SDS Library

Andrew Owen Martin

3 May 2020

Contents

Chapter 1

Standard SDS

1.1 SDS

4 ⟨*sds* 4⟩≡

```
def SDS4(I, H):

    while not H():

        I()
```

This code is used in chunk 40.
Defines:
SDS, used in chunk 42a.

1.2 Agent

5 ⟨*standard agent* 5⟩≡

```
class Agent5:
    def __init__(self, active=False, hyp=None):
        self.active = active
        self.hyp = hyp

    @property
    def inactive(self):

        return not self.active

    @property
    def clone(self):

        return ReadOnlyAgent(active=self.active, hyp=self.hyp)

    def __iter__(self):

        yield ("active", self.active)
        yield ("hyp", self.hyp)

    def __str__(self):

        if self.active:

            return str(self.hyp)

        else:

            return "Inactive"
```

This code is used in chunk 40.
Defines:
Agent, used in chunks 6, 7, 35a, and 42a.

1.2.1 Unit test

6 ⟨*unit tests* 6⟩≡

```
def test_agent(self):
    agent = sds.Agent_5()
    self.assertIsNone(agent.hyp)
    self.assertFalse(agent.active)
    agent = sds.Agent_5(hyp="hello", active=True)
    self.assertEqual(agent.hyp, "hello")
    self.assertTrue(agent.active)
```

This code is used in chunk 42b.
Uses Agent 5.

1.3 Swarm

7 ⟨*standard swarm 7*⟩≡

```
class Swarm7(collections.UserList):
    def __init__(self, agent_count=None, swarm=None, AgentClass=Agent5):

        if swarm is None:

            if agent_count is None:

                raise ValueError("One of agent_count or swarm must be passed")

            else:

                self.data = [AgentClass() for _ in range(agent_count)]

        else:

            self.data = swarm

    def __str__(self):

        return ", ".join(
            f"(Hyp:{hyp}, Agents:{cluster_size})"
            for hyp, cluster_size in self.clusters.most_common()
        )

    @property
    def activity(self):

        if not self:

            return 0

        return sum(1 for agent in self if agent.active) / len(self)

    @property
    def clusters(self):

        return collections.Counter(agent.hyp for agent in self if agent.active)

    @property
    def largest_cluster(self):

        try:

            hyp, agents = self.clusters.most_common(1)[0]

        except IndexError:
```

```
            hyp, agents = None, 0

        return Cluster(hyp=hyp, agents=agents, size=agents / len(self))

    def report_clusters(self, significant_hypotheses):

        clusters = self.clusters

        opt = tuple((hyp, clusters[hyp]) for hyp in significant_hypotheses)

        active = sum(clusters.values())

        inactive = len(self) - active

        noise_active = active - sum(size for hyp, size in opt)

        log.debug(
            "Opt hyp: %s, active: %s, inactive: %s, noise active: %s",
            opt,
            active,
            inactive,
            noise_active,
        )

        return {
            "opt-hyp": opt,
            "active": active,
            "inactive": inactive,
            "noise active": noise_active,
        }
```

This code is used in chunk 40.
Defines:
`Swarm`, used in chunks 38a and 42a.
Uses `Agent` 5.

8a ⟨*standard imports* 8a⟩≡

```
import collections
```

This code is used in chunk 40.

8b ⟨*cluster* 8b⟩≡

```
Cluster = collections.namedtuple("Cluster", ("hyp", "agents", "size"))
```

This code is used in chunk 40.

1.4 Modes of iteration

Synchronous iteration

9a ⟨*synchronous iteration* 9a⟩≡

```
def I_sync9a(D, T, swarm):
    def I():

        for agent in swarm:

            D(agent)

        for agent in swarm:

            T(agent)

    return I
```

This code is used in chunk 40.
Defines:
I_sync, used in chunk 42a.

Asynchronous iteration Each agent in the swarm performs diffusion then testing.

9b ⟨*iteration variants* 9b⟩≡

```
def I_async9b(D, T, swarm):

    def I_prime():

        for agent in swarm:

            D(agent)

            T(agent)

    return I_prime
```

This definition is continued in chunk 19.
This code is used in chunk 41a.
Defines:
I_async, never used.

1.5 Modes of diffusion

Passive diffusion

10 ⟨*passive diffusion* 10⟩≡

```
def D_passive10(DH, swarm, rng):
    def D(agent):

        if agent.inactive:

            polled = rng.choice(swarm)

            if polled.active:

                agent.hyp = polled.hyp

            else:

                agent.hyp = DH()

    return D
```

This code is used in chunk 40.
Defines:
D_passive, used in chunk 42a.

Context-free diffusion

11 ⟨*diffusion variants* 11⟩≡

```
def D_context_free11(DH, swarm, rng):

    def D(agent):

        polled = rng.choice(swarm)

        if agent.inactive or polled.active:

            if agent.inactive and polled.active:

                agent.hyp = polled.hyp

            else:

                agent.active = False

                agent.hyp = DH()

    return D
```

This definition is continued in chunks 12–14.
This code is used in chunk 41a.
Defines:
D_context_free, never used.

Context-sensitive diffusion

12 ⟨*diffusion variants* 11⟩+≡

```
def D_context_sensitive12(DH, swarm, rng):

    def D(agent):

        polled = rng.choice(swarm)

        if polled.active and agent.inactive:

            agent.hyp = polled.hyp

        else:

            if agent.inactive or polled.active and agent.hyp == polled.hyp:

                agent.active = False

                agent.hyp = DH()

    return D
```

This code is used in chunk 41a.
Defines:
D_context_sensitive, never used.

Multi-diffusion

13 ⟨*diffusion variants* 11⟩+≡

```
def D_multidiffusion13(rng, swarm, multidiffusion_amount, DH):
    def D(agent):

        if agent.inactive:

            polled_agents = (rng.choice(swarm) for num in range(multidiffusion_amount))

            active_agents = [agent for agent in polled_agents if agent.active]

            if active_agents:

                agent.hyp = rng.choice(active_agents).hyp

            else:

                agent.hyp = DH()

    return D
```

This code is used in chunk 41a.
Defines:
D_multidiffusion, never used.

1.5.1 Diffusion with noisy hypothesis transmission

Noisy diffusion

14a ⟨*diffusion variants* 11⟩+≡

```
def D_noise14a(swarm, DN, DH, rng):

    def D(agent):

        if agent.inactive:

            polled = rng.choice(swarm)

            if polled.active:

                agent.hyp = DN(polled.hyp)

            else:

                agent.hyp = DH()

    return D
```

This code is used in chunk 41a.
Defines:
D_noise, never used.

Gaussian noise

14b ⟨*diffusion noise functions* 14b⟩≡

```
def DN_gauss14b(mean, sigma, rng):
    """ add noise from a gaussian distribution to hypothesis transmission """

    def DN(hyp):

        return hyp + rng.gauss(mean, sigma)

    return DN
```

This definition is continued in chunk 15a.
This code is used in chunk 41a.
Defines:
DN_gauss, used in chunk 15a.

Normal distribution noise

15a ⟨*diffusion noise functions* 14b⟩+≡

```
def DN_normal15a(rng):
    """ add noise from a normal gaussian distribution to hypothesis
    transmission """

    DN = DN_gauss14b(mean=0, sigma=1, rng=rng)

    return DN
```

This code is used in chunk 41a.
Defines:
DN_normal, never used.
Uses DN_gauss 14b.

1.6 Modes of hypothesis selection

Uniform random

15b ⟨*uniform hypothesis selection* 15b⟩≡

```
def DH_uniform15b(hypotheses, rng):
    """ uniformly random hypothesis generation """

    def DH():

        return rng.choice(hypotheses)

    return DH
```

This code is used in chunk 40.
Defines:
DH_uniform, used in chunk 42a.

Uniform continuous random

15c ⟨*hypothesis selection variants* 15c⟩≡

```
def DH_continuous15c(min_hyp, max_hyp, rng):

    def DH():

        return rng.uniform(min_hyp, max_hyp)

    return DH
```

This code is used in chunk 41a.
Defines:
DH_continuous, never used.

1.7 Modes of testing

Boolean

16a ⟨*boolean testing* 16a⟩≡

```
def T_boolean16a(TM):
    """ Boolean testing """

    def T(agent):

        microtest = TM()

        agent.active = microtest(agent.hyp)

    return T
```

This code is used in chunk 40.
Defines:
T_boolean, used in chunks 39c and 42a.

Comparative Each agent performs a random microtest against its own hypothesis and the hypothesis of a randomly selected agent, they become active if their hypothesis returned a higher value than the hypothesis of the polled agent.

16b ⟨*testing variants* 16b⟩≡

```
def T_comparative16b(TM, swarm, rng):

    def T_prime(agent):

        microtest = TM()

        agent_partial_evaluation = microtest(agent.hyp)

        polled = rng.choice(swarm)

        polled_partial_evaluation = microtest(polled.hyp)

        agent.active = agent_partial_evaluation > polled_partial_evaluation

    return T_prime
```

This definition is continued in chunk 17a.
This code is used in chunk 41a.
Defines:
T_comparative, never used.

Multi-testing

17a ⟨*testing variants* 16b⟩+≡

```
def TM_multitesting17a(microtests, rng, multitesting_amount, combinator):

    def TM():

        microtest_sample = iter(
            rng.choice(microtests) for num in range(multitesting_amount)
        )

        def multi_test(hyp):

            return combinator(microtest(hyp) for microtest in microtest_sample)

        return multi_test

    return TM
```

This code is used in chunk 41a.
Defines:
TM_multitesting, never used.

1.8 Modes of microtest selection

Uniform random

17b ⟨*uniform microtest selection* 17b⟩≡

```
def TM_uniform17b(microtests, rng):
    """ uniform microtest selection """

    def TM():

        return rng.choice(microtests)

    return TM
```

This code is used in chunk 40.
Defines:
TM_uniform, used in chunk 42a.

1.9 Modes of halting

Fixed iteration

18 ⟨*fixed iteration halting* 18⟩≡

```
def H_fixed18(iterations):
    """ makes a function for halting after a fixed number of iterations """

    iteration_count = 0

    def H():

        nonlocal iteration_count

        iteration_count += 1

        if iteration_count > iterations:

            log.log(logging.DEBUG, "h_fixed(%s) halting", iterations)

            return True

        else:

            return False

    return H
```

This code is used in chunk 40.
Defines:
H_fixed, used in chunk 42a.

Fixed time

19 ⟨*iteration variants* 9b⟩+≡

```
def H_time19(duration):

    start = None

    def H():

        nonlocal start

        if start is None:

            start = datetime.datetime.now()

        return (now - start) > duration

    return H
```

This code is used in chunk 41a.
Defines:
H_time, never used.

Global activity

20a ⟨*halting variants* 20a⟩≡

```
def H_threshold20a(swarm, threshold):
    """ makes a function for halting once the global activity is over a fixed
    threshold """

    def H():

        activity = swarm.activity
        # return activity > threshold
        if activity > threshold:
            log.log(
                SILENT, f"Threshold activity {activity} > threshold {threshold}. halt!"
            )
            return True
        else:
            log.log(
                SILENT,
                "Threshold activity {activity} < threshold {threshold}. not halting",
            )
            return False

    return H
```

This definition is continued in chunks 20–24.
This code is used in chunk 41a.
Defines:
H_threshold, never used.

Largest cluster

20b ⟨*halting variants* 20a⟩+≡

```
def H_largest_cluster_threshold20b(swarm, threshold):
    """ makes a function for halting once the largest cluster activity is over
    a fixed threshold """

    def H():

        return swarm.largest_cluster.size >= threshold

    return H
```

This code is used in chunk 41a.
Defines:
H_largest_cluster_threshold, never used.

Unique hypothesis count

21a ⟨*halting variants* 20a⟩+≡

```
def H_unique_hyp_count21a(swarm, unique_threshold):
    def H():

        unique_hyps = len(swarm.clusters)

        return unique_hyps < unique_threshold

    return H
```

This code is used in chunk 41a.
Defines:
H_unique_hyp_count, never used.

Elite cluster consensus

21b ⟨*halting variants* 20a⟩+≡

```
def H_elite_cluster_consensus21b(swarm, elite_count, rng):

    elite_agents = rng.sample(swarm, elite_count)

    def H():

        elite_agent_gen = (agent for agent in swarm if agent in elite_agents)

        first_elite_agent = next(elite_agent_gen)

        elite_hyp = first_elite_agent.hyp

        return first_elite_agent.active and all(
            elite_agent.active and elite_agent.hyp == elite_hyp
            for elite_agent in elite_agent_gen
        )

    return H
```

This code is used in chunk 41a.
Defines:
H_elite_cluster_consensus, never used.

Global activity stability

22 ⟨*halting variants* 20a⟩+≡

```
def H_stable22(swarm, max_memory_length, stability_threshold, min_stable_iterations):

    memory = collections.deque(maxlen=max_memory_length)

    stable_iterations = 0

    def H():

        nonlocal stable_iterations

        activity = swarm.activity

        memory.append(activity)

        mean_activity = sum(memory) / len(memory)

        deviations = [activity - mean_activity for activity in memory]

        sum_of_squared_deviations = sum(pow(deviation, 2) for deviation in deviations)

        standard_deviation = math.sqrt(sum_of_squared_deviations / len(memory))

        if standard_deviation > stability_threshold:

            stable_iterations = 0

            return False

        stable_iterations += 1

        is_stable = (stable_iterations >= min_stable_iterations)

        return is_stable

    return H
```

This code is used in chunk 41a.
Defines:
H_stable, never used.

Weak halting criterion

23 ⟨*halting variants* 20a⟩+≡

```
def H_weak23(swarm, threshold_activity, stability_threshold, min_stable_iterations):

    stable_iterations = 0

    if not (
        ((2 * stability_threshold) < 1)
        and ((stability_threshold + threshold_activity) <= 1)
        and (threshold_activity - stability_threshold >= 0)
    ):

        raise ValueError("not valid values")

    def H():

        nonlocal stable_iterations

        activity = swarm.activity

        stability = abs(activity - threshold_activity)

        if stability < stability_threshold:

            stable_iterations += 1

        else:

            stable_iterations = 0

        return stable_iterations > min_stable_iterations

    return H
```

This code is used in chunk 41a.
Defines:
H_weak, never used.

Strong halting criterion

24 ⟨*halting variants* 20a⟩+≡

```
def H_strong24(
    swarm, threshold_cluster_size, stability_threshold, min_stable_iterations  # a  # b
):

    stable_iterations = 0

    swarm_size = len(swarm)

    if not (
        ((2 * stability_threshold) < swarm_size)
        and ((stability_threshold + threshold_cluster_size) <= swarm_size)
        and (threshold_cluster_size - stability_threshold >= 0)
    ):

        raise ValueError(
            (
                f"not valid values. "
                f"((2 * stability_threshold) < 1) {((2 * stability_threshold) < 1)}, "
                f"((stability_threshold + threshold_cluster_size) <= 1) {((stability_threshold
                f"(threshold_cluster_size - stability_threshold >= 0) {(threshold_cluster_size
            )
        )

    def H():

        nonlocal stable_iterations

        cluster_size = swarm.largest_cluster.agents

        stability = abs(cluster_size - threshold_cluster_size)

        if stability < stability_threshold:

            stable_iterations += 1

        else:

            stable_iterations = 0

        return stable_iterations > min_stable_iterations

    return H
```

This code is used in chunk 41a.
Defines:
H_strong, never used.

1.9.1 Halting combinators

All functions

25a ⟨*halting combinators* 25a⟩≡

```
def all_functions25a(*function_list):
    def F():

        results = [function() for function in function_list]

        log.log(SILENT, "all functions %s", results)

        return all(results)

    return F
```

This definition is continued in chunk 25b.
This code is used in chunk 41a.
Defines:
`all_functions`, never used.

Any functions

25b ⟨*halting combinators* 25a⟩+≡

```
def any_functions25b(*function_list):
    def F():

        results = [function() for function in function_list]

        log.log(SILENT, "any functions %s", results)

        return any(results)

    return F
```

This code is used in chunk 41a.
Defines:
`any_functions`, used in chunk 39b.

1.10 Modes of extraction

Rounded clusters

26 ⟨*extraction functions* 26⟩≡

```
def round_clusters26(clusters):

    rounded_clusters = collections.Counter()

    for hyp, size in clusters.items():

        rounded_clusters[round(hyp)] += size

    return rounded_clusters
```

This code is used in chunk 41a.
Defines:
round_clusters, never used.

Chapter 2

Reducing SDS

Confirmation reducing diffusion

27 ⟨*reducing diffusion* 27⟩≡

```
def D_confirmation27(swarm, removed_clusters, DH, rng):

    non_removed_agents = [agent for agent in swarm if not agent.removed]

    def D(agent):

        if agent.removed:

            return

        polled = rng.choice(non_removed_agents)

        if agent.active:

            if polled.active and agent.hyp == polled.hyp:

                agent.terminating = True

        else:

            if polled.active:

                if polled.terminating:

                    agent.remove(final_hyp=polled.hyp)
                    non_removed_agents.remove(agent)

                    removed_clusters[polled.hyp] += 1
```

```
                else:

                    agent.hyp = polled.hyp

            else:

                agent.hyp = DH()

    return D
```

This definition is continued in chunks 29, 31, and 33.
This code is used in chunk 41b.
Defines:
D_confirmation, never used.

Independent reducing diffusion

29 ⟨*reducing diffusion* 27⟩+≡

```
def D_independent29(swarm, removed_clusters, DH, rng):

    remaining_swarm = [agent for agent in swarm if not agent.removed]

    swarm_is_empty = False

    def D(agent):

        nonlocal swarm_is_empty

        if agent.removed or swarm_is_empty:

            return

        while True:

            polled = rng.choice(remaining_swarm)

            if polled is not agent:

                break

        if agent.inactive and polled.inactive:

            agent.hyp = DH()
            polled.hyp = DH()

        elif agent.active and (not agent.terminating) and polled.inactive:

            polled.hyp = agent.hyp

        elif polled.active and (not polled.terminating) and agent.inactive:

            agent.hyp = polled.hyp

        elif agent.terminating and not polled.terminating:

            polled.remove(final_hyp=agent.hyp)
            remaining_swarm.remove(polled)
            swarm_is_empty = len(remaining_swarm) < 2

            removed_clusters[agent.hyp] += 1

        elif polled.terminating and not agent.terminating:

            agent.remove(final_hyp=polled.hyp)
            remaining_swarm.remove(agent)
```

```
            swarm_is_empty = len(remaining_swarm) < 2

            removed_clusters[polled.hyp] += 1

        elif agent.terminating and polled.terminating:

            both_agents = [agent, polled]
            rng.shuffle(both_agents)
            removed, removing = both_agents

            removed.remove(final_hyp=removing.hyp)
            remaining_swarm.remove(removed)
            swarm_is_empty = len(remaining_swarm) < 2

            removed_clusters[removing.hyp] += 1

        elif agent.hyp == polled.hyp:

            agent.terminating = polled.terminating = True

    return D
```

This code is used in chunk 41b.
Defines:
D_independent, never used.

Running mean diffusion

31 ⟨*reducing diffusion* 27⟩+≡

```
def D_running_mean31(DH, quorum_threshold, min_interaction_count, activities, swarm, rng):

    non_removed_agents = [agent for agent in swarm if not agent.removed]

    swarm_is_empty = False

    def D(agent):

        nonlocal swarm_is_empty


        if agent.removed or swarm_is_empty:

            return

        polled = rng.choice(non_removed_agents)

        if agent.inactive:

            agent.memory.clear()

            if polled.active:

                agent.hyp = polled.hyp

            else:

                agent.hyp = DH()

        else:  # agent is active

            if agent.terminating:

                if not (agent.hyp == polled.hyp):

                    polled.remove(final_hyp=agent.hyp)
                    non_removed_agents.remove(polled)
                    swarm_is_empty = len(non_removed_agents) < 2

            else:  # agent has not sensed quorum

                activity_at_hypothesis = activities[agent.hyp]/len(non_removed_agents)

                agent.memory.append(activity_at_hypothesis)

                interaction_count = len(agent.memory)
```

```
                # confidence is 0 if interaction_count < min_iteration_count
                confidence = (
                    interaction_count >= min_interaction_count
                    and (sum(agent.memory) / interaction_count)
                )

                if confidence >= quorum_threshold:
                    # agent has sensed quorum

                    agent.terminating = True

    return D
```

This code is used in chunk 41b.
Defines:
D_running_mean, never used.

Quorum sensing diffusion

33 ⟨*reducing diffusion* 27⟩+≡

```
def D_qs33(DH, quorum_threshold, decay, swarm, rng):

    non_removed_agents = [agent for agent in swarm if not agent.removed]

    swarm_is_empty = False

    def D(agent):

        nonlocal swarm_is_empty

        if agent.removed or swarm_is_empty:

            return

        polled = rng.choice(non_removed_agents)

        if agent.inactive:

            agent.confidence = 0

            if polled.active:

                agent.hyp = polled.hyp

            else:

                agent.hyp = DH()

        else:  # agent is active

            if agent.terminating:

                if not (agent.hyp == polled.hyp):

                    polled.remove(final_hyp=agent.hyp)
                    non_removed_agents.remove(polled)
                    swarm_is_empty = len(non_removed_agents) < 2

            else:  # agent has not sensed quorum

                if polled.active and (agent.hyp == polled.hyp):

                    agent.confidence += 1

                agent.confidence *= decay

                if agent.confidence >= quorum_threshold:  # agent has sensed quorum
```

```
                    agent.terminating = True

    return D
```

This code is used in chunk 41b.
Defines:
D_qs, never used.

2.0.1 Agent and Swarm subclasses

Reducing agent

35a ⟨*reducing agent* 35a⟩≡

```
class ReducingAgent35a(sds.Agent5):
    def __init__(self, active=False, hyp=None, terminating=False, removed=False):
        super().__init__(active=active, hyp=hyp)
        self.terminating = terminating
        self.removed = removed

    def remove(self, final_hyp):

        self.removed = True
        self.hyp = final_hyp
        self.active = False
        self.terminating = False

    def __str__(self):

        s = super().__str__()

        if self.terminating:

            s = f"{s} Terminating"

        elif self.removed:

            s = f"{self.hyp} Removed"

        return s

    def __iter__(self):

        yield from super().__iter__()
        yield ("terminating", self.terminating)
        yield ("removed", self.removed)
```

This definition is continued in chunks 36 and 37.
This code is used in chunk 41b.
Defines:
`ReducingAgent`, used in chunks 36 and 37.
Uses `Agent` 5.

35b ⟨*reducing imports* 35b⟩≡

```
import sds.standard
```

This code is used in chunk 41b.

Quorum sensing agent

36 ⟨*reducing agent* 35a⟩+≡

```
class QSAgent36(ReducingAgent35a):
    def __init__(
        self, active=False, hyp=None, terminating=False, removed=False, confidence=0
    ):
        super().__init__(
            active=active, hyp=hyp, terminating=terminating, removed=removed
        )
        self.confidence = confidence

    def __str__(self):

        s = super().__str__()

        if self.active:

            s = f"{s} Confidence: {self.confidence:2.2g}"

        return s

    def __iter__(self):

        yield from super().__iter__(self)
        yield ("confidence", self.confidence)
```

This code is used in chunk 41b.
Defines:
QSAgent, never used.
Uses ReducingAgent 35a.

Running mean agent

37 ⟨*reducing agent* 35a⟩+≡

```
class QSRunningMeanAgent37(ReducingAgent35a):
    def __init__(self, active, hyp, terminating, removed, memory):

        super().__init__(
            active=active, hyp=hyp, terminating=terminating, removed=removed
        )

        self.memory = memory

    def new(memory_length):

        return QSRunningMeanAgent37(
            active=False,
            hyp=None,
            terminating=False,
            removed=False,
            memory=collections.deque(maxlen=memory_length),
        )

    def __str__(self):

        s = super().__str__()

        if self.active:

            memory_str = ", ".join(format(avg, ".2g") for avg in self.memory)

            s = f"{s} Confidence: [{memory_str}]"

        return s

    def __iter__(self):

        yield from super().__iter__(self)
        yield ("memory", self.memory)
```

This code is used in chunk 41b.
Defines:
QSRunningMeanAgent, never used.
Uses ReducingAgent 35a.

Reducing swarm

38a ⟨*reducing swarm* 38a⟩≡

```
class ReducingSwarm38a(sds.standard.Swarm7):
    @property
    def clusters(self):

        return collections.Counter(
            agent.hyp for agent in self if agent.active or agent.removed
        )

    @property
    def size(self):

        return len(self) - len(self.removed)

    @property
    def removed(self):

        return [agent for agent in self if agent.removed]
```

This code is used in chunk 41b.
Defines:
ReducingSwarm, never used.
Uses Swarm 7.

Reducing halting

38b ⟨*reducing halting* 38b⟩≡

```
def H_all_terminating38b(swarm):
    def H():

        halt = all(agent.terminating for agent in swarm if not agent.removed)

        return halt

    return H
```

This definition is continued in chunk 39.
This code is used in chunk 41b.
Defines:
H_all_terminating, used in chunk 39b.

39a ⟨*reducing halting* 38b⟩+≡

```
def H_empty_swarm39a(swarm):
    def H():
        return all(agent.removed for agent in swarm)

    return H
```

This code is used in chunk 41b.
Defines:
H_empty_swarm, used in chunk 39b.

39b ⟨*reducing halting* 38b⟩+≡

```
def H_reducing39b(swarm):

    is_empty = H_empty_swarm39a(swarm)
    is_all_terminating = H_all_terminating38b(swarm)

    return halting_methods.any_functions25b(is_empty, is_all_terminating)
```

This code is used in chunk 41b.
Defines:
H_reducing, never used.
Uses any_functions 25b, H_all_terminating 38b, and H_empty_swarm 39a.

Reducing testing

39c ⟨*reducing testing* 39c⟩≡

```
def T_reducing39c(TM):

    T_inner = sds.standard.T_boolean16a(TM=TM)

    def T(agent):

        if not (agent.terminating or agent.removed):

            T_inner(agent)

    return T
```

This code is used in chunk 41b.
Defines:
T_reducing, never used.
Uses T_boolean 16a.

Appendix A

Files

A.1 `standard.py`

40 ⟨*standard.py* 40⟩≡
⟨*standard imports* 8a⟩
```
import logging
```
⟨*init logging* 42c⟩
⟨*standard agent* 5⟩
⟨*standard swarm* 7⟩
⟨*cluster* 8b⟩
⟨*sds* 4⟩
⟨*synchronous iteration* 9a⟩
⟨*passive diffusion* 10⟩
⟨*uniform hypothesis selection* 15b⟩
⟨*uniform microtest selection* 17b⟩
⟨*fixed iteration halting* 18⟩
⟨*boolean testing* 16a⟩
Root chunk (not used in this document).

A.2 variants.py

41a ⟨*variants.py* 41a⟩≡
⟨*iteration variants* 9b⟩
⟨*diffusion variants* 11⟩
⟨*diffusion noise functions* 14b⟩
⟨*hypothesis selection variants* 15c⟩
⟨*testing variants* 16b⟩
⟨*halting variants* 20a⟩
⟨*halting combinators* 25a⟩
⟨*extraction functions* 26⟩
Root chunk (not used in this document).

A.3 reducing.py

41b ⟨*reducing.py* 41b⟩≡
⟨*reducing imports* 35b⟩
```
import logging
```
⟨*init logging* 42c⟩
⟨*reducing diffusion* 27⟩
⟨*reducing agent* 35a⟩
⟨*reducing swarm* 38a⟩
⟨*reducing halting* 38b⟩
⟨*reducing testing* 39c⟩
Root chunk (not used in this document).

A.4 __init__.py

42a ⟨*–init–.py* 42a⟩≡

```
from sds.standard import (
    Agent5,
    D_passive10,
    DH_uniform15b,
    H_fixed18,
    I_sync9a,
    SDS4,
    Swarm7,
    T_boolean16a,
    TM_uniform17b,
)
__all__ = [
    "Agent5",
    "D_passive10",
    "DH_uniform15b",
    "H_fixed18",
    "I_sync9a",
    "SDS4",
    "Swarm7",
    "T_boolean16a",
    "TM_uniform17b",
]
```

Root chunk (not used in this document).
Uses Agent 5, D_passive 10, DH_uniform 15b, H_fixed 18, I_sync 9a, SDS 4, Swarm 7, T_boolean 16a, and TM_uniform 17b.

A.5 test-sds.py

42b ⟨*test-sds.py* 42b⟩≡

```
⟨test imports 43⟩
class TestSDS(unittest.TestCase):
    def setUp(self):
        logging.basicConfig(level=logging.INFO)
        self.log = logging.getLogger(__file__)
    ⟨unit tests 6⟩
```

Root chunk (not used in this document).

42c ⟨*init logging* 42c⟩≡

```
log = logging.getLogger(__name__)
```

This code is used in chunks 40 and 41b.

43 ⟨*test imports* 43⟩≡

```
import unittest
import sds
import sds.standard
import sds.reducing
import sds.variants
import logging
```

This code is used in chunk 42b.

Appendix B

Indices

B.1 Index

B.2 Code Chunks

⟨*standard imports* 8a⟩ 8a, 40
⟨*standard swarm* 7⟩ 7, 40
⟨*standard.py* 40⟩ 40
⟨*synchronous iteration* 9a⟩ 9a, 40
⟨*test imports* 43⟩ 42b, 43
⟨*test-sds.py* 42b⟩ 42b
⟨*testing variants* 16b⟩ 16b, 17a, 41a
⟨*uniform hypothesis selection* 15b⟩ 15b, 40
⟨*uniform microtest selection* 17b⟩ 17b, 40
⟨*unit tests* 6⟩ 6, 42b
⟨*variants.py* 41a⟩ 41a