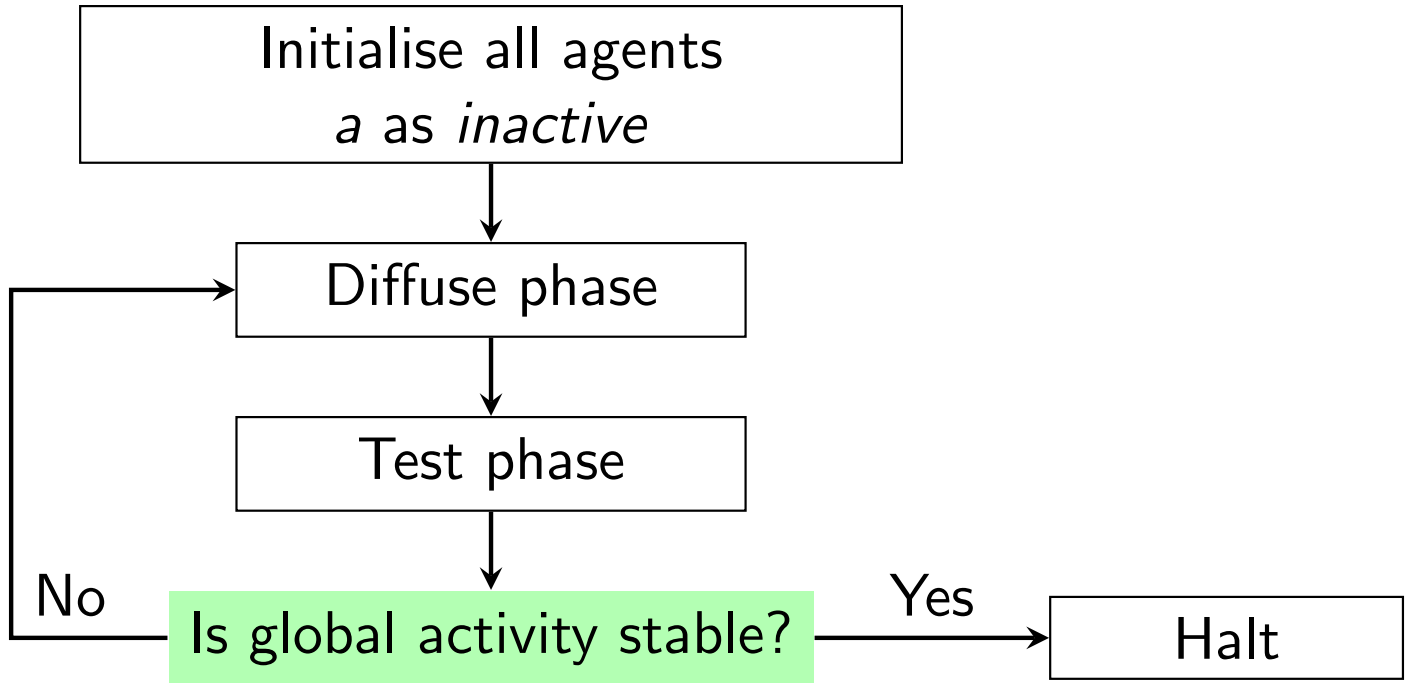
# Natural Computing — Stochastic Diffusion Search

Andrew O. Martin
The Centre for Intelligent Data Analytics
http://www.aomartin.co.uk/uploads/natural-computing.pdf

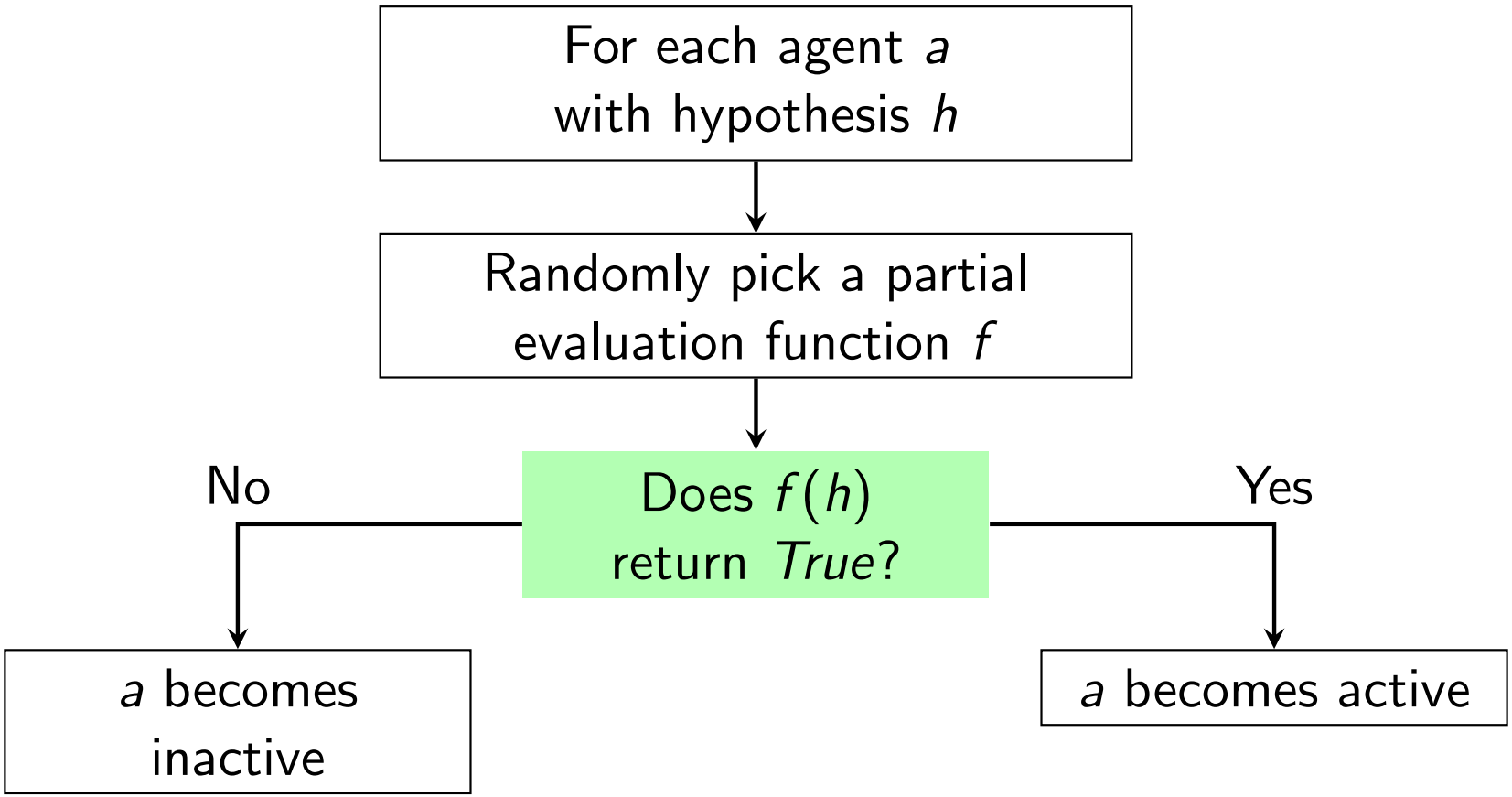
9:05–10:55 Wednesday 17 October 2018
PSH 314

# Stochastic Diffusion Search (SDS)



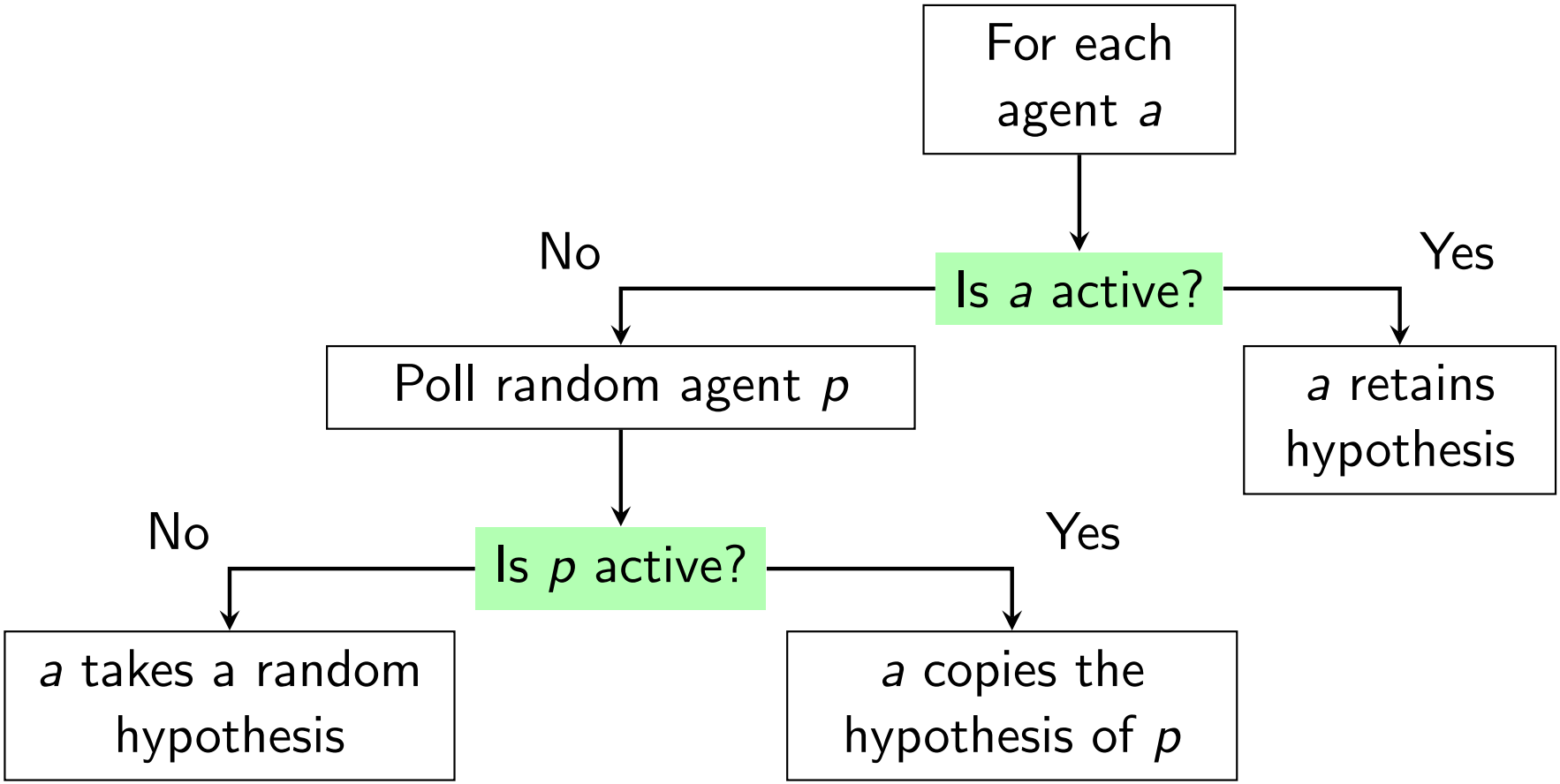

*Global activity* is the proportion of *active* agents.

# Test phase




$f$ can be any boolean function of $h$.

# Diffuse phase




Inspired by the ants, we added one condition to both sides.

# An instance of SDS

1. Initialise a list of inactive agents
2. While the halting condition is not met
   - 2.1 Perform a diffusion phase; where all agents update their hypothesis
   - 2.2 Perform a test phase; where all agents update their activity
3. Return the hypothesis with the most active agents

# Example SDS

**Algorithm 1** Standard SDS

```
1:  while halting condition is not reached do
2:      for each agent in swarm do                    ▷ Diffusion phase
3:          if agent is inactive then
4:              poll a random agent
5:              if polled agent is active then
6:                  agent copies polled agent's hypothesis
7:              else
8:                  agent adopts a random hypothesis
9:      for each agent in swarm do                    ▷ Test phase
10:         agent selects a random microtest (f)
11:         agents set activity to result of f(agent's hypothesis)
12: return largest cluster
```

# String Search Example

**Algorithm 2** String Search SDS

```
1:  while active_agents() < 90 do
2:      for each agent in swarm do                    ▷ Diffusion phase
3:          if agent is inactive then
4:              poll a random agent
5:              if polled agent is active then
6:                  agent copies polled agent's hypothesis
7:              else
8:                  agent.hyp = random.int(text.length)
9:      for each agent in swarm do                    ▷ Test phase
10:         f = random.int(model.length)
11:         agent.active = model[f] == text[agent.hyp+f]
12: return largest cluster
```

# The Diffusion Phase

Each inactive agent randomly polls another agent and acts depending on the polled agent's activity value.

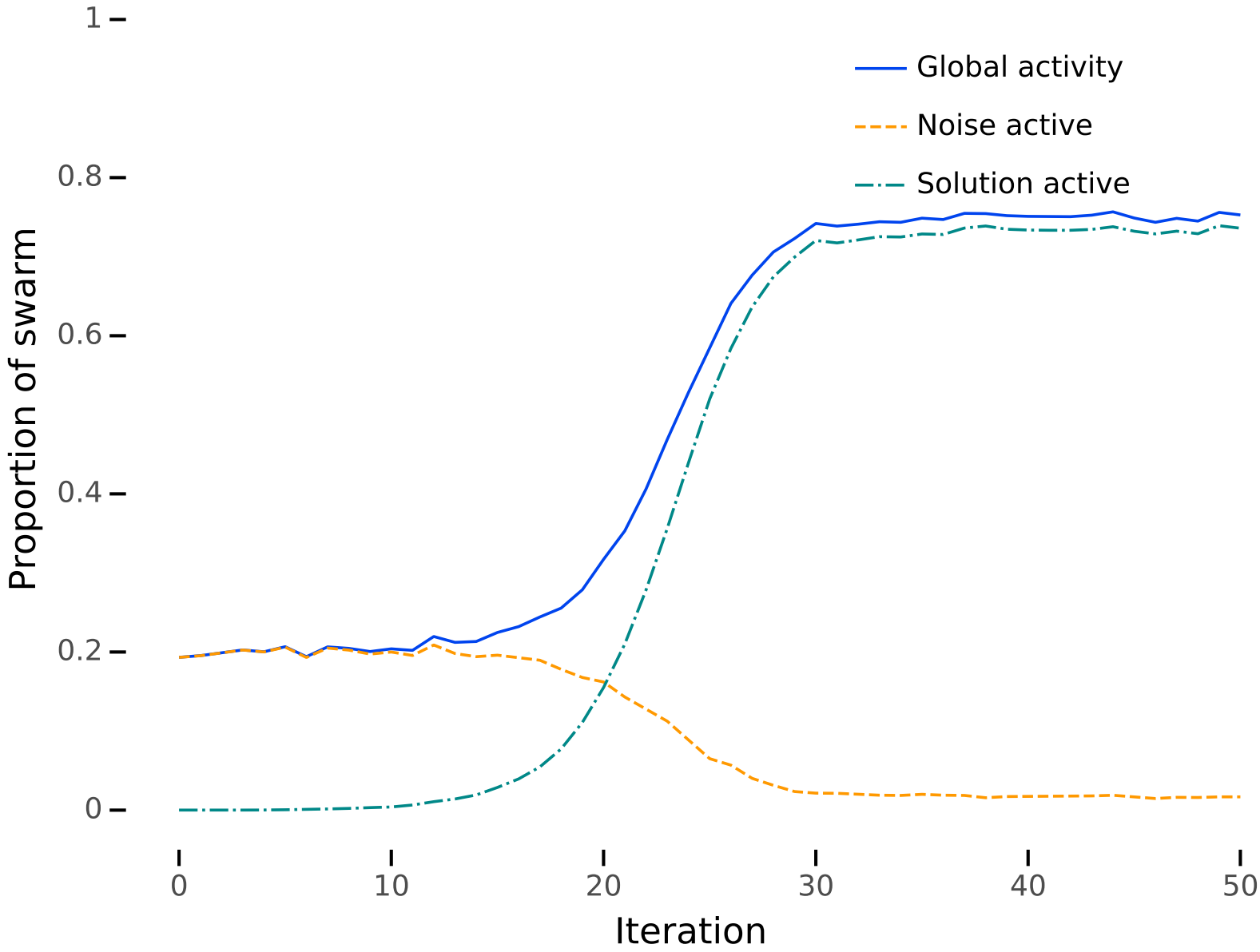
**active** The inactive agent copies the hypothesis of polled agent.

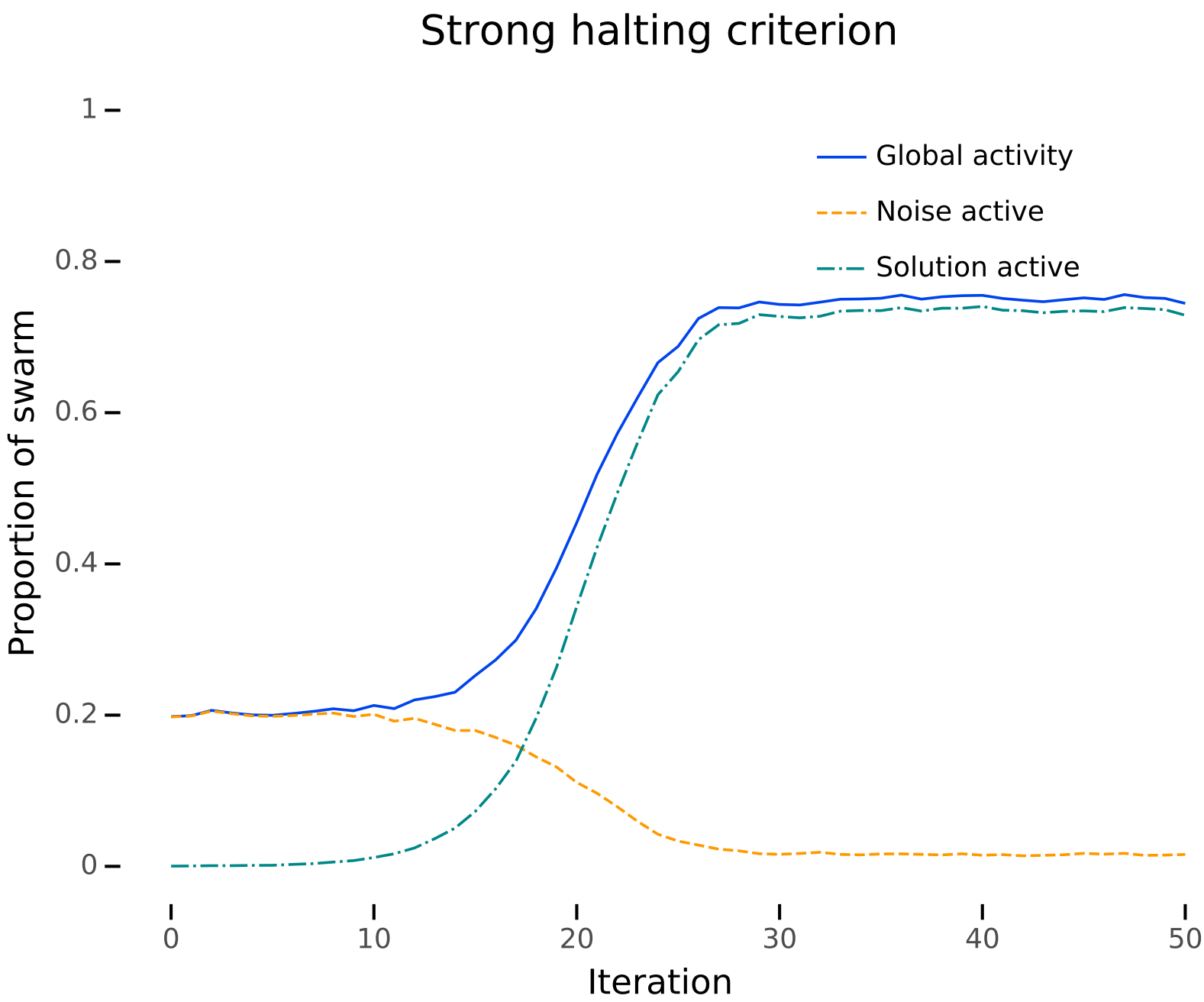
**inactive** The inactive agent selects a hypothesis at random.

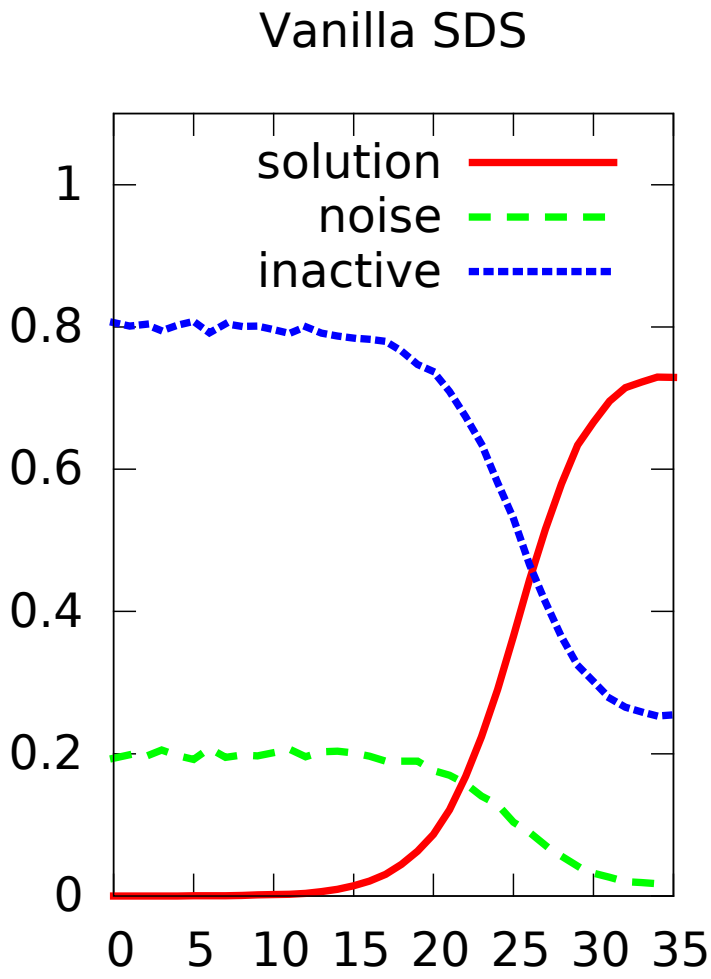
# The Test Phase

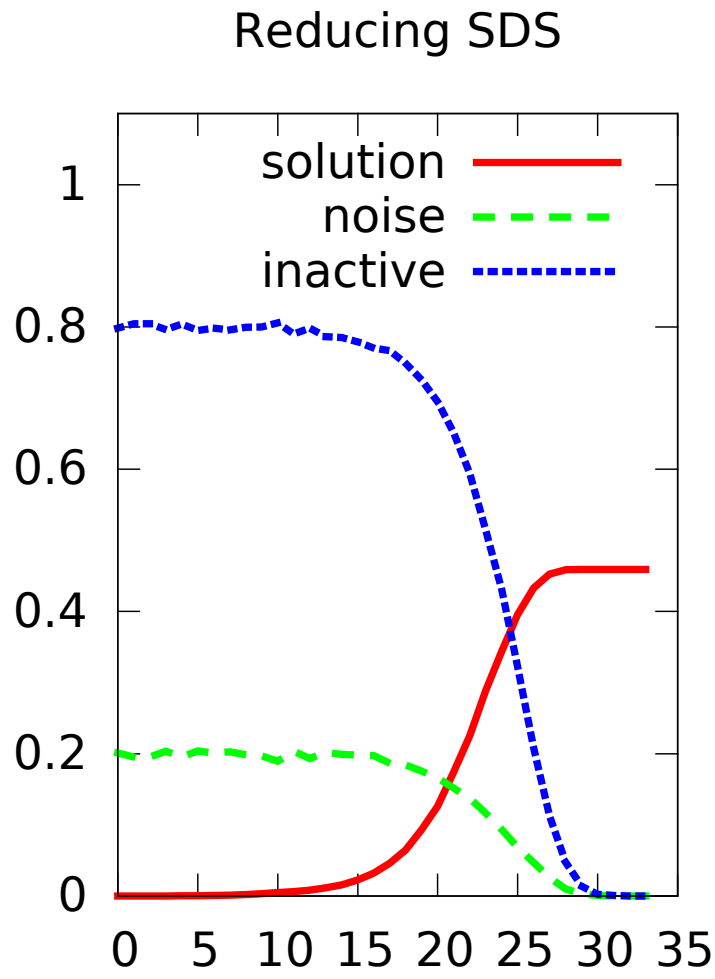
Each agent partially evaluates their hypothesis by passing it as input to a randomly selected microtest function.

The agents update their activity value depending on the result.


Weak halting criterion
Global activity
Noise active
Solution active
Proportion of swarm
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
Iteration


Strong halting criterion
Global activity
Noise active
Solution active
Proportion of swarm
Iteration
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50


Vanilla SDS
solution
noise
inactive
1
0.8
0.6
0.4
0.2
0
0 5 10 15 20 25 30 35


Reducing SDS
solution
noise
inactive
1
0.8
0.6
0.4
0.2
0
0 5 10 15 20 25 30 35

# Why swarm intelligence?

No executive control.

Can avoid symbolic computation.

Nature inspired, biologically plausible.

Tolerant to noise, including a dynamic search space.

Real world problems often require good solutions quickly.

# Why SDS in particular?

Invented by Bishop in 1989. First Swarm Intelligence algorithm.

Easy to model mathematically.

Easy to program.

Many interesting major variants.

# Mathematical results

SDS clusters are stable, lasting for $10^{602}$ iterations.

SDS clusters are sensitive, to within $10^{-6}$.

SDS will converge to the optimal with probability tending to 1.

SDS outperforms other algorithms when the evaluation function is highly decomposable.

SDS time complexity increases sub-linearly with the search space.

# Mathematical results

Given;
$\alpha$, the probability of an agent becoming active when at the optimal hypothesis; and
$\beta$, the probability of becoming active when not at the solution.

SDS will not converge if

$$\alpha < \frac{1}{2-\beta} \tag{1}$$

Otherwise SDS converges to $\gamma$, the final number of active agents.

$$\gamma = \frac{\alpha(2-\beta)-1}{\alpha-\beta} \tag{2}$$

# Major SDS variants

## Context Free SDS

In the diffusion phase active agents also poll random agents.

If the polled agent is active, the polling agent becomes inactive.

+ More inactive agents means more exploration of the search space
− More inactive agents means less cluster stability

Good for cases with large search spaces and the requirement to find the single stand-out optimal solution.

# Major SDS variants

## Context Sensitive SDS

In the diffusion phase active agents also poll random agents.

If the polled agent is active, *and they share a hypothesis* the polling agent becomes inactive.

+ Multiple stable and proportionate clusters may form
− Multiple clusters leaves very few agents exploring the search space

Good for cases where a few good solutions are needed quickly.

# Major SDS variants

## Multitesting

An agent performs multiple tests and sets their activity to a function of the results. E.g. ANDing all the results together to reduce activity, or ORing all the results together to increase activity.

+ Allows SDS to converge to optimal solutions in cases which would otherwise reach local minima, or not converge at all
− Extra tests increase computational complexity.

Good for search spaces where there are strong suboptimal solutions, or weak optimal solutions.

# Major SDS variants

## Multidiffusion

An agent polls multiple agents in the diffusion phase, and responds if any/all of them are active.

+ Can be used to increase cluster stability
− Increased cluster stability leads to more local minima

Good for cases where hypothesis generation is computationally expensive.

# Major SDS variants

## Comparative Test Phase

Microtest functions return real numbers, not boolean values. Agents then determine their activity by comparing their result with that of a random agent.

+ No need to define a manual threshold for going active
+ Half the swarm will be active in proportionately sized clusters
+ Can be used to sort a list, even a changing one
− "Nontransitive dice"[1] situations where $A > B > C > A$

Good for continuous global optimisation tasks.

[1] A:249, B:168, C:357

# Major SDS variants

## Transmission Error

When an agent copies the hypothesis of another agent, it is perturbed somewhat.

+ A form of local hill-climbing optimisation emerges
− The stability of a cluster is reduced by an amount proportional to the perturbation.

Good for continuous search spaces, where randomly generating the globally optimal solution is very unlikely.

# Major SDS variants

## Asynchronous/Parallel SDS

Individual agents perform test phases and diffusion phases in real time.

+ Enables parallel/grid/cluster/cloud SDS implementations
− Added complexity, can't be optimised by the compiler.

Good for truly large, or continuous online problems.

# Major SDS variants

## Lattice SDS

Agents may only communicate with certain agents.

+ Reduces the cost of hardware implementations
− Slightly reduced performance, increased code complexity

Good for on-circuit implementations of SDS, with small-worlds connectivity.

# End of Introduction to SDS

Online Demo animation:
http://www.aomartin.co.uk/sds-animation/

Local Demo animation:
file:///home/amartin/sds-animation/index.html

SDS Library Documentation:
http://www.aomartin.co.uk/sds-library/

SDS Library on PyPi:
https://pypi.python.org/pypi/sds/

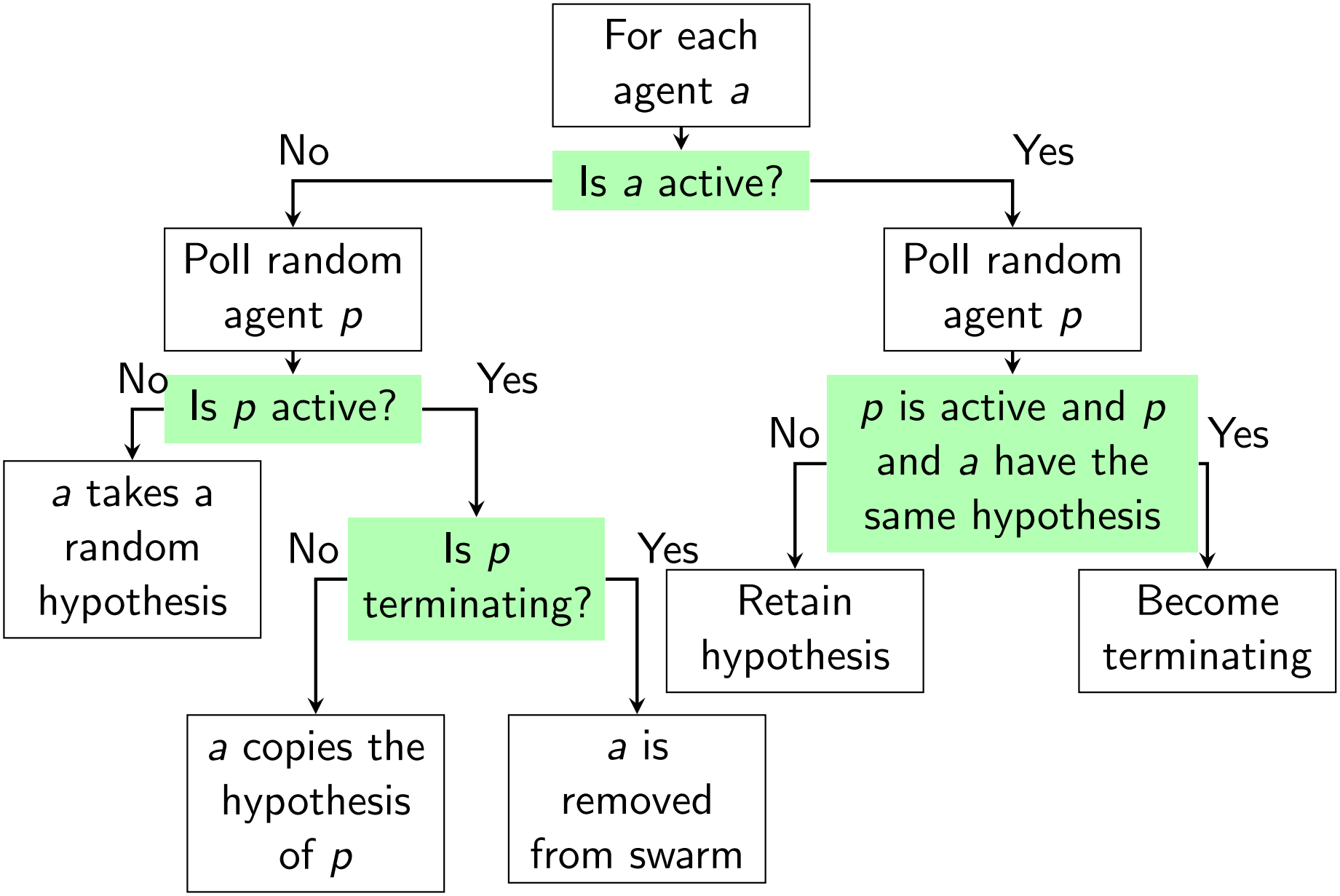
# Ant nest-site selection

Bishop and I attended a lecture on Robinson's studies of ant nest-site selection.

1. An ant doing its duties, stumbles upon a candidate nest.
2. The ant runs around the candidate nest, evaluating the nest *partially* and *randomly*.
3. If the ant is satisfied it returns to the nest and the first ant it encounters is recruited into the process, and is lead back to the candidate nest through tandem-running.
4. When the ants detect quorum, they shift behaviour into lifting and dropping.

That sounds like SDS!

# *Reducing* Diffuse phase



# Extra Slides

# Stochastic Diffusion Search

**Random hypothesis selection** All agents randomly assume a location in the search space as their *hypothesis*.

**Partial evaluation** All agents randomly perform a partial test against their hypothesis. Becoming *active* or *inactive*

**Recruitment positive feedback loop**

- **Active agents** Retain their hypothesis to continue to partially evaluate it.
- **Inactive agents** Randomly poll an agent, if they poll an inactive agent then the polling agent assumes a random hypothesis; if they poll an active agent the polling agent assumes the hypothesis of the polled agent.

**Terminate on quorum detection** Once the global number of active agents in the swarm has stabilised, halt the process. The answer is the hypothesis held by the greatest number of active agents.

# Example hypothesis generating functions

Each SDS needs *one* function which produces a random hypothesis.

**TSP** Randomly generated path

```
def random_hypothesis():
    return random_path(cities)
```

**Go** Randomly selected next move

```
def random_hypothesis():
    return random_valid_move(state)
```

**Analogies** Randomly selected word from the corpus

```
def random_hypothesis():
    return random.choice(corpus)
```

**Search** Randomly selected location in a text

```
def random_hypothesis():
    return random.randint(0,len(text))
```

# Microtests

A set of functions which each take a hypothesis and return a value.

| | |
|---|---|
| TSP | The distance between two random cities |
| Go | Who wins if the game is completed randomly |
| Analogies | If a random feature between the hypothesis and the third element is the same as between the first two elements |
| Search | If a random letter lines up with the search space |

```
def t0(hyp):  return text[hyp+0] == 'h'
def t1(hyp):  return text[hyp+1] == 'e'
def t2(hyp):  return text[hyp+2] == 'l'
def t3(hyp):  return text[hyp+3] == 'l'
def t4(hyp):  return text[hyp+4] == 'o'
def is_upper(hyp):  return text[hyp+0].is_uppercase
microtests = [t0, t1, t2, t3, t4, is_upper]
```