Assignment 6

15-816: Substructural Logics
Frank Pfenning

Due
Tuesday, November 15, 2016

This assignment consists of several, somewhat open-ended problems. **You should pick one of them to do.** If you would like to do a half-semester project instead, please submit your project proposal instead of the assignment. A project proposal should be 2–5 pages and map out background, motivation, technical approach, and expected outcome of your project.

You may do this assignment **by yourself or in pairs**. They are somewhat open-ended, so you have to use your judgment as to when you consider the homework completed. Feel free to contact the instructor when you have questions about the extent of a problem.

As usual, you are allowed and encouraged to use *all* resources (papers, lecture notes, technical reports) that you can find, but you must properly cite and acknowledge any resources you use.

Please submit this assignment as a PDF by email. LaTeX templates and macros that may be helpful are available on the course web pages, but you are not required to use them.

Exercise 1 (Nondeterminism) Consider a nondeterministic extension of subsingleton logic where multiple branches of a caseL and caseR expression can have the same label. Follow our typical methodology and assess identity expansion, cut reduction, and rules of computation derived from them. Can we use this to faithfully model nondeterministic versions of finite automata, pushdown automata, and Turing machines? Explore the consequences of generalizing even further to allow redundant labels (not mentioned in the type) or missing labels (required by the type, but without a matching branch).

Exercise 2 (Full Ordered Focusing) Prove full focusing for ordered logic by adapting Simmons's structural focalization technique [Sim14]. You should

present inversion in a deterministic manner and avoid the use of proof terms for the purpose of this exercise. You may assume the correctness of polarization, so that the result comes down the a completeness theorem as well as admissibility of cut and identity in the focused sequent calculus.

Exercise 3 (Ordered Theorem Proving) Build several theorem provers for ordered logic in an implementation language of your choice. The different theorem provers should integrate different techniques for limiting nondeterminism in proof search so you can compare the results. Create a suite of test problems, both provable and not provable, drawing from two domains: internal laws such as associativity, distributivity, idempotence and similar properties, and grammars and parsing problems presented as proposed by Lambek [Lam58].

Exercise 4 (Concurrent Data Structure Library) Develop a small library of concurrent data structures in either SILL[1] or Concurrent C0[2]. Examples could range from various forms of heaps, to binary trees of some form (e.g., treaps), or other data structures and concurrent algorithms of your choice. Make sure your code includes some nontrivial clients for illustration and testing purposes. Do not be concerned with practical efficiency, but analyze parallel complexity (reaction time, latency, throughput, work) where you can.

Exercise 5 (Strict Logic) Strict logic is the stepchild of substructural logics and does not seem to get much attention. Strict logic is easily defined as a substructural logic in which we have contraction but not weakening. In other words, every antecedent must be use at least once. Give a sequent calculus for strict logic which is amenable to a structural proof of the admissibility of cut and show the key cases in the proof. Are there any surprising interactions once you integrate it with linearity using shifts? How do the available connectives compare to linear and structural logics? Further, consider and discuss either focusing (conjecture the rules, which you do not need to prove) or a concurrent operational semantics.

Exercise 6 (Garbage Collection) In the purely ordered and linear process calculi, no garbage collection is necessary (even though a looping process may prevent termination and overall freeing of resources). If we add an

[1] a functional implementation of session-typed concurrency [Gri16] in OCaml, available at `https://github.com/ISANobody/sill`

[2] an imperative implementation of session-typed concurrency [WPP16], available at `https://svn.concert.cs.cmu.edu/c0`

affine mode F, processes that no longer have a client should be explicitly terminated. Develop a clean type assignment system and operational semantics such that upon termination of an initial process of type $\cdot \vdash \mathit{main} :: (c_0 : \mathbf{1})$ the process configuration is guaranteed to clean up all resources. Sketch how your design might be extended to a reference-counted garbage collector for an extension with a structural mode U with the only propositions of this mode being $\uparrow^{\mathsf{U}}_{\mathsf{F}} A_{\mathsf{F}}$, or explain why it would not work.

References

[Gri16] Dennis Griffith. *Polarized Substructural Session Types*. PhD thesis, University of Illinois at Urbana-Champaign, April 2016. In preparation.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170, 1958.

[Sim14] Robert J. Simmons. Structural focalization. *ACM Transactions on Computational Logic*, 15(3):21:1–21:33, 2014.

[WPP16] Max Willsey, Rokhini Prabhu, and Frank Pfenning. Design and implementation of Concurrent C0. In *Fourth International Workshop on Linearity*, June 2016.