Graduate Programming Languages, Write a Technical Perspective
Due: Three Deadlines As Described Below

Introduction: In recent years, the magazine *Communications of the ACM* (`http://cacm.acm.org/magazines/`) has republished two computer-science research articles in each issue. It can be difficult for a typical computer scientist to appreciate cutting-edge research because it may use unusual terminology/notation or rely on other results that are not widely known. Therefore, each paper is preceded by a "Technical Perspective," which is written by an expert other than the authors. *This short summary explains the importance of the research and the particular contributions of the paper.*

We are going to simulate this experience, with you in the role of the Technical Perspective author. This assignment will involve three stages, *each of which is more challenging than it may appear*:

- Choose an appropriate research paper
- Understand the paper and identify the paper's contributions over previous research
- Write a 1–3 page technical perspective

Stage 1: Choosing a Paper
Due to-be-determined-approximately-14-days-before-end-of-term (earlier encouraged)

This assignment includes a list of papers from which you can choose. Most students will probably select from this list, but doing so is not required. This list has the following significant biases:

1. Concepts and technical machinery from class are *necessary* to understand them.
2. Concepts and technical machinery from class are *not sufficient* to understand them. That is, you will likely need to learn additional concepts on your own via additional reading.
3. Recent papers (the last few years) on topics of increasing importance (concurrency, scripting languages) are over-represented.

There are many other papers that could have been included even given these biases.

Choose a paper by reading the abstracts and skimming or reading papers that sound interesting. Do not panic if most of a paper is impenetrable on first reading — see Stage 2.

You may choose a paper not on the list, *with instructor permission before the due date.* You can either find a paper on your own, by skimming the proceedings of programming-languages research conferences, or you can work with the instructor to find a paper on a topic that interests you. Note, however, that biases 1 and 2 above are essential — you need a paper that at least indirectly relies on formal semantics, type systems, or some other topic in the course. In short, pick a paper that needs knowledge from the course.

What if your friend wants the same paper: This is an individual assignment. If multiple class members choose the same paper, that's fine, *but* then you *cannot* work together or discuss the paper. On the other hand, if your friend chooses a different paper, then you *are* allowed to discuss your papers together and even proofread and provide suggestions for your technical perspectives. Therefore, there is some incentive to coordinate with a friend or two to avoid picking the same paper.

Turn-in / Grading: Submit your choice to the course staff. If you change your choice later, your grade for the assignment will be multiplied by 0.9, i.e., there is a 10% penalty. This policy is to motivate you to start early and to choose a paper that you have looked at enough to have some confidence that you will not regret your selection.

Stage 2: Understanding Your Paper and Its Contributions
Due to-be-determined-approximately-5-days-before-end-of-term (earlier encouraged)

You have two goals in this stage:

- Thoroughly understand the paper
- Understand what this paper contributed to human knowledge

While the course has given you a solid foundation in programming-language semantics, a gap remains between the classic concepts you have learned and the state-of-the-art. In short, you are unlikely to be able to read your paper front-to-back. To find appropriate background reading, consider several strategies:

- Your paper cites previous papers. Identify which of those are most likely to provide the background you need. Continue following references transitively until you find what you need.
- Search the web for tutorials and explanations.
- Ask the course staff questions about specific topics. Good questions would be, "What is an open class?" or "Do you know any tutorials on monads?" or "I understand Section 3.1 is about X but then the first sentence of Section 3.2 is completely opaque – can you help?"

Turn-in / Grading: Submit whatever you want provided that:

- It is approximately one page, and definitely not more than two.
- It makes a convincing case that you have read the paper and understand the vast majority of it.

An outline of the paper and list of contributions is a natural approach. It is not necessary to use complete sentences. You might also list what other papers and references you found most useful.

As with Stage 1, this won't really be graded, but you will receive a 10% penalty for not getting it done by the deadline. This is for your protection, to avoid any suggestion that it is possible to do a good job on your technical perspective without sufficient time *after* you understand the paper.

Stage 3: Write Your Technical Perspective
Due to-be-determined-approximately-end-of-term

The technical perspective must be **more than 1 and at most 3 pages**, single-spaced, single-column. Writing concisely should be more difficult than writing a longer paper. Treasure your reader's time, with each sentence being interesting and essential. Convey all the main ideas and contributions of the paper.

The *pretend audience* is a senior studying computer science who has not taken the course. That is, you can assume your audience is a decent programmer with a good education, but you should be very wary of jargon or technology that would be known only to programming-languages experts. In contrast, the paper you are writing about *does* make such assumptions, since it was written for a more expert audience. Hence your technical perspective is providing real value by making the ideas in the work more accessible.

The *actual audience* is the course staff. They want to see that the course has given you the ability (1) to learn more about programming-languages research and (2) to communicate what you learn to others.

Turn-in / Grading: Submit your paper, preferably as a PDF document, to the course staff.

Suggested Papers (alphabetical by first author)

1. Virtual values for language extension
   Thomas H. Austin, Tim Disney, Cormac Flanagan
   ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, 2011
   http://dx.doi.org/10.1145/2048066.2048136

2. A Type and Effect System for Deterministic Parallel Java
   Robert L. Bocchino, Jr., Vikram S. Adve, Danny Dig, Sarita V. Adve, Stephen Heumann, Rakesh Komuravelli, Jeffrey Overbey, Patrick Simmons, Hyojin Sung, Mohsen Vakilian
   ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, 2009
   http://dx.doi.org/10.1145/1640089.1640097

3. MultiJava: Modular Open Classes and Symmetric Multiple Dispatch for Java
   Curtis Clifton, Gary T. Leavens, Craig Chambers, Todd Millstein
   ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, 2000
   http://dx.doi.org/10.1145/354222.353181

4. Transactional Events
   Kevin Donnelly, Matthew Fluet
   ACM International Conference on Functional Programming, 2006
   http://dx.doi.org/10.1145/1160074.1159821

5. A Type and Effect System for Atomicity
   Cormac Flanagan, Shaz Qadeer
   ACM Conference on Programming Language Design and Implementation, 2003
   http://dx.doi.org/10.1145/780822.781169

6. Phantom Types and Subtyping
   Matthew Fluet, Riccardo Pucella
   Journal of Functional Programming, 2006
   http://dx.doi.org/10.1017/S0956796806006046

7. The Essence of JavaScript
   Arjun Guha, Claudiu Saftoiu, Shriram Krishnamurthi
   European Conference on Object-Oriented Programming, 2010
   http://www.cs.brown.edu/~sk/Publications/Papers/Published/gsk-essence-javascript/

8. Automatically Restructuring Programs for the Web
   Jacob Matthews, Robert Bruce Findler, Paul T. Graunke, Shriram Krishnamurthi, Matthias Felleisen
   Automated Software Engineering Journal, 2004
   http://www.cs.brown.edu/~sk/Publications/Papers/Published/mfgkf-web-restructuring-cps-journal/

9. A compiler and run-time system for network programming languages
   Christopher Monsanto, Nate Foster, Rob Harrison, David Walker
   ACM Symposium on the Principles of Programming Languages, 2012
   http://dx.doi.org/10.1145/2103656.2103685

10. High-Level Small-Step Operational Semantics for Transactions
    Katherine F. Moore, Dan Grossman
    ACM Symposium on the Principles of Programming Languages, 2008
    http://dx.doi.org/10.1145/1328438.1328448

11. Fault-Tolerant Typed Assembly Language
Frances Perry, Lester Mackey, George A. Reis, Jay Ligatti, David I. August, David Walker
ACM Conference on Programming Language Design and Implementation, 2007
http://dx.doi.org/10.1145/1250734.1250741

12. The F# Asynchronous Programming Model
Tomas Petricek, Dmitry Lomov, Don Syme
International Symposium on Practical Aspects of Declarative Languages, 2011
http://blogs.msdn.com/cfs-file.ashx/__key/CommunityServer-Components-PostAttachments/00-10-07-89-59/async_2D00_padl.pdf

13. Formal Verification of Object Layout for C++ Multiple Inheritance
Tahina Ramananandro, Gabriel Dos Reis, Xavier Leroy
ACM Symposium on the Principles of Programming Languages, 2011
http://dx.doi.org/10.1145/1925844.1926395

14. Addressing covert termination and timing channels in concurrent information flow systems
Deian Stefan, Alejandro Russo, Pablo Buiras, Amit Levy, John C. Mitchell, David Mazires
ACM International Conference on Functional Programming, 2012
http://dx.doi.org/10.1145/2364527.2364557

15. Extensible Pattern Matching via a Lightweight Language Extension
Don Syme, Gregory Neverov, James Margetson
ACM International Conference on Functional Programming, 2007
http://dx.doi.org/10.1145/1291151.1291159

16. The Design and Implementation of Typed Scheme
Sam Tobin-Hochstadt, Matthias Felleisen
ACM Symposium on the Principles of Programming Languages, 2008
http://dx.doi.org/10.1145/1328438.1328486

17. Practical Affine Types
Jesse A. Tov, Riccardo Pucella
ACM Symposium on the Principles of Programming Languages, 2011
http://dx.doi.org/10.1145/1926385.1926436

18. Integrating Typed And Untyped Code in a Scripting Language
Tobias Wrigstad, Francesco Zappa Nardelli, Sylvain Lebresne, Johan Östlund, Jan Vitek
ACM Symposium on the Principles of Programming Languages, 2010
http://dx.doi.org/10.1145/1706299.1706343

19. A language for automatically enforcing privacy policies
Jean Yang, Kuat Yessenov, Armando Solar-Lezama
ACM Symposium on the Principles of Programming Languages, 2012
http://dx.doi.org/10.1145/2103656.2103669