Operatorski račun nad programskimi prostori

Žiga Sajovic

14. september 2017

Univerza v Ljubljani, Fakulteta za Računlništvo in Informatiko

Tematika naloge

Razvijte algebraični jezik, ki bo deloval kot formalni račun za globoko učenje, hkrati pa bo orodje za proučevanje programov, ki so v njem implementirani. Jezik naj deluje kot poln model globokega učenja. Omogoča naj tako izražanje programov, da bo že njihov zapis nudil teoretični vpogled vanje.

Reševane pomanjkljivosti

Pomanjkljivosti

Z delom smo reševali dve družini pomankljivosti:

- pomanjkljivosti algebre jezikov
- pomanjkljivosti globokega učenja

ki ju pred začetkom utemeljimo z mnenji strokovnjakov obeh področij.

Pomanjkljivosti algebre jezikov

> *"Von Neumann languages do not have useful properties for reasoning about programs. Axiomatic and denotational semantics are precise tools for describing and understanding conventional programs, but they only talk about them and cannot alter their ungainly properties. Unlike von Neumann languages, the language of ordinary algebra is suitable both for stating its laws and for transforming an equation into its solution, all within the language."*
> *– John Backus, Can Programming Be Liberated From the von Neumann Style?*

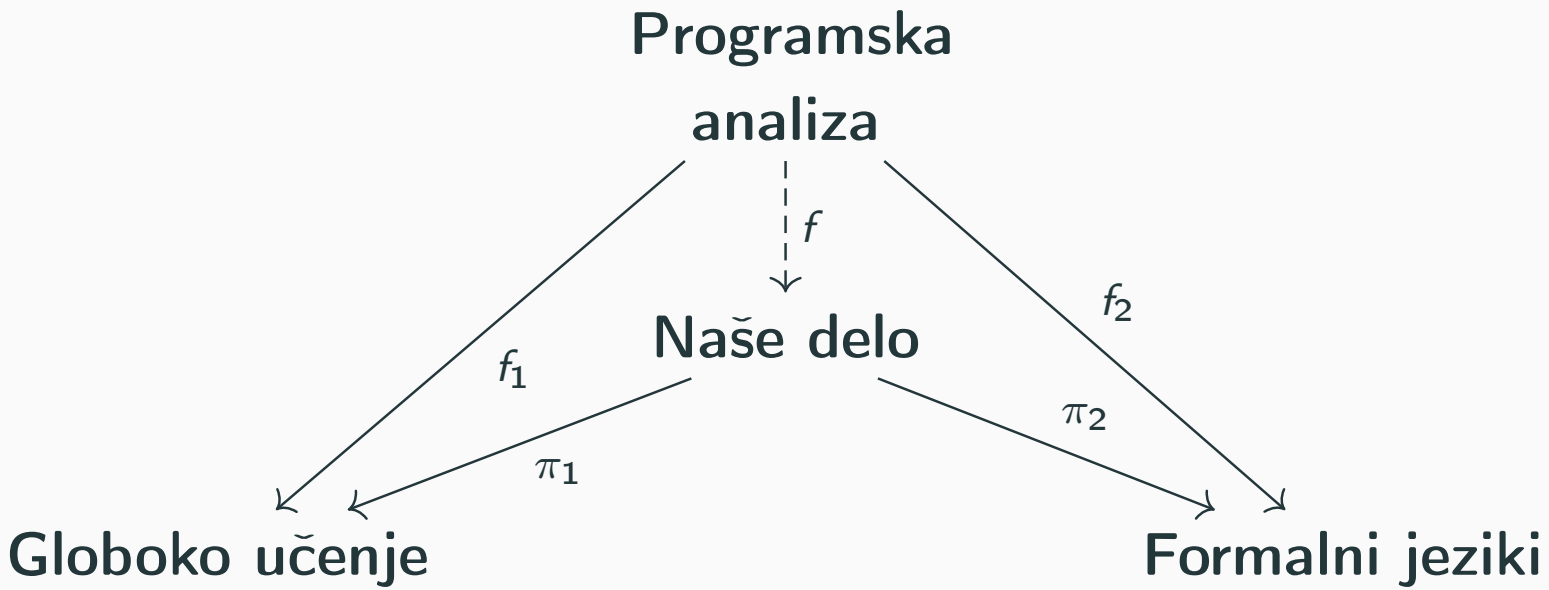
Obstoječi modeli nam ne omogočajo algebraičnega manipuliranja odvedljivih programov.

Pomanjkljivosti globokega učenja

"... There is no proper definition of what deep learning even is at this stage. Your [avtorjeva] theory of Operational Calculus on Programming Spaces [naše delo] could offer a first such definition, and a tool for theoretic investigations, through the formalism you [avtor] propose."
– dr. Leon Buttou, v korespondenci z avtorjem

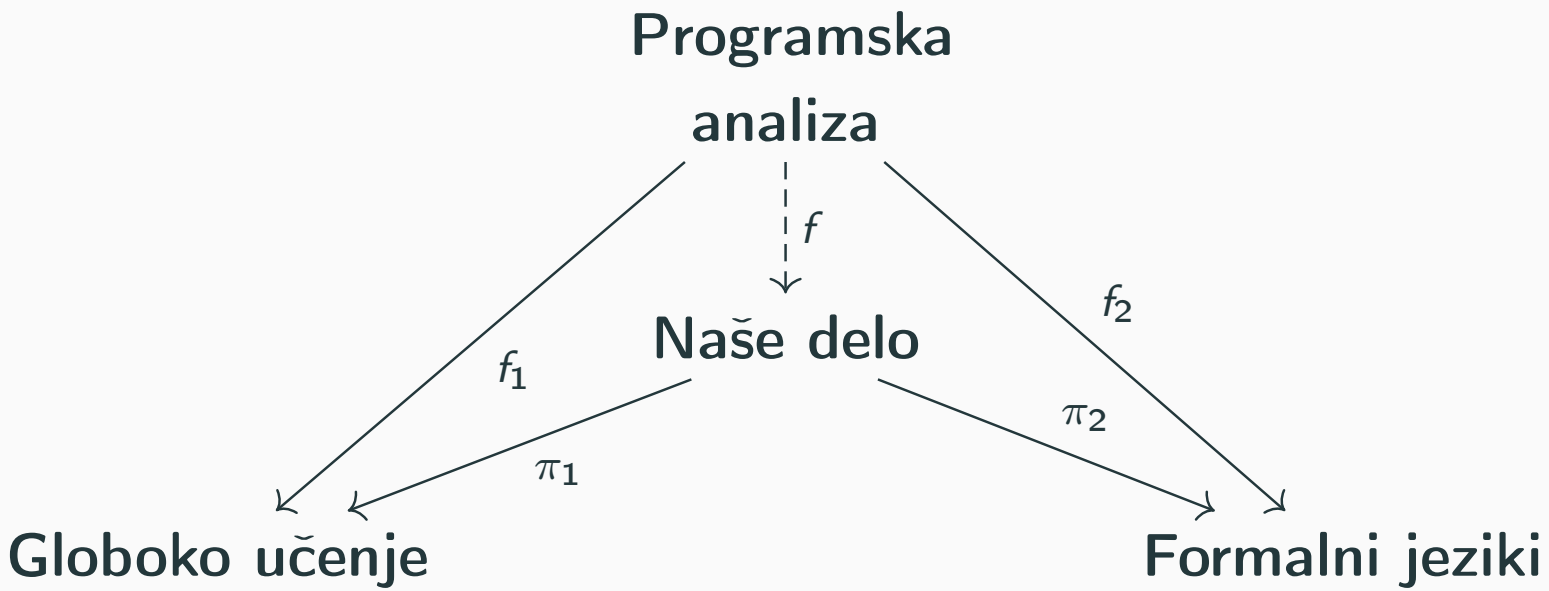
Zaradi neobstoja ustreznega formalnega aparata preiskovanja področja potekajo predvsem empirično.

Komutativen diagram



Naše delo hkrati rešuje tako pomanjkljivost algebre jezikov kot pomanjkanje formalizma v globokem učenju, in sicer tako, da zgornji diagram komutira.

Komutativen diagram



Tako je na delo mogoče gledati, kot na produkt globokega učenja in formalnih jezikov v kategoriji programske analize.

Programski prostori

Programski prostor

Programe bomo modelirali kot preslikave vektorskih prostorov, vase. Če se osredotočimo na realne spremenljivke (tipa float ali double), je trenutno stanje pomnilnika predstavljivo z visoko-dimenzionalnim vektorjem. Množico vseh možnih stanj pomnilnika lahko modeliramo s končno-dimenzionalnim vektorskim prostorom $\mathcal{V} \simeq \mathbb{R}^n$ (podobno kot pri *nevronskih Turingovih strojih*).

Programski prostor **je prostor vseh preslikav $\mathcal{V} \to \mathcal{V}$, ki se dajo zapisati v obliki programa v izbranem programskem jeziku.**

Evklidski stroj

Definicija (Evklidski stroj)
Par $(\mathcal{V}, \mathcal{F})$, kjer sta

- *$\mathcal{V}$ končno-dimenzionalni vektorski prostor nad obsegom K,*
- *$\mathcal{F} < \mathcal{V}^{\mathcal{V}}$ podprostor prostora $\mathcal{V}^{\mathcal{V}}$ vseh preslikav $\mathcal{V} \to \mathcal{V}$,*

imenujemo Evklidski stroj (z akcijami iz $\mathcal{F}$ nad $\mathcal{V}$).

Na prvi pogled se Evklidski stroj zdi kot opis funkcijskega programiranja s kompozicijami, podedovanimi od $\mathcal{F}$, a ob zajemanju prednosti funkcijskega programiranja ponuja tudi druge prednosti.

Odvedljivi programski prostori

Navidezni pomnilniški prostor

Pomnilniški prostor programa je redko obravnavan kot kaj več kot zgolj shramba. Da pa bi obogatili Evklidski stroj z dodano strukturo, se osredotočimo prav nanj. Ohlapno rečeno je funkcijsko programiranje opisano z monoidi (grupami brez inverzov), zato se (multi)linearno algebraičen opis pomnilnika zdi primeren korak k pridobitvi dodatne strukture.

Navidezni pomnilniški prostor je tenzorski produkt pomnilniškega prostora $\mathcal{V}$ s tenzorsko algebro njegovega duala $T(\mathcal{V}^*)$.

$$\mathcal{V}_\infty = \mathcal{V} \otimes T(\mathcal{V}^*) = \mathcal{V} \oplus (\mathcal{V} \otimes \mathcal{V}^*) \oplus \dots$$

Odvedljiv programski prostor

Definicija (Odvedljiv programski prostor)

Odvedljiv programski prostor $\mathcal{P}_0$ je vsak podprostor prostora $\mathcal{F}_0$, kjer

$$\partial\mathcal{P}_0 \subset \mathcal{P}_0 \otimes T(\mathcal{V}^*).$$

Prostor $\mathcal{P}_n < \mathcal{F}_n$, ki ga napenja $\{\partial^k\mathcal{P}_0; \quad 0 \leq k \leq n\}$ nad K, imenujemo odvedljiv programski prostor reda n. Ko so vsi elementi $\mathcal{P}_0$ analitični, $\mathcal{P}_0$ imenujemo analitični programski prostor.

Odvedljiv programski prostor

Izrek (Neskončna odvedljivost)

Vsak odvedljiv programski prostor $\mathcal{P}_0$ je neskončnokrat odvedljiv programski prostor, kar pomeni,

$$\partial^k \mathcal{P}_0 \subset \mathcal{P}_0 \otimes T(\mathcal{V}^*)$$

za vsak $k \in \mathbb{N}$.

Posledica

Odvedljiv programski prostor reda n, $\mathcal{P}_n : \mathcal{V} \to \mathcal{V} \otimes T(\mathcal{V}^*)$, lahko vložimo v tenzorski produkt funkcijskega prostora $\mathcal{P}_0$ in prostora $T_n(\mathcal{V}^*)$ multi-tenzorjev reda n ali manj:

$$\mathcal{P}_n < \mathcal{P}_0 \otimes T_n(\mathcal{V}^*).$$

Navidezni tenzorski stroji

Iz Izreka o Neskončni odvedljivosti in njegovi Posledici sledi, da par $(\mathcal{V}, \mathcal{P}_0)$ – skupaj s strukturo tenzorske algebre $T(\mathcal{V}^*)$ – zadošča za konstrukcijo odvedljivih programskih prostorov $\mathcal{P}_\infty$ z uporabo linearne kombinacije elementov $\mathcal{P}_0 \otimes T(\mathcal{V}^*)$.

Par $(\mathcal{V}, \mathcal{P}_0)$, skupaj s strukturo tenzorske algebre $T(\mathcal{V}^*)$, poimenujemo navidezni tenzorski stroj, v katerem so osnovni programi tenzorski mreže

$$\mathcal{N}(v) = \phi_k \circ W_k \circ \cdots \circ \phi_0 \circ W_0(v),$$

in predstavlja poln model globokega učenja.

Operatorski račun nad programskimi prostori

Razvoj v tenzorsko vrsto

Naslednje, kar naš model potrebuje, je operator, ki bi prestavil vrednost funkcije iz njene začetne vrednosti. **Tak operator bi lahko kasneje uporabili za implementacijo iteratorjev in komponerjev.**

Razvoj v tenzorsko vrsto

Takšen operator definiramo kot

$$e^{h\partial} = \sum_{n=0}^{\infty} \frac{(h\partial)^n}{n!}$$

Torej je operator $e^{h\partial}$ preslikava med programskimi prostori

$$e^{h\partial} : \mathcal{P} \to \mathcal{P}_\infty.$$

Hkrati pa definira preslikavo

$$e^{h\partial} : \mathcal{P} \times \mathcal{V} \to \mathcal{V} \otimes \mathcal{T}(\mathcal{V}^*),$$

iz prostora programov v prostor multi-linearnih preslikav.

Razvoj v tenzorsko vrsto

Izrek (Razvoj v tenzorsko vrsto)

Za program $P \in \mathcal{P}$ se razvoj v neskončno tenzorsko vrsto v točki $\mathbf{v}_0 \in \mathcal{V}$ izraža z več kontrakcijami

$$P(\mathbf{v}_0 + h\mathbf{v}) = \Big((e^{h\partial} P)(\mathbf{v}_0)\Big)(\mathbf{v}) = \sum_{n=0}^{\infty} \frac{h^n}{n!} \partial^n P(\mathbf{v}_0) \cdot (\mathbf{v}^{\otimes n})$$

$$= \sum_{n=0}^{\infty} \frac{h^n}{n!} \sum_{\forall i,\alpha} \frac{\partial^n P_i}{\partial x_{\alpha_1} \dots \partial x_{\alpha_n}} \mathbf{e}_i \cdot dx_{\alpha_1}(\mathbf{v}) \cdot \ldots \cdot dx_{\alpha_n}(\mathbf{v}).$$

Uporaba

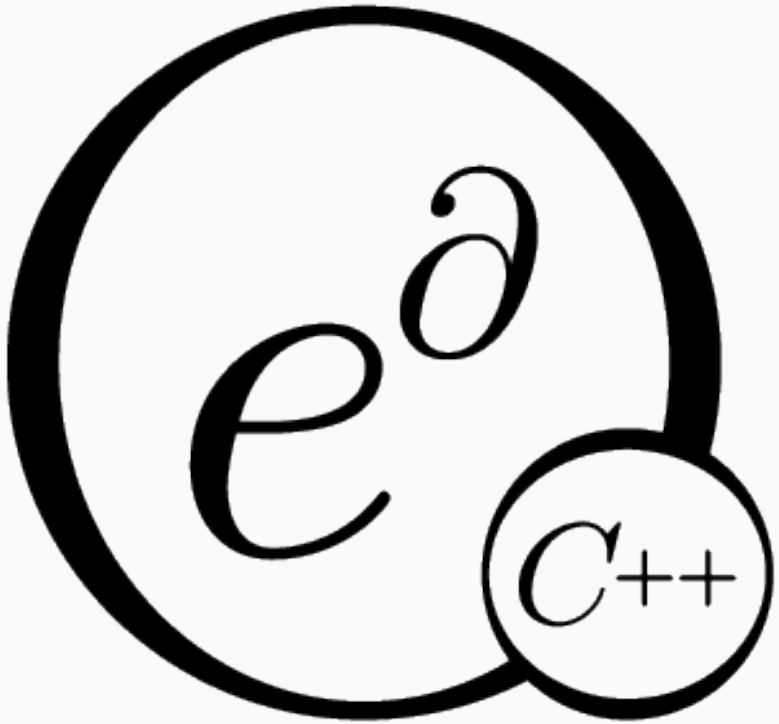
Predstavljena teorija ponuja več zmožnosti:

- algebraične manipulacije programov
- uporaba operatorjev kot oblika abstrakcije
- izpeljave teoretično utemeljenih modelov globokega učenja
- poenotenje teorije avtomatskega odvajanja
- funkcijske transformacije in prevedbe programov
- itd.

Te zmožnosti so predstavljene v diplomskem delu in avtorjevih člankih [1, 2].

Implementacija

GitHub



Odprtokodna implementacija odvedljive različice jezika *C++* je dostopna na avtorjevi strani. [3, 4]

Literatura


Žiga Sajovic and Martin Vuk.
Operational calculus on programming spaces.
arXiv:1610.07690, 2016.


Žiga Sajovic.
Implementing operational calculus on programming spaces.
arXiv:1612.0273, 2016.


Žiga Sajovic.
dcpp: Infinite differentiability of conditionals, recursion and all things c++, 2016.
https://zigasajovic.github.io/dCpp/.


Žiga Sajovic.
dcpp, 2016.
https://github.com/zigasajovic/dCpp.