Experiments in Alchemy
The Language

• Variation of Pure Lisp
• Expression
  – Atom
    • Operator: +, -, ’, <, >, *(,)
    • Variable: a, b, c, …
  – List: (Expr1, Expr2, Expr3…..)
Semantics
(The “Meaning” of Expressions)

• The meaning of an expression is also an expression
  – Thus expressions have dual-identities as functions and objects

• The meaning of an expression is the result of “evaluating” the expression
  – Thus “meaning” is synonymous with expression evaluation
Molecule = Expression

- A Molecule in the system is an expression
  - Thus a molecule has a syntactic sense and a semantic sense
Interactions Between Molecules

Exp1 + Exp2 =

The meaning of ( (' Exp1) (' Exp2) )
Expression Evaluation

\[((+a)(*(>a)(>(+a)))) + ((*(+a)(*(-a)(<a))))a\]

= meaning of [ \(((+(a)(*(>a)(>(+a))))((*(+a)(*(-a)(<a))))a\) ]

= \((a(*(\neg a)(<a)))\)
See Trace of Expression Evaluation

- Evaluating Defined function (must first evaluate all expressions in the list):('(((+a)(*a)(*a)))('((*(+a)({-a}(<a)))a))
  - Evaluating ('((+a)(*a)(*a))) with args:a=a
    - Evaluating Primitive function (must first evaluate all argument Expressions) :('((+a)(*a)(*a)))
      - Value returned = ((+a)(*a)(*a)))
  - Evaluating ('((*(+a)({-a}(<a)))a)) with args:a=a
    - Evaluating Primitive function (must first evaluate all argument Expressions) :'((*(+a)({-a}(<a)))a))
      - Value returned = ((*(+a)({-a}(<a)))a)

---Expression Evaluation Completed. We now Sequentially apply the resultant values to each other
- Evaluating ('((+a)(*a)(*a))) with args:a=(((+a)(*a)({-a}(<a)))a)
  - Evaluating Defined function (must first evaluate all expressions in the list):((+a)(*a)(*a)))
    - Evaluating (+a) with args:a=(((+a)(*a)({-a}(<a)))a)
      - Evaluating Primitive function (must first evaluate all argument Expressions) :(+a)
      - Evaluating a with args:a=(((+a)(*a)({-a}(<a)))a)
        - ----Argument evaluation Completed. We now apply the primitive to the resultant values
    - Evaluating + with args:a=(((+a)(*a)({-a}(<a)))a)
      - Value returned = (+(+a)(*a)({-a}(<a))))
  - Evaluating ('((*(+a)({-a}(<a)))a)) with args:a=a
    - Evaluating Primitive function (must first evaluate all argument Expressions) :'((*(+a)({-a}(<a)))a))
      - Evaluating a with args:a=(((+a)(*a)({-a}(<a)))a)
        - ----Argument evaluation Completed. We now apply the primitive to the resultant values
    - Evaluating > with args:a=(((+a)(*a)({-a}(<a)))a)
      - Value returned = a
  - Evaluating (>a) with args:a=(((+a)(*a)({-a}(<a)))a)
    - Evaluating Primitive function (must first evaluate all argument Expressions) :>(a)
      - Evaluating a with args:a=(((+a)(*a)({-a}(<a)))a)
        - ----Argument evaluation Completed. We now apply the primitive to the resultant values
    - Evaluating > with args:a=(((+a)(*a)({-a}(<a)))a)
      - Value returned = >a
  - Evaluating (>a) with args:a=(((+a)(*a)({-a}(<a)))a)
    - Evaluating Primitive function (must first evaluate all argument Expressions) :>(a)
      - Evaluating a with args:a=(((+a)(*a)({-a}(<a)))a)
        - ----Argument evaluation Completed. We now apply the primitive to the resultant values
    - Evaluating > with args:a=(((+a)(*a)({-a}(<a)))a)
      - Value returned = >a
  - Evaluating > with args:a=(((+a)(*a)({-a}(<a)))a)
    - Evaluating Primitive function (must first evaluate all argument Expressions) :>(a)
      - Evaluating a with args:a=(((+a)(*a)({-a}(<a)))a)
        - ----Argument evaluation Completed. We now apply the primitive to the resultant values
    - Evaluating > with args:a=(((+a)(*a)({-a}(<a)))a)
      - Value returned = >a
  - Evaluating > with args:a=a
    - Evaluating Primitive function (must first evaluate all argument Expressions) :>(a)
      - Evaluating a with args:a=a
        - ----Argument evaluation Completed. We now apply the primitive to the resultant values
    - Evaluating > with args:a=a
      - Value returned = >a
Interaction Graph

- What do the lines mean?
- Notion of Elastic collisions
Iterated Interaction Graph

- See handout
- Autocatalytic Subgraphs
  - All nodes in the subgraph must have two incoming edges from other nodes in the subgraph
- Seeding Sets of an Autocatalytic subgraph
  - What happens if you remove a subset of the nodes in a subgraph
- Minimal Seeding Set of an Autocatalytic subgraph
Iterated Interaction Graph (cntd...)

- Closure
Turing Gas

- Dynamics
  - Fixed size reactor
  - Reactions
    - Random collisions between molecules
  - Dilution
    - Random removal of a molecule in the reactor
Innovation and Absolute Innovation

- Innovative reaction: the species of the product molecule is not currently in the reactor
- Absolutely innovative reaction: the species of the product molecule has never been seen by the reactor during the run
One Sample Run

Number of Different Molecular Species (i) Ever Seen [Red] & (ii) Currently in the Reactor [Black]

(i) Innovation Rate [blue] & (ii) Absolute Innovation Rate [magenta]