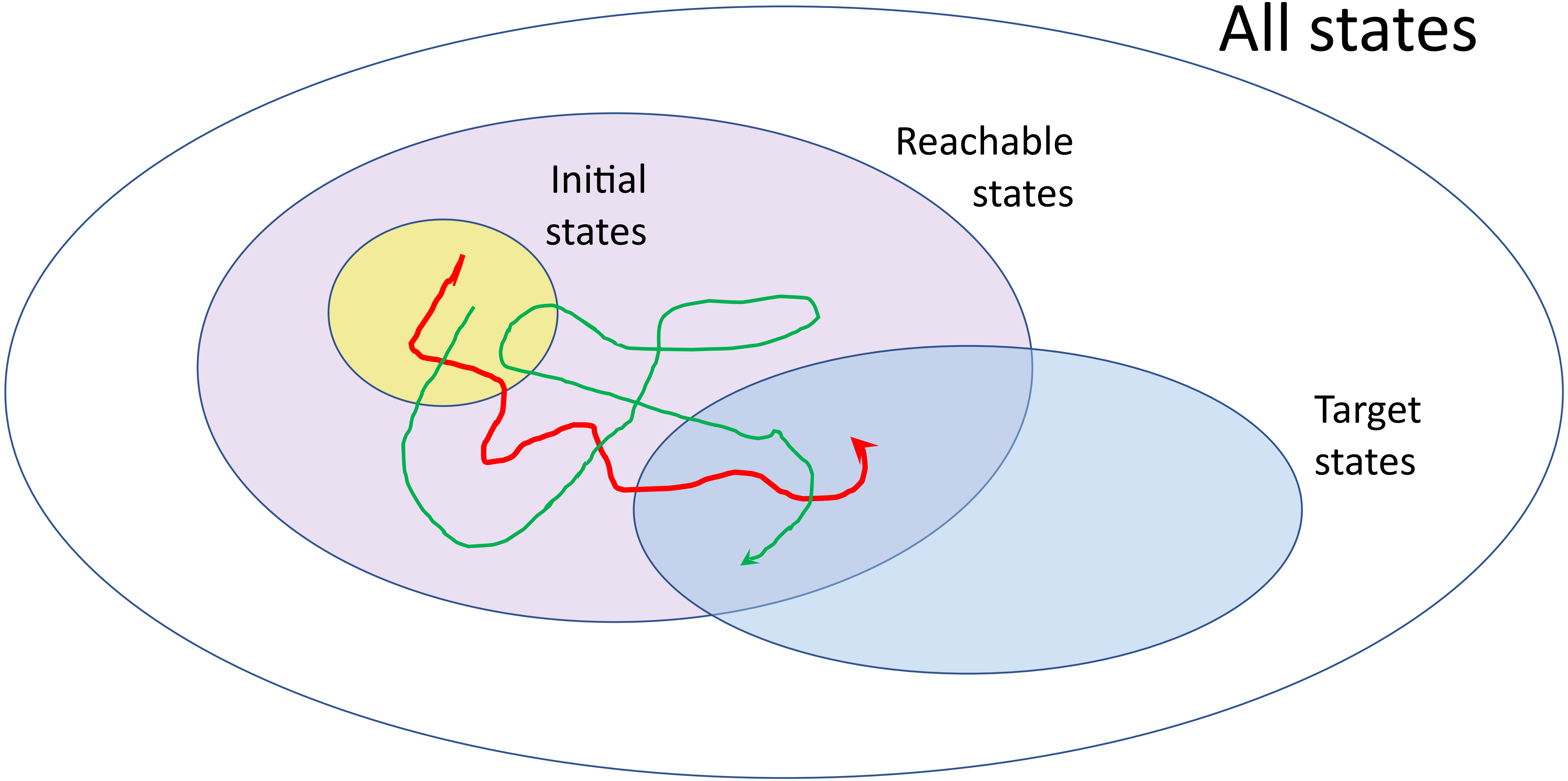
Cornell CS6480
Lecture 3
Dafny

Robbert van Renesse

Review



Review

- Behavior: infinite sequence of states
- Specification: characterizes all possible/desired behaviors
- Consists of conjunction of
  - State predicate for the initial states
  - Action predicate characterizing steps
  - Fairness formula for liveness
- TLA+ formulas are temporal formulas invariant to stuttering
  - Allows TLA+ specs to be part of an overall system


And now
for something
completely different...

Introduction to Dafny

What’s Dafny?

- An imperative programming language
- A (mostly functional) specification language
- A compiler
- A verifier

Dafny programs rule out

- Runtime errors:
  - Divide by zero
  - Array index out of bounds
  - Null reference
- Infinite loops or recursion
- Implementations that do not satisfy the specifications
  - But it’s up to you to get the latter correct

Example 1a: Abs()

```
method Abs(x: int) returns (x': int)
    ensures x' >= 0
{
    x' := if x < 0 then -x else x;
}

method Main()
{
    var x := Abs(-3);
    assert x >= 0;
    print x, "\n";
}
```

Example 1b: Abs()

```
method Abs(x: int) returns (x': int)
   ensures x' >= 0
{
   x' := 10;
}

method Main()
{
   var x := Abs(-3);
   assert x >= 0;
   print x, "\n";
}
```

Example 1c: Abs()

```
method Abs(x: int) returns (x': int)
    ensures x' >= 0
    ensures if x < 0 then x' == -x else x' == x
{
    x' := 10;
}

method Main()
{
    var x := Abs(-3);
    print x, "\n";
}
```

Example 1d: Abs()

```
method Abs(x: int) returns (x': int)
    ensures x' >= 0
    ensures if x < 0 then x' == -x else x' == x
{
    if x < 0 {
        x' := -x;
    } else {
        x' := x;
    }
}
```

Example 1e: Abs()

```
method Abs(x: int) returns (x': int)
    ensures x' >= 0
    ensures x < 0 ==> x' == -x
    ensures x >= 0 ==> x' == x
{
    if x < 0 {
        x' := -x;
    } else {
        x' := x;
    }
}
```

Example 1f: Abs()

No code generated

```
function abs(x: int): int {  if x < 0 then -x else x  }

method Abs(x: int) returns (x': int)
    ensures x' >= 0
    ensures x' == abs(x)
{
    x' := x;
    if x' < 0 {  x' := x' * -1;  }
}
```

Example 1g: Abs()

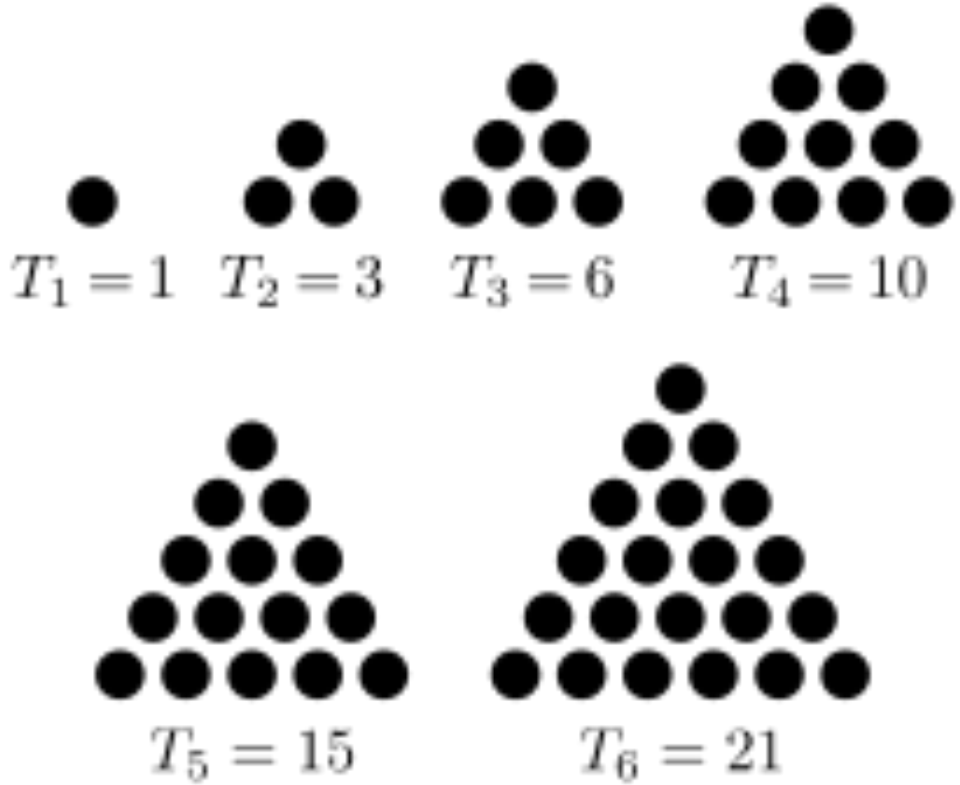
Code generated

```
function method abs(x: int): int {
    if x < 0 then -x else x
}

method Abs(x: int) returns (x': int)
    ensures x' >= 0
    ensures x' == abs(x)
{
    x' := abs(x);
}
```

Loop Invariants

```
method TriangleNumber(N: int) returns (t: int)
  requires N >= 0
  ensures t == N * (N + 1) / 2
{
  t := 0;
  var n := 0;
  while n < N
    invariant 0 <= n <= N
    invariant t == n * (n + 1) / 2
  {
    n, t := n + 1, t + n + 1;
  }
}
```



Based on Fig. 1, Developing Verified Programs with Dafny by Rustan Leino

Loop Invariants

```
method TriangleNumber(N: int) returns (t: int)
   requires N >= 0
   ensures t == N * (N + 1) / 2
{
   t := 0;
   var n := 0;
   while n < N
      invariant 0 <= n <= N
      invariant t == n * (n + 1) / 2
   {
      n, t := n + 1, t + n + 1;
   }
}
```

Would < work instead of <= ?

Loop Termination

```
method TriangleNumber(N: int) returns (t: int)
  requires N >= 0
  ensures t == N * (N + 1) / 2
{
  t := 0;
  var n := 0;
  while n < N
    invariant 0 <= n <= N
    invariant t == n * (n + 1) / 2
    decreases N – n                  // can be left out because it is guessed correctly by Dafny
  {
    n, t := n + 1, t + n + 1;
  }
}
```

Factorial: specification

```
function factorial(n: nat): nat {
    if n == 0 then 1 else n * factorial(n - 1)
}
```

Factorial: specification + implementation

```
method ComputeFactorial(n: nat) returns (r: nat)
    ensures r == factorial(n)
{
    var i := 1;
    r := 1;
    while i < n
        invariant 1 <= i <= n
        invariant r == factorial(i)
    {
        i := i + 1;
        r := r * i;
    }
}
```

Factorial: alternative

```
function method factorial(n: nat): nat
    decreases n                    // not needed – Dafny guesses this correctly
{
    if n == 0 then 1 else n * factorial(n - 1)
}

method ComputeFactorial(n: nat) returns (r: nat)
    ensures r == factorial(n)
{
    r := factorial(n);
}
```

Lemma: *ghost* method

```
method ComputePow2(n: nat) returns (p: nat)
  ensures p == pow2(n)
{
  if n == 0 { p := 1; }
  else if n % 2 == 0 {
    p := ComputePow2(n / 2);
    Pow2lemma(n);
    p := p * p;
  } else {
    p := ComputePow2(n - 1);
    p := 2 * p;
  }
}
```

```
function pow2(n: int): int
  requires 0 <= n
{
  if n == 0 then 1 else 2 * pow2(n-1)
}

lemma Pow2lemma(n: nat)
  requires n % 2 == 0
  ensures pow2(n) == pow2(n/2) * pow2(n/2)
{
  if n != 0 { Pow2lemma(n - 2); }
}
```

Based on Fig. 5, Developing Verified Programs with Dafny by Rustan Leino

Datatypes and Pattern Matching

```
datatype Tree<T> = Leaf | Node(Tree, T, Tree)

function Contains <T>(t: Tree<T>, v: T): bool
{
	match t
	case Leaf => false
	case Node(left, x, right) =>
		x == v || Contains(left, v) || Contains(right, v)
}
```

Based on Fig. 3, Developing Verified Programs with Dafny by Rustan Leino

Arrays

```
method FindZero(a: array<nat>) returns (index: int)
  ensures index < 0  ==> forall i :: 0 <= i < a.Length ==> a[i] != 0
  ensures 0 <= index ==> index < a.Length && a[index] == 0
{
  index := 0;
  while index < a.Length
    invariant forall k :: 0 <= k < index && k < a.Length ==> a[k] != 0
  {
    if a[index] == 0 { return; }
    index := index + 1;
  }
  index := -1;
}
```

Array: next element at most 1 lower

```
method FindZero(a: array<nat>) returns (index: int)
  requires forall i :: 0 < i < a.Length ==> a[i-1] <= a[i] + 1
  ensures index < 0  ==> forall i :: 0 <= i < a.Length ==> a[i] != 0
  ensures 0 <= index ==> index < a.Length && a[index] == 0
{
  index := 0;
  while index < a.Length
    invariant forall k :: 0 <= k < index && k < a.Length ==> a[k] != 0
  {
    if a[index] == 0 { return; }
    index := index + 1;
  }
  index := -1;
}
```

Array: next element at most 1 lower

```
method FindZero(a: array<nat>) returns (index: int)
  requires forall i :: 0 < i < a.Length ==> a[i-1] <= a[i] + 1
  ensures index < 0  ==> forall i :: 0 <= i < a.Length ==> a[i] != 0
  ensures 0 <= index ==> index < a.Length && a[index] == 0
{
  index := 0;
  while index < a.Length
    invariant forall k :: 0 <= k < index && k < a.Length ==> a[k] != 0
  {
    if a[index] == 0 { return; }
    index := index + a[index];
  }
  index := -1;
}
```

Based on https://rise4fun.com/Dafny/tutorial/Lemmas

Array: next element at most 1 lower

```
method FindZero(a: array<nat>) returns (index: int)
  requires forall i :: 0 < i < a.Length ==> a[i - 1] <= a[i] + 1
  ensures index < 0  ==> forall i :: 0 <= i < a.Length ==> a[i] != 0
  ensures 0 <= index ==> index < a.Length && a[index] == 0
{
  index := 0;
  while index < a.Length
    invariant forall k :: 0 <= k < index && k < a.Length ==> a[k] != 0
  {
    if a[index] == 0 { return; }
    SkippingLemma(a, index);
    index := index + a[index];
  }
  index := -1;
}
```

Based on https://rise4fun.com/Dafny/tutorial/Lemmas

Lemma example

```
lemma SkippingLemma(a : array<nat>, j : nat)
  requires forall i :: j < i < a.Length ==> a[i - 1] <= a[i] + 1
  requires j < a.Length
  ensures forall k :: j <= k < j + a[j] && k < a.Length ==> a[k] != 0
{
  var i := j;
  while i < j + a[j] && i < a.Length
    invariant i < a.Length ==> a[j] - (i - j) <= a[i]
    invariant forall k :: j <= k < i && k < a.Length ==> a[k] != 0
  {
    i := i + 1;
  }
}
```

Based on https://rise4fun.com/Dafny/tutorial/Lemmas

Alternative lemma (proof by induction)

```
lemma SkippingLemma(a : array<nat>, j : nat)
  requires forall i :: j < i < a.Length ==> a[i-1] <= a[i] + 1
  requires j < a.Length
  ensures forall k :: j <= k < j + a[j] && k < a.Length ==> a[k] != 0
  decreases a.Length - j
{
      if j < a.Length - 1 {
            SkippingLemma(a, j + 1);
      }
}
```

Example: proof by contradiction

```
lemma singleton<T>(s: set<T>, e: T)      // if s is a singleton set and e is in s then s == { e }
   requires |s| == 1
   requires e in s
   ensures s == {e}
{
   if s != {e} {
         assert |s - {e}| == 0;
         assert s == {e};          // don't need this --- Dafny figured that out already
         assert false;             // ditto
   }
}
```

Framing: shared memory is hard…

```
method copy<T>(src: array<T>, dst: array<T>)
    requires src.Length == dst.Length
    ensures forall i :: 0 <= i < src.Length ==> src[i] == dst[i]
    modifies dst
{
    var k := 0;
    while k < src.Length
        invariant forall i :: 0 <= i < k && i < src.Length ==> src[i] == dst[i]
    {
        dst[k] := src[k];
        k := k + 1;
    }
}
```

Class example: Queue

```
method Main()
{
    var q := new Queue();
    q.Enqueue(5);
    q.Enqueue(12);
    var x := q.Dequeue();
    assert x == 5;
}
```

Based on Fig. 4, Developing Verified Programs with Dafny by Rustan Leino

Class Queue

```
class {:autocontracts} Queue {
  ghost var Contents: seq<int>;
  var a: array<int>;
  var hd: int, tl: int;

  predicate Valid() {              // class invariant
    a.Length > 0 && 0 <= tl <= hd <= a.Length && Contents == a[tl..hd]
  }

  constructor ()   ensures Contents == []
  {
    a, tl, hd, Contents := new int[10], 0, 0, [];
  }
}
```

Class Queue: continued

```
method Enqueue(d: int)     ensures Contents == old(Contents) + [d]    {
    if hd == a.Length {
      var b := a;
      if tl == 0 { b := new int[2 * a.Length]; }                    // a is full
      forall (i | 0 <= i < hd – tl)  { b[i] := a[tl + i]; }         // shift
      a, tl, hd := b, 0, hd – tl;
    }
    a[hd], hd, Contents := d, hd + 1, Contents + [d];
  }

  method Dequeue() returns (d: int)
    requires Contents != []
    ensures d == old(Contents)[0] && Contents == old(Contents)[1..];
  {
    d, tl, Contents := a[tl], tl + 1, Contents[1..];
  }
```

Try all this out yourself

- Start online: http://rise4fun.com/Dafny/tutorial
- Install mono and Dafny on your laptop
- Second assignment:
  - Specify and implement two sorting functions in Dafny:
    1. A “functional” version that takes a sequence as input and produces one as output
    2. An “imperative” version sorting an array in place
    3. Ideally use two different sorting methods for this
       - Quicksort, mergesort, bubblesort, …

Dafny resources

- Tutorial: http://rise4fun.com/Dafny/tutorial
- Another (pdf): https://arxiv.org/pdf/1701.04481.pdf
- Reference manual: https://homepage.cs.uiowa.edu/~tinelli/classes/181/Papers/dafny-reference.pdf
- Good quick overview: https://homepage.cs.uiowa.edu/~tinelli/classes/181/Fall15/Papers/Lein13.pdf