DECLARATIVE PROGRAMMING: STATUS AND CHALLENGES

JOE HELLERSTEIN, BERKELEY

INSPIRATION

- GRAY TURING
  - “This talk defines a set of fundamental research problems that broaden the Babbage, Bush, and Turing visions.”
- LAMPSON JACM 50’th
  - “I have chosen two problems whose solution will make a big difference both to computer science and to the larger world of which computing is now such an important part.”

Automatic Programming

Do What I Mean (not 100$ Line of code!, no programming bugs)

The holy grail of programming languages & systems

12. Devise a specification language or UI
    1. That is easy for people to express designs (1,000x easier)
    2. That computers can compile.
    3. That can describe all applications (is complete).

- System should "reason" about application
  - Ask about exception cases.
  - Ask about incomplete specification.
  - But not be onerous.
- This already exists in domain-specific areas. (i.e. 2 out of 3 already exists)
- An imitation game for a programming staff.

LAMPSON: “GETTING COMPUTERS TO UNDERSTAND”

1. no more traffic deaths
2. automatic programming
   - “As stated, the problem is too hard …”

- existing results:
  - declarative (SQL, spreadsheets, HTML layout)
  - transactions
  - big components

PERSPIRATION

- FOUR CHALLENGES
  - SEMANTICS
  - PROGRAMMABILITY
  - METRICS
  - THE CRITICS AND THE STREET

SEMANTICS

- two anchors:
  - partitioned data
  - local logic
- extend formally:
  - modeling local state
  - modeling events
  - modeling asynchrony distribution, and partial failure

We can devise a fully declarative (model-theoretic) logic language that captures the familiar systems notions of manipulating events and state across space and time, and in doing so provide a foundation for correctly and compactly specifying, checking and building *distributed systems.*

PROGRAMMABILITY

- Syntax
  - abandon datalog
  - reify the subtler semantics
- Tools
  - debuggers: static/dynamic
  - code structure: encapsulation, aspects
  - alpo tools: declarative configuration/provisioning, test harnesses, SCM.
  - IDE integration
  - interoperability

Overlog is hard to read for reasons both shallow and deep. We must overcome the shallow problems immediately, and start addressing state, time, and failure in a way that makes the hard programming decisions self-evident.

Language design is both more useful and more interesting in the context of a full development lifecycle. Making declarativity real is good research in every sense.

METRICS

- the age of LOCs is past
- it is time to perform
- and how shall we measure:
  - malleability
  - correctness and debuggability

Some researchers are driven by elegance, beauty, cleverness. But the research market thrives on metrics, and we should foster that market.

There is no inherent reason today for dataflow execution of logic to lag thread-programming in performance. It is time to step up to this challenge.

Meanwhile we need to define metrics that capture programmer productivity.

THE CRITICS AND THE STREET

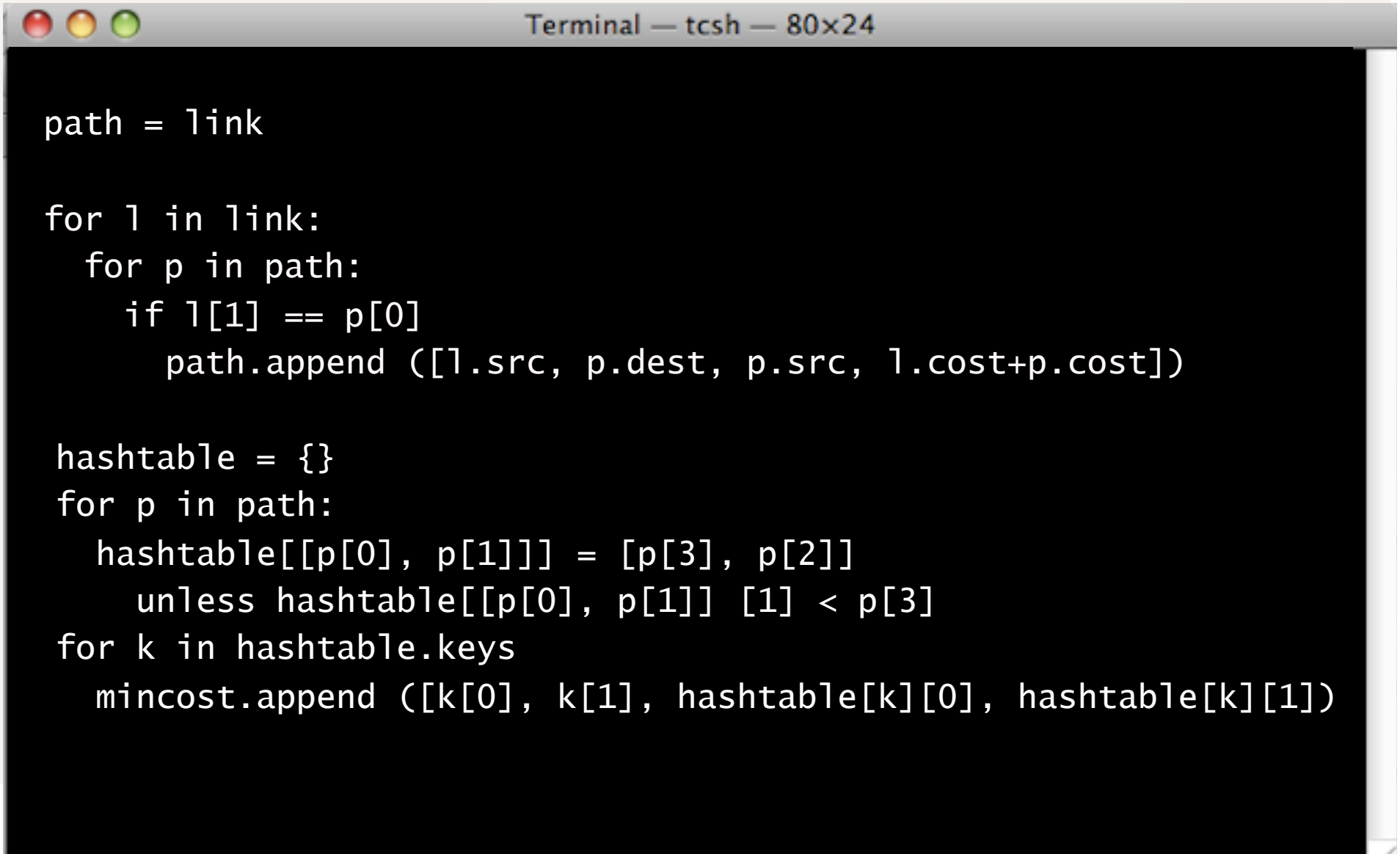
- there will be naysayers
- but there will be many more who ignore the work
  - this is the audience
- take on a pressing agenda
  - network protocols are a playpen, but let's expand
  - cloud programming, client/server for handhelds, system management…

Howard Hathaway Aiken: "If your ideas are any good, you'll have to ram them down people's throats."

Don't ignore critics, they keep you honest. But don't let them dissuade you.

The real challenge for this work is relevance. Not because it's not useful, but because people may choose to suffer for another decade. It would not be the first time in the PL wars.

IS THIS DECLARATIVE?



IS THIS DECLARATIVE?