Dodělat název (...)

Tomáš Křen

Roman Neruda

***Abstract*— Dodělat abstrakt (...)**

I. Introduction

DODĚLAT úvod (...) aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc aaa bbb ccc.

Jak to celý pojmout?

Pokud chceme dělat GP nad stromama větší vyjadřovací síly než maj klasický S-výrazy, tak máme v zásе dvě přirozený možnosti pro obecný řešení: buď to dělat celý v (polymorfních) kombinátorech od začátku (a vyhnout se tak proměnejm a lambda abstrakcím), nebo to skusit v lambda kalkulu a nejak se vypořádat s proměnejma a lambdama. Motivace pro práci s lambda termama je, že za nima košatá teorie, která například popisuje redukce lambda termů (ty zajišťujou jak zmenšení samotných stromu, tak to, že prohledávací prostor se zmenší, díky tomu, že se různé stromy redukují na ten samej).

Jak se vypořádat s proměnejma a lambdama? (aka jak křížit)

Po vygenerování převod do kombinátorů pomocí eliminace abstrakcí (tak jak se to dělá v diplomce). To ale působí v něčem neohrabaně, když to porovnáme s generováním přímo v kombinatorech, když vezmeme v potaz to že eliminace abstrakcí má za důsledek až kvadratickej nárůst stromu - čili to co sme nahnali na redukcích stratíme eliminací.

Jedince držíme jako redukovane malinké-kompaktní-a-elegantní lambda termy. Ve chvíly kdy křížíme provedeme na obou rodičích eliminaci abstrakcí (která ubere proměny a lambdy a namísto toho tam dá kombinátory S,K,I případně i další při fikanějších eliminacích), tím nám sice narostou, ale my z toho máme jedine radost, protože tím se nám zvýšil počet míst ke křížení (což se ukazuje jako dobráv věc, viz s-expr reprezentace vs @-tree reprezentace lambda termů). Po skřížení se vložený kombinátory nahradí odpovídajícim lambda termem (tzn např všude kde je K dám (x y . x) atd) výslednej term zredukuju a dostávam zase malinké-kompaktní-a-elegantní dítě. Nevýhoda toho zahrnout do článku i tohle je v tom, že k tomu nemám ještě žádný pokusy - ale k tomu zbytku mám upřímě v zato taky dost ubohý pokysy, takže toho bych se asi nebál. Většinu potřebnýho kodu bych k tomu ale měl už mít víceméně hotovou, takže pokud by se to na něčem nezaseklo, tak myslim že je realný udělat i pokus do toho dvacátýho. Zvlášť přitažlivý mi tohle křížení příde i kvůli tomu, že v přírodě se taky rozbalujou a zabaloujou chromozomi při meioze/mitoze.

II. Related work

Vzit asi dost stejny jako při minulym článku, vyzvednout kombinátoři a barendrechta.

III. Preliminaries

Definice lambda kalkulu tentokrát bez nutnosti do toho extra podrobně zavádět typy.

Redukce

Eliminaci abstrakcí

—

Jak reprezentovat stromy programů pro GP?

Reprezentace v klasickym kozovy je S-expression.

Nebo můžem používat kombinátory jako Briggs a O'Neil (to si myslim je jakoby hlavní konkurence, vůči který by to chtělo obhájit)

Lambda termy a jejich awesomeness

Povídaní o redukcích

Povídání o lnf

že lnf je přirozený rozšíření s-exprešnů do lambda kalkulu

vlastnosti termů v lnf

proč je eta redukovat (eta redukcí lnf dostanem beta-eta-nf a že nemusíme beta redukovat pač se to tim nerozbyje)

Generování

Generování gramatikou

Gramatika pro lnf

Inhabitation trees jako intuitivní model takovýhodle lnf generování

Problémy s proměnýma a jak je řešit.

Křížit lambda termy i s proměnejma a abstrakcema, problémy řešit když nastanou (pomluvit a odsoudit)

Zmenšit prostor termů (tim že někerý nejsme schopný vygenerovat) s kterým operujeme tak aby křížení už nebyl problém (to dělá Yu - v těle lambda fce dovoluje jen použití proměnných z její hlavy)

Převod hned po vygenerování

Převod až při křížení

eliminace abstrakcí

skřížim

vložený kombinátory nahradim odpovídajícím termem

celý to redukuju

IV. Our approach

A. Introduction

B. Algorithm

V. Experiments

VI. Conclusions

Dodělat závěr (...)

REFERENCES

[1] TODO. Todo, *Dát sem místo toho ten bibtex*. MIT Press, Cambridge, MA, 2014.

[2] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[3] Koza, J.R., Keane, M., Streeter, M., Mydlowec, W.,Yu, J., Lanza, G. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Springer, 2005. ISBN 978-0-387-26417-2

[4] Riccardo Poli, William B. Langdon, Nicholas F. McPhee *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[5] T. Yu. *Hierachical processing for evolving recursive and modular programs using higher order functions and lambda abstractions*. Genetic Programming and Evolvable Machines, 2(4):345–380, December 2001. ISSN 1389-2576.

[6] D. J. Montana. *Strongly typed genetic programming.* Evolutionary Computation, 3(2): 199–230, 1995.

[7] T. D. Haynes, D. A. Schoenefeld, and R. L. Wainwright. *Type inheritance in strongly typed genetic programming.* In P. J. Angeline and K. E. Kinnear, Jr., editors, Advances in Genetic Programming 2, chapter 18, pages 359–376. MIT Press, Cambridge, MA, USA, 1996. ISBN 0-262-01158-1.

[8] J. R. Olsson. *Inductive functional programming using incremental program transformation and Execution of logic programs by iterative-deepening A* SLD-tree search.* Dr scient thesis, University of Oslo, Norway, 1994.

[9] Forrest Briggs, Melissa O'Neill. *Functional Genetic Programming and Exhaustive Program Search with Combinator Expressions.* International Journal of Knowledge-based and Intelligent Engineering Systems, Volume 12 Issue 1, Pages 47-68, January 2008.

[10] H. P. Barendregt, *The Lambda Calculus: its Syntax and Semantics*, revised ed., North-Holland, 1984.

[11] H. Barendregt , S. Abramsky , D. M. Gabbay , T. S. E. Maibaum. *Lambda Calculi with Types.* Handbook of Logic in Computer Science, 1992.

[12] Henk Barendregt, Wil Dekkers, Richard Statman, *Lambda Calculus With Types*. Cambridge University Press, 2010.

[13] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall, 1987.

[14] Stuart J. Russell, Peter Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.