A classifier system:

A ML system that learns syntactically simple rules (called classifiers) to guide performance in arbitrary environments.

1. **Rule and Message System**
2. **Credit Allocation System**
3. **Genetic Algorithm**

Rule and Message System

IF (cond) THEN action

Classifier systems restrict syntax but not semantics of rules

Fixed length strings (ternary)
Rules/Classifiers have a strength/fitness
Rules/Classifiers compete to fire
Competition based on relative strengths(more)
Competition in an information based economy
Succesful rules gain strength
Unsuccesfull rules lose strength

Learning:

- By changing rule strengths
- By using GA to generate new rules

```
cond  :Message(action)
010001:0
11#001:1
10#00#:0
10##1#:1
######:0
```

Simple TYPE I CS

```
        Get environmental message
        Make list of matching classifiers
        Matching Classifiers BID to fire
        Auction determines winner
        Winning Classifier fires
                 -> message to env.
        Repeat
```

TYPE II

```
        Message board to store internal state

        Get environmental message
        Look at message board
        Make list of matching classifiers
                (env, board)
        Matching Classifiers BID to fire
        Auction determines winner
        Winning Classifier fires
                -> message to env/board
        Repeat
```

TYPE III, TYPE IV (more complex and I don't know of implementations)

We only consider TYPE I

Strengths:

```
 S(t+1) = S(t) - P(t) - T(t) + R(t);
```

Bid:

```
 B = Cb * S
```

Effective bid (to reduce stagnation)

```
 Eb = B * N(sigma)
```

Winners pay B not EB to activators thru a clearinghouse

Taxes: (no freeloading)

```
 T = Ct * S
```

```
 S(t+1) = S(t) - Cb*S(t) - Ct*S(t) + R(t)

 S(t+1) = (1 - K) S(t) + R(t)

        K = Cb + Ct

Forget R(t)

 S(t+1) = (1 - k) S(t)

at t = n

 S(n) = (1 - K)^n S(0)

when 0 <= K <= 2, S(n) is not infinite
We in practice restrict 0 <= K <= 1 to have
positive strengths only.
```

```
 Steady state value:

   Sss = Rss/K  for steady Reward R

 Steady state Bid:

   Bss = Cb * Rss/K
     = Rss * Cb/(Cb + Ct)

If we make Ct << Cb

   Bss =~ Rss
```

Multiplexor:

2 address 6 data lines, one out bit

```
###000:0
###100:1
```

Here's a rule set that should work

```
###000:0
##0#01:0
#0##10:0
0###11:0
###100:1
##1#01:1
#1##10:1
1###11:1
```

What os steady state strenth?

Sss = Rss/(Ct + Cb)

Rss = 1, Ct = 0.01, Cb = 0.1

Sss = 9.09

Bss = Cb * 9.09 = 0.909

Monkey wrenches:

######:0
######:1

Default hierarchy:

```
###000:0
##0#01:0
#0##10:0
0###11:0
######:1
```

Only five rules!

Default Hierarchies:

Parsimony
Enlargement of Solution Set

How?

```
 B = Cb * S(t) * F(Specificity)

 F(Specificity) = b1 + b2 * Specificity
```

Leads to:

```
 Sss = Rss/ (Cb * F(Spec)  + Ct)

Bss = (Cb * F(Spec) * Rss)/(Cb*F(Spec) + Ct)
```

Without specificity dependant bidding
SCS performs badly

With specificity dependant bidding
SCS learns default hierarchy and does better.