Tufts UNIVERSITY

Metamorph: Synthesizing Large Objects from Dafny Specifications

Aleksandr Fedchin, **Alexander Y. Bai**, Jeffrey S. Foster

Key Challenge

- Testing in OOP context may require setting up complex objects that have private fields
- Challenge: generating programs (sequences of method calls) that produce objects with required properties
- Our idea: use method specifications in a verification-aware setting to guide the synthesis

Example Dafny API

Postconditions are assumed after method calls

Preconditions must be proved before method calls

```
datatype User = User(friends:set<nat>,
                     requests:set<nat>)
class SocialNetwork {
 ghost var users: map<nat,User>
 constructor() ensures users == map[] {...}

 method AddUser(id:nat)
  requires id ∉ users
  ensures users == old(users)[id:=User({},{})]
 {...}

 method RemoveUser(id:nat)...
 method FriendRequest(from:nat, to:nat)...
 method RequestResponse
        (from:nat, to:nat, answer:bool)... }
```

Example Dafny API

Ghost fields are unavailable at runtime, cannot be set directly

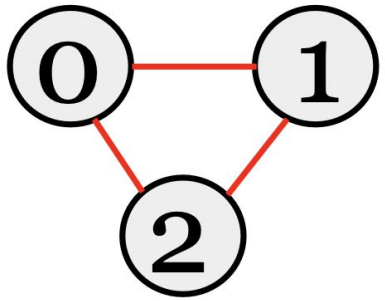
Method bodies hidden for modular verification, may not be implemented in Dafny

```
datatype User = User(friends:set<nat>,
                     requests:set<nat>)
class SocialNetwork {
 ghost var users: map<nat,User>
 constructor() ensures users == map[] {...}

 method AddUser(id:nat)
  requires id ∉ users
  ensures users == old(users)[id:=User({},{})]
 {...}

 method RemoveUser(id:nat)...
 method FriendRequest(from:nat, to:nat)...
 method RequestResponse
        (from:nat, to:nat, answer:bool)... }
```

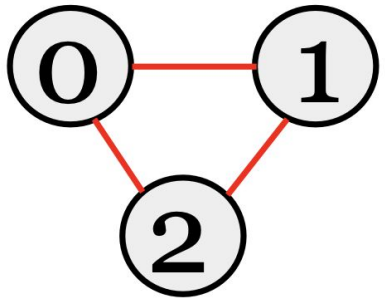
Example Synthesis Problem



(describes a social network with 3 users, all of whom are friends)

```
method Goal()
  returns (s:SocialNetwork)
  ensures s.users.Keys == {0, 1, 2}
        ∧ 1 ∈ s.users[0].friends
        ∧ 1 ∈ s.users[2].friends
        ∧ 2 ∈ s.users[0].friends
{
  s := new SocialNetwork();
  s.AddUser(0); s.AddUser(1);
  s.FriendRequest(0, 1);
  s.RequestResponse(1, 0, true);
  s.AddUser(2);
  s.FriendRequest(0, 2);
  s.RequestResponse(2, 0, true);
  s.FriendRequest(1, 2);
  s.RequestResponse(2, 1, true);
}
```

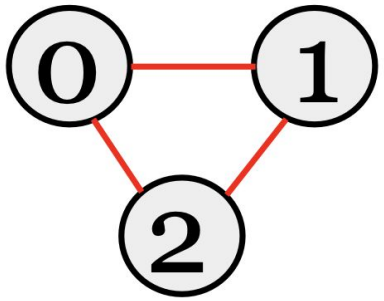
Example Synthesis Problem



(describes a social network with 3 users, all of whom are friends)

```
method Goal()
  returns (s:SocialNetwork)
  ensures s.users.Keys == {0, 1, 2}
        ∧ 1 ∈ s.users[0].friends
        ∧ 1 ∈ s.users[2].friends
        ∧ 2 ∈ s.users[0].friends
{
  s := new SocialNetwork();
  s.AddUser(0); s.AddUser(1);
  s.FriendRequest(0, 1);
  s.RequestResponse(1, 0, true);
  s.AddUser(2);
  s.FriendRequest(0, 2);
  s.RequestResponse(2, 0, true);
  s.FriendRequest(1, 2);
  s.RequestResponse(2, 1, true); }
```

Example Synthesis Problem



(desribes a social network with 3 users, all of whom are friends)

```
method Goal()
  returns (s:SocialNetwork)
  ensures s.users.Keys == {0, 1, 2}
        ∧ 1 ∈ s.users[0].friends
        ∧ 1 ∈ s.users[2].friends
        ∧ 2 ∈ s.users[0].friends
{
  s := new SocialNetwork();
  s.AddUser(); s.AddUser();
  s.FriendRequest(      );
  s.RequestResponse(            );
  s.AddUser();
  s.FriendRequest(      );
  s.RequestResponse(            );
  s.FriendRequest(      );
  s.RequestResponse(            ); }
```

Problem: Inferring all primitive arguments at once may take minutes even if the method sequence is already known

Key Ideas

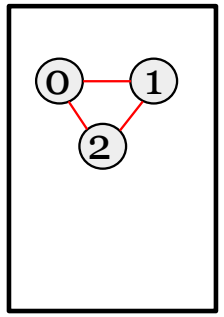
- Perform goal-directed backward synthesis
  (goal = an object state = conjunction of constraints)

- Reconstruct method arguments and prior objects states from verifier counterexamples

- Guide synthesis to minimize the number of unsatisfied constraints

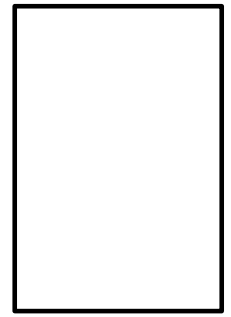
Example

Target State



(3 users, all of whom are friends)

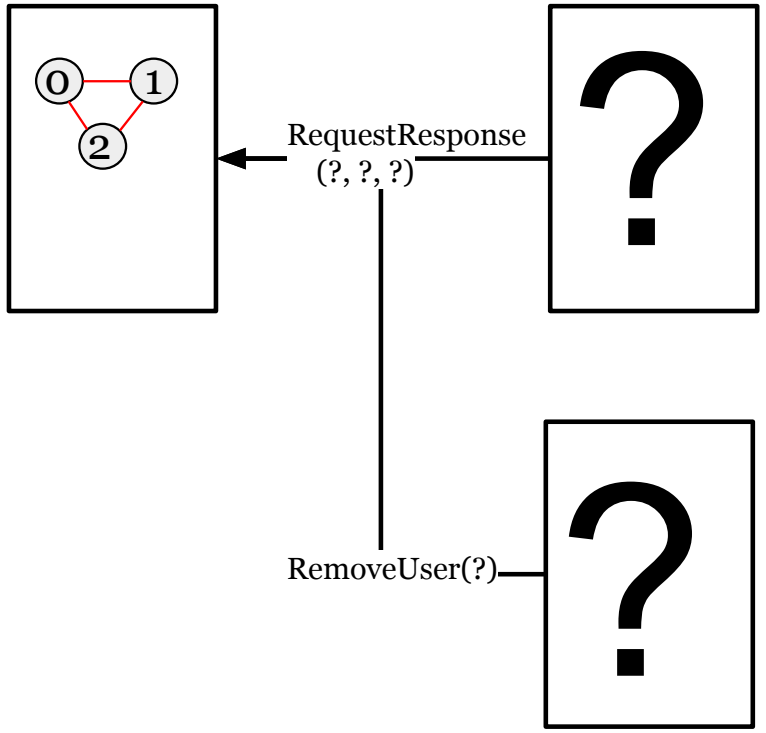
Initial State



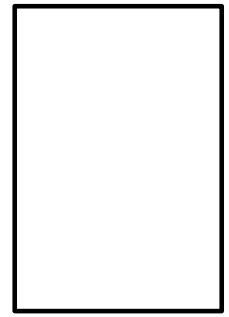
(empty social network)

Example

Target State

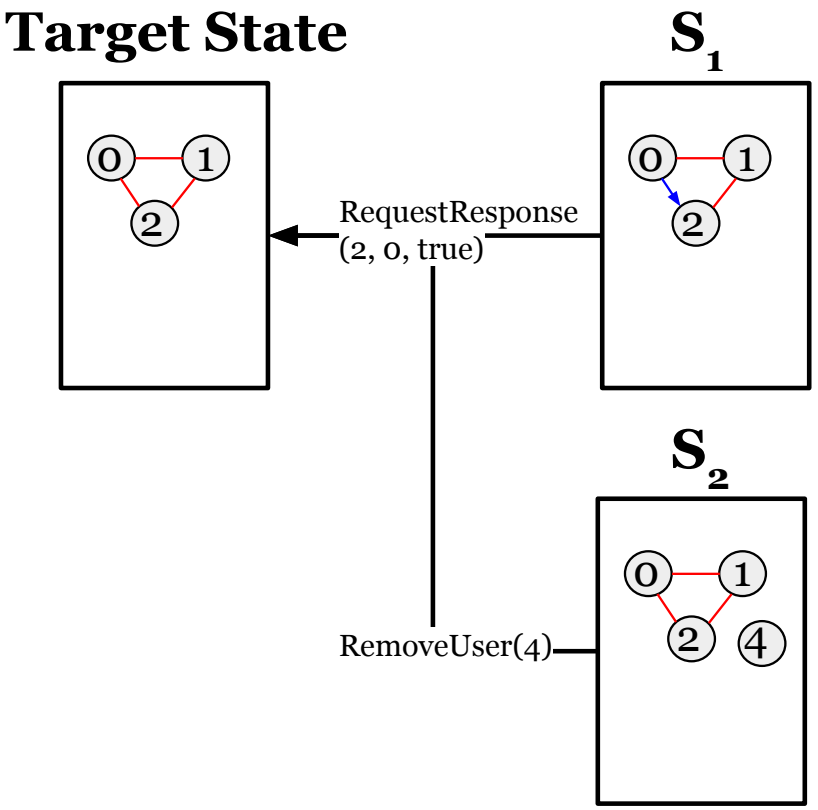


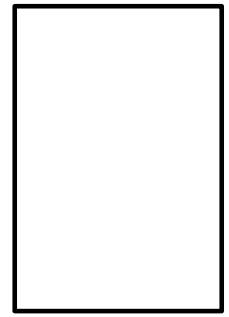
Initial State



Asking the verifier to infer the method arguments and preceding states

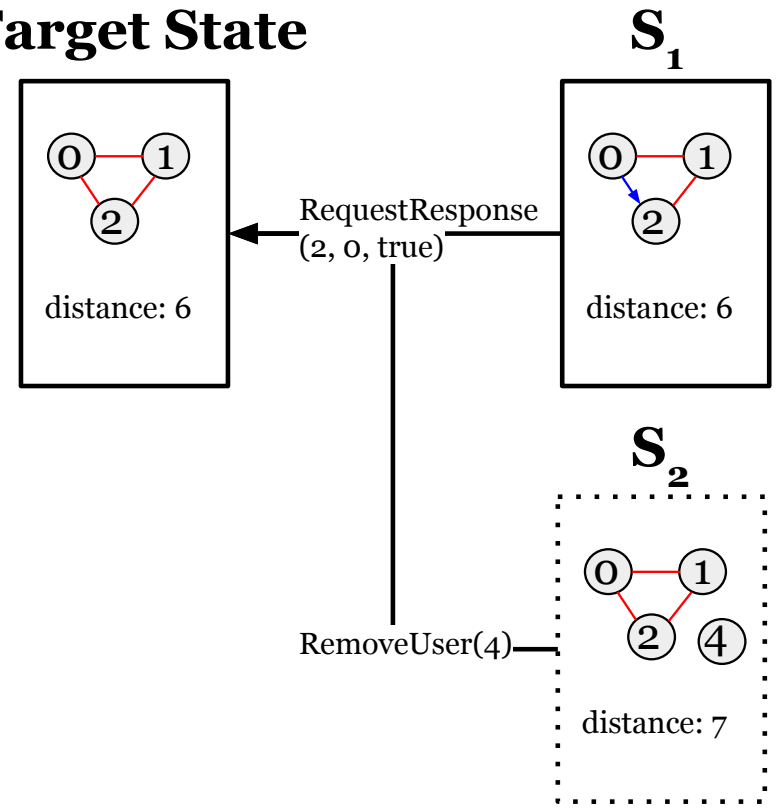
Example





Which of the two states to explore first?

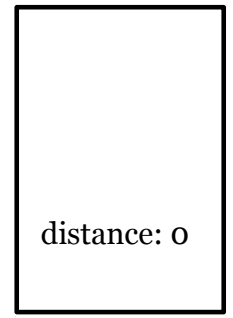
Example: Piecewise Distance Metric



to add 3 users, ≥ 3 calls to AddUser*
to add 2 connections, ≥ 2 calls to RequestResponse
to add 1 request, ≥ 1 call to FriendRequest
TOTAL ≥ 6 calls

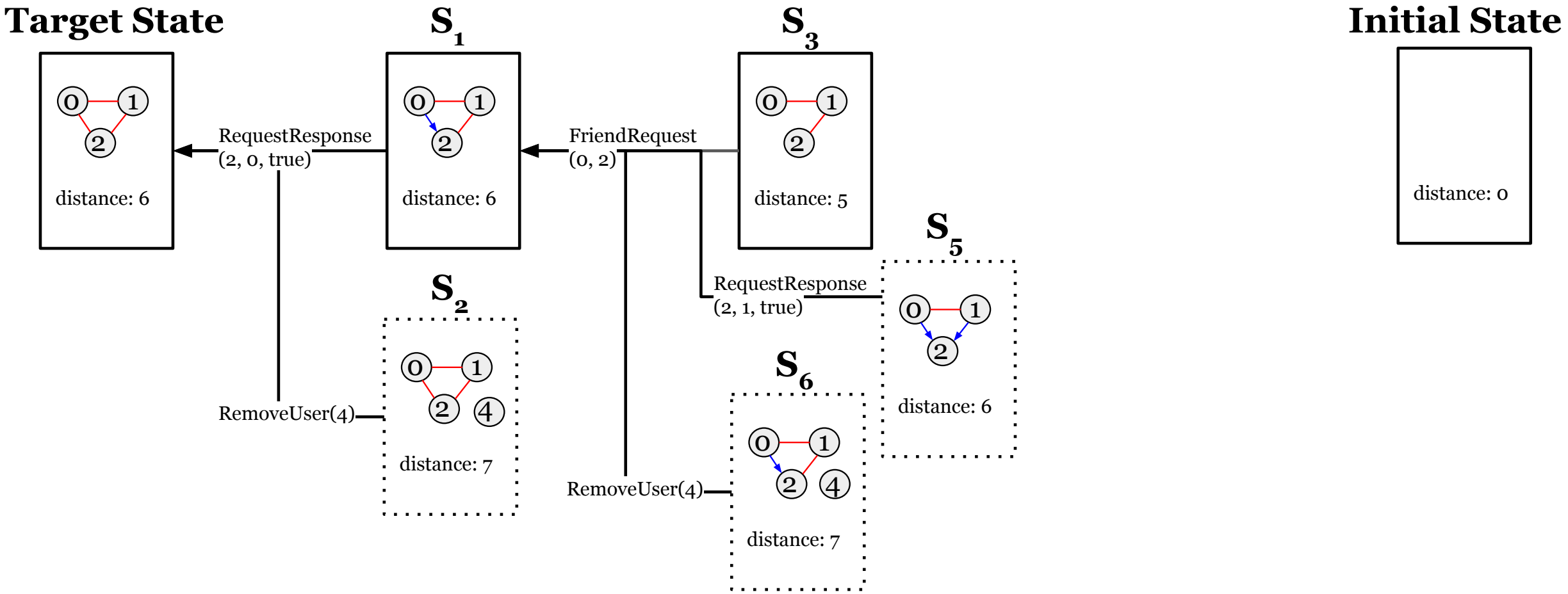
Better!

to add 4 users, ≥ 4 calls to AddUser
to add 3 connections, ≥ 3 calls to RequestResponse
TOTAL ≥ 7 calls

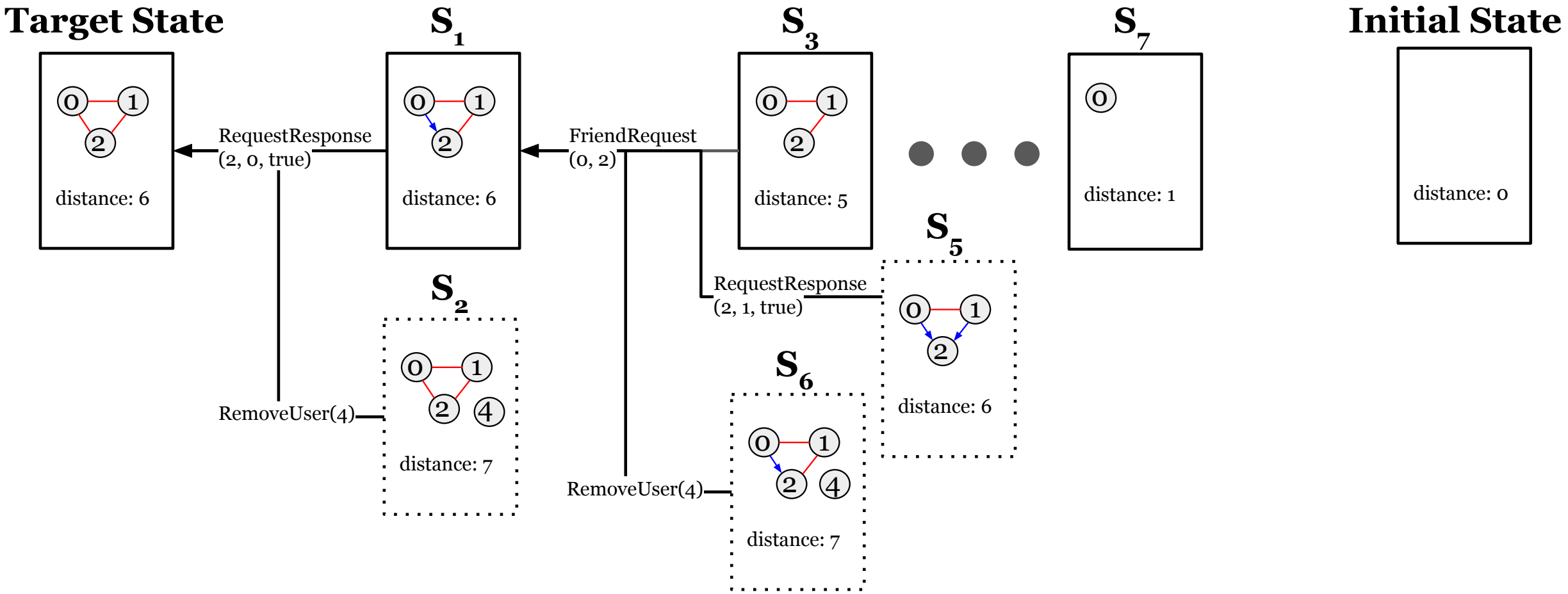


*Because Dafny verifier proves that a single call to AddUser adds at most one user and that no other method can add users.

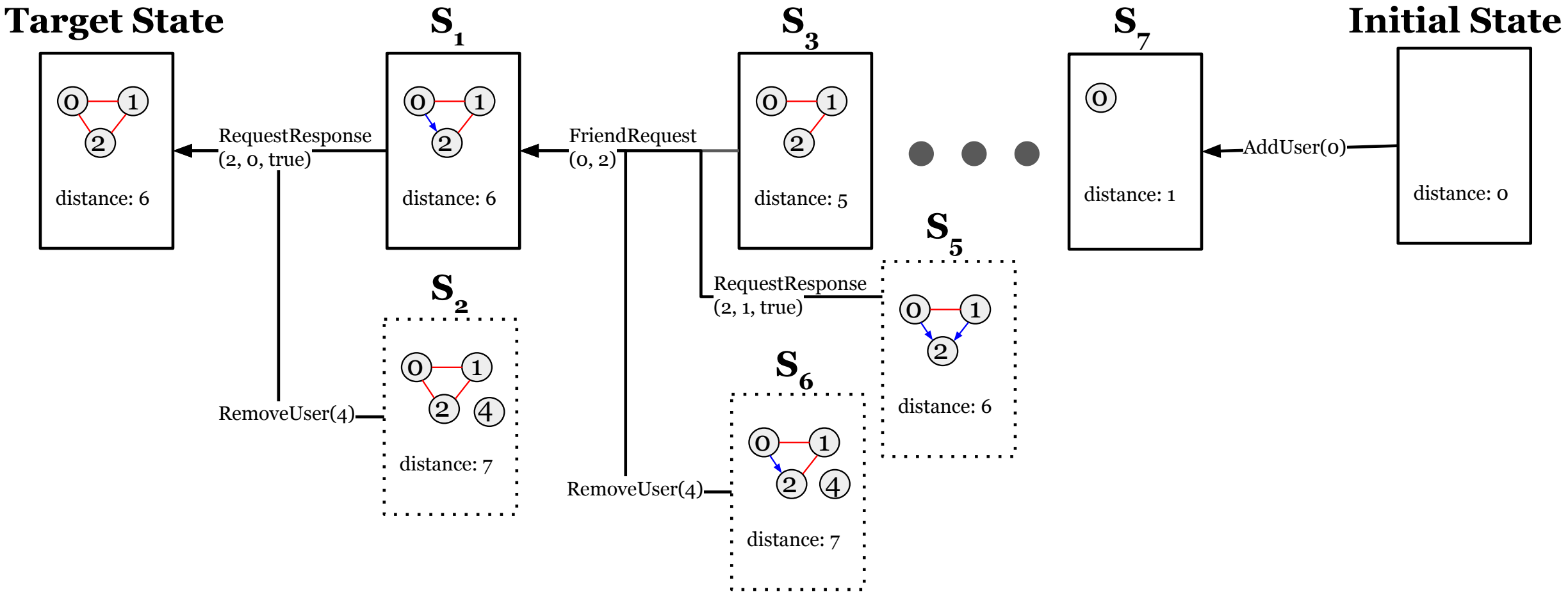
Example


Target State
distance: 6
RequestResponse
(2, 0, true)
S1
distance: 6
FriendRequest
(0, 2)
S3
distance: 5
Initial State
distance: 0
S2
RemoveUser(4)
distance: 7
RequestResponse
(2, 1, true)
S5
distance: 6
S6
RemoveUser(4)
distance: 7

Example


Target State
distance: 6
RequestResponse
(2, 0, true)
S1
distance: 6
FriendRequest
(0, 2)
S3
distance: 5
S7
distance: 1
Initial State
distance: 0
S2
distance: 7
RemoveUser(4)
S5
RequestResponse
(2, 1, true)
distance: 6
S6
RemoveUser(4)
distance: 7

Example


Target State
distance: 6
RequestResponse
(2, 0, true)
S1
distance: 6
FriendRequest
(0, 2)
S3
distance: 5
S7
distance: 1
AddUser(0)
Initial State
distance: 0
S2
distance: 7
RemoveUser(4)
RequestResponse
(2, 1, true)
S5
distance: 6
S6
distance: 7
RemoveUser(4)

Evaluation

We evaluate Metamorph on 8 APIs:

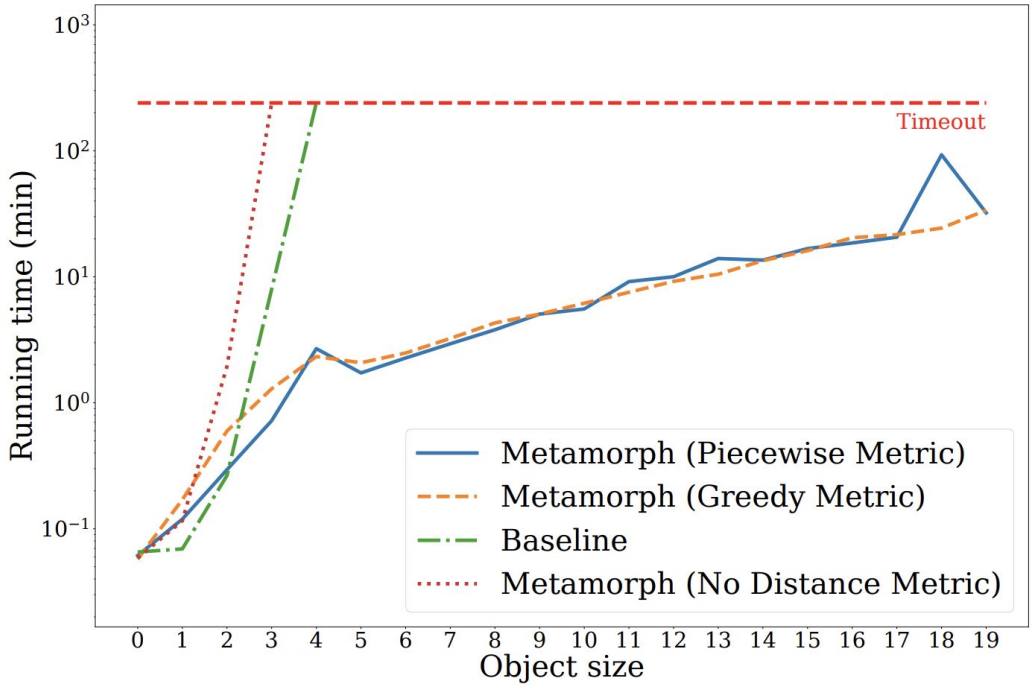
- Binary Tree
- Freezable Array*
- Doubly Linked List
- Queue
- Firewall* (graph-like API)
- Social Network
- Simple Virtual Machine
- HKDF* (from AWS ESDK)

*adapted from prior studies

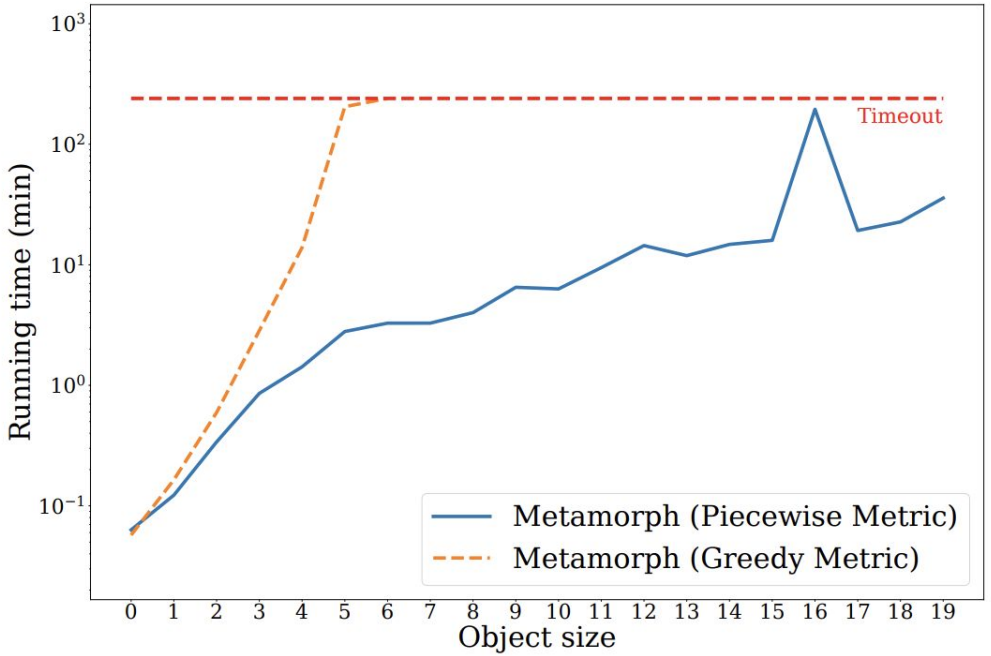
We compare the synthesis time of several approaches on increasingly larger objects:

1. Baseline - asking Dafny verifier to prove that no program generates the desired object, reconstruct the result from the counterexample.
2. Metamorph (No Distance Metric)
3. Metamorph (Piecewise Distance Metric)
4. Metamorph (Greedy Distance Metric, counts number of constraints that change)

Evaluation: Social Network Benchmark



(a) Social Network.



(b) Social Network with Relaxed Object Invariant.

Summary

- We use counterexamples to reason about effects of method calls
- We propose 2 distance metrics to guide the search over object states
- We evaluate Metamorph on 8 APIs
- Metamorph can synthesize programs up to 57 method calls
- We found that Metamorph performs better when a distance metric is used, piecewise distance metric is better than the greedy metric in select cases (e.g., incomplete API specification)
- We prototyped an integration of Metamorph with Dafny Testing Toolkit

Backup Slides

Relaxed Object Invariant

```
datatype User = User(friends:set<nat>,
                     requests:set<nat>)
class SocialNetwork {
  ...
    predicate IsValid() reads this {
    && (forall u1, u2 ::
      // removed "u2 in users" condition
       (u1 in users && u2 in users[u1].requests) ==> (u1 != u2))
    ...
    }
}
```

Guiding the Search: Distance Metrics

Greedy Distance →
Best-first Search

3 constraints change

State A

```
&& 0 in users
&& 1 in users
&& 2 in users
```

Piecewise Distance →
A* search
(more in the paper)

≥ 1 method call
to RemoveAllUsers

State B

```
&& 0 !in users
&& 1 !in users
&& 2 !in users
```