol{animal}$

## Conclusions

- Ontology based on lambda calculus and higher-order types
- We can directly express experiments and hierarchical models