The Eduction Algorithm is a well understood computational model for evaluating Lucid programs [FaWa86]. Eduction has several key characteristics that distinguishes it from other computational models:

- explicit operators manipulate the environment context,
- function parameters are lazily evaluated, and
- several educers can concurrently evaluate the same program.

The Eduction Algorithm shows promise at achieving parallelism by concurrent program evaluation on multi-processor computer systems.

The major disadvantage of Eduction is that the original Lucid program must be compiled into Yaghi code. This requirement conflicts with an interactive, incremental software programming environment. The programmer assembles and test equations in isolation. By combining equations with well-known and validated properties, the programmer can construct more complex equations. Whenever an unexpected result occurs, the programmer can watch expression evaluation to determine where the error lies.
Our motivation is to enlarge the eduction algorithm to allow these activities to occur dynamically.

The software programming environment for Lucid, called lsh (lush), is under development. A subset of Lucid has been identified and designated as Lurid, the lsh instruction set. Lurid is similar to Lucid except

- no nested scopes are permitted, the where clause is entirely eliminated,
- a program is a sequence of expressions and definitions,
- expressions and definitions are represented in prefix notation,
- a definition associates an expression with an word
- an word may have several definitions,
- definitions have guards,
- guards are expressions.

Lurid itself is implemented in LAM (Lurid Abstract Machine) instructions. LAM closely resembles conventional assembly instructions that perform single instructions on a variety of computer systems.

The Eduction Algorithm will either operate over Lurid programs in interpretation mode or be directly represented by a sequence of LAM instructions in compiled mode. The Education Algorithm must efficiently and effectively evaluate Lurid programs, i.e. perform pattern matching, lazy evaluation, and handle alternative actions. These requirements require the addition of unification and continuations (failure and success continuations) to the Eduction Algorithm. These requirements are scrutinized in more detail in the remaining sections.

References


---

Eductive Pattern Matching

105