Derivatives of Turing Machines and Inductive Inference

Daniel Murfet

University of Melbourne

based on joint work with J. Clift "Derivatives of Turing machines in Linear Logic" arXiv:1805.11813

therisingsea.org
Talk Outline

1. Derivatives of Algorithms?
   - Turing machines, Universal Turing machines
   - The Ehrhard-Regnier derivative

2. The problem of inductive inference / program synthesis
   - General problem statement
   - The probabilistic approach

3. Program synthesis via naive probability
What is an algorithm?

Def: A Turing Machine $M$ on alphabet $\Sigma$ and set of states $Q$ (both finite sets, with special symbols blank $\square \in \Sigma$ and $0,1 \in \Sigma$ and states $\text{START}, \text{HALT} \in Q$) is a function ($\delta$) $\delta : \Sigma \times Q \rightarrow \Sigma \times Q \times \{\text{LEFT, RIGHT, STAY}\}$ which describes the time evolution of a configuration

\[ \delta : \Sigma \times Q \rightarrow \Sigma \times Q \times \{\text{LEFT, RIGHT, STAY}\} \]

(\text{all but finitely many } \beta_i \text{ are blank})

\[ ((\beta_i)_{i \in \mathbb{Z}}, q) \]
What is an algorithm?

**Definition** Given a Turing machine $M$ we write $M(x) = y$ for $x, y \in \{0, 1\}^*$ if on a configuration $(x, \text{START})$ the machine reaches the configuration $(y, \text{HALT})$ after finitely many steps.

**Theorem (Turing)** There exists a Turing machine $U$ such that for every Turing machine $M$ there exists a string $c_u(M) \in \{0, 1\}^*$ such that for all $x \in \{0, 1\}^*$

- if $M(x) = y$ then $U(c_u(M), x) = y$, and
- if $M$ does not halt on input $x$, then $U$ does not halt on $(c_u(M), x)$

* Such a machine is called a **Universal Turing Machine (UTM)**.
What is an algorithm?

Example The Turing Machine $M$ has $\Sigma = \{0, 1\}$, $Q = \{\text{START, HALT}\}$

\[
\delta(x, \text{HALT}) = (x, \text{HALT, STAY})
\]

\[
\delta(0, \text{START}) = (1, \text{START, RIGHT})
\]

\[
\delta(1, \text{START}) = (0, \text{START, RIGHT})
\]

\[
M(0110) = 1001
\]

\[
M(1010) = 0101
\]

The code of this Turing machine is the string $c(M)$:

```
O START I START RIGHT 0 HALT 0 HALT STAY I START ...
```

and by choosing binary encodings we may assume $c(M) \in \{0, 1\}^*$. 

The Ehrhard-Regnier derivative

\[
\begin{align*}
\{ \text{Turing machines} \} & \xleftarrow{\text{Turing}} \{ \lambda\text{-calculus terms} \} \\
M & \xrightarrow{t(M)} t(M) \\
D_i t(M).u & \subseteq \{ \text{Differential } \lambda\text{-calculus terms} \} \\
D_i (\lambda x.(\lambda y.x)) \cdot u & \subseteq \{ \text{Differential } \lambda\text{-calculus terms} \}
\end{align*}
\]

Question: what do these derivatives mean? And what are they good for?

A semantic approach via Linear Logic

{ Turing machines } encode \{ linear logic proofs \} \{ differential linear logic proofs \} \rightarrow Vec_k

based on work of Girard

language of closed symmetric monoidal categories with cofree coalgebras

add dual number

Ehrhard-Regnier derivative

Sweedler semantics (based on cofree coalgebras)
Inductive Inference

• (Informal) Given several instances of a pattern, infer the pattern

• Given \((x_1, y_1), \ldots, (x_n, y_n)\) with \(x_i, y_i \in \{0, 1\}^*\), infer an algorithm \(M\) such that \(M(x_i) = y_i\) for \(1 \leq i \leq n\).

  "Inductive: we take \(M(x)\) as a prediction for unobserved \(x\)"

  - What do we mean by "algorithm"?
  - There are infinitely many such \(M\)'s \(\Rightarrow\) prefer simpler ones
  - But what does "simpler" mean?

  - For technical reasons we may also bound \(M\)'s runtime (i.e., we assume the pattern is effectively computable) and allow \(M(x_i) = y_i\)*.
Inductive Inference, formalised

- Every UTM \( U \) gives a "parametrisation" of the set of algorithms

  \[
  \{ \text{algorithms} \} := \{ \text{Turing machines} \} \xrightarrow{\varepsilon} \{0,1\}^*, \text{ i.e. codes}
  \]

- Kolmogorov, Solomonoff: algorithm \( M \) is simpler than algorithm \( N \) if

  \[
  |c_u(M)| < |c_u(N)|
  \]

- Inductive Inference: given \( E = \{(x_i, y_i, t_i)\}_{i=1}^{N} \) with \( x_i, y_i \in \{0,1\}^* \) and \( t_i \in \mathbb{N} \), find the shortest code \( C \in \{0,1\}^* \) such that \( U(C, x_i)^{t_i} = y_i^* \) for all \( 1 \leq i \leq N \). Call any such \( C \) a solution of \( E \). meaning a string with prefix \( y_i^* \)
Error propagation

• **Inductive Inference**: given $\mathcal{E} = \{(x_i, y_i, t_i)\}_{i=1}^N$ with $x_i, y_i \in \{0,1\}^*$ and $t_i \in \mathbb{N}$, find the shortest code $C \in \{0,1\}^*$ such that $U(C, x_i) = y_i$ for all $1 \leq i \leq N$. Call any such $C$ a solution of $\mathcal{E}$.

• **Logician's solution**: enumerate codes by length and try them one-by-one!

• **Probabilist's solution**: view $C = (C_0, C_1, \ldots)$ as a sequence of distributions over $\{0,1\}$ and vary $C$ so as to minimise a loss

$$L(C) = \sum_{i=1}^N D_{\text{KL}}\left(y_i \parallel \Delta\text{Step}_u^{t_i}(C, x_i)\right) + \text{regularisation}$$

KL divergence, propagation of uncertainty in $C$ through UTM
Error propagation

\[ \Delta \text{Step}_u \]

\( t_i \) time steps of UTM evolution (given some method of propagating uncertainty)

\[ L(C) = \sum_{i=1}^{N} D_{KL}(y_i \| p_i) + \ldots \]

\[ D_{KL}(q \| p) = \sum_{i \in \{0, 1\}} q_i \ln \left( \frac{q_i}{p_i} \right) \]

\( p, q \) distributions on \( \{0, 1\} \).
**Error propagation**

- **Question**: how to propagate error/uncertainty through a TM?

\[
\text{Def}^n \text{ Given a finite set } Z, \text{ let } \Delta Z := \{ \text{probability distributions over } Z \}. 
\]

\[
(\Delta \Sigma)^Z \times \Delta Q \xrightarrow{\Delta \text{step}_M} (\Delta \Sigma)^Z \times \Delta Q
\]

- i.e. what should this be?
Standard probability: sample from distributions representing state and tape squares, run the machine for \( t \) steps, amalgamate results

\[
\mathbb{P}( S(t+1) = q ) = \sum_{\delta \in \Sigma, q \in Q} \mathbb{P}( Y(t) = \delta \land S(t) = q' )
\]

Suppose the code \( C = (\alpha_0, \alpha_1, \alpha_2, \ldots) \) depends on \( \alpha_j \) so

\[
\frac{\partial}{\partial x_j} L(C) = \sum_{i=1}^{2^n} \frac{\partial}{\partial x_j} D_{KL}\left( y_i \parallel \Delta_{\text{step}^t_x}(C, x_i) \right) + \ldots
\]

Unfortunately, \( \nabla L \) is useless for solving the inductive inference problem!
Naive probability we can define an alternative $\Delta$-step propagating uncertainty through a TM, by making certain "naive" conditional independence hypotheses, e.g.

$$P(S(t+1) = q) = \sum_{\delta \in \Sigma, \gamma \in \mathbb{Q}} P(Y_0(t) = \delta) P(S(t) = q')$$

state at time $t+1$ \hspace{1cm} \delta(2, q') = (?, q, ?) \hspace{1cm} symbol under head \hspace{1cm} state.

**Theorem (Clift-M)** The propagation of naive probability

$\Delta \text{step}_M : (\Delta \Sigma)^\mathbb{Z} \times \Delta \mathbb{Q} \rightarrow (\Delta \Sigma)^\mathbb{Z} \times \Delta \mathbb{Q}$

arises from the denotational semantics of linear logic,

and the Ehrhard-Regnier derivative of $M$ computes $T(\Delta \text{step}_M)$. 
References


• (Microsoft) A. Gaunt et. al “TerpreT: a probabilistic programming language for program induction” 2016.
The Ehrhard-Regnier derivative

\[
\{\text{Turing machines}\} \xleftarrow{\text{Turing}} \{\text{\(\lambda\)-calculus terms}\}
\]

\[
M \mapsto t(M)
\]

\[
D_i t(M) \cdot u \quad D_i \subseteq \{\text{Differential \(\lambda\)-calculus terms}\}
\]

\[
D_1 (\lambda x. (\lambda y. x)) \cdot u, \ldots
\]

**Question:** what do these derivatives mean? And what are they good for?

- They compute rates of change of naive probability.
- (Potentially) program synthesis computed by Ehrhard-Regnier.