Autograding Weakest Precondition Proofs and Dafny Specifications

Graeme Smith and Hunter Whitlock

The University of Queensland, Australia

CSSE3100/7100 Reasoning about Programs

- Introduction (1 week)
- Weakest precondition proofs (3 weeks)
- Dafny methods (4 weeks)
  - pre/postconditions
  - loop invariants / variants
- Dafny classes (3 weeks)
  - data abstraction
- Other topics (1 week)

3rd-year undergraduate and Masters

Number of students

2018	73
⋮	
2025	217

CSSE3100/7100 Reasoning about Programs

- Introduction (1 week)
- Weakest precondition proofs (3 weeks) ← Assignment 1
- Dafny methods (4 weeks) ← Assignment 2
  - pre/postconditions
  - loop invariants / variants
- Dafny classes (3 weeks) ← Assignment 3
  - data abstraction
- Other topics (1 week)

Requirements for the autograder

1. Allow fine-grained marking
   – part marks for partially correct solutions

2. Not artificially constrain students' use of programming techniques *
   – allow them to apply best programming practices

3. Be reusable for new versions of assignments
   – with minimal input from teaching staff

* Unless there is a pedagogical reason

Proofs in Dafny

```
method M(s: nat, t: nat) returns (r: nat)
    requires s > 0 && t > 1
    ensures r < s
{
    r := s/t;
}
```

postcondition could not be proved

Proofs in Dafny

```
method M(s: nat, t: nat) returns (r: nat)
    requires s > 0 && t > 1
    ensures r < s
{
    { s/t < s }
    r := s/t;
    { r < s }
}
```

postcondition could not be proved

Proofs in Dafny

```
method M(s: nat, t: nat) returns (r: nat)
    requires s > 0 && t > 1
    ensures r < s
{
    { s/t < s }
    r := s/t;
    { r < s }
}
```

postcondition could not be proved

```
method M(s: nat, t: nat) returns (r: nat)
    requires s > 0 && t > 1
    ensures r < s
{
    calc {
            s/t < s;
        == (s/s)/t < s/s;
        == 1/t < 1;
    }
    r := s/t;
}
```

Assignment 1 – weakest precondition proofs

```
…
while n != 33
        invariant s == n*(n-1)/2
{
        { s == n*(n-1)/2 && n != 33 }          (strengthening)
        { s == n*(n-1)/2 }                     (arithmetic)
        { s + n == n*(n-1)/2 + n }
        s := s + n;
        { s == n*(n-1)/2 + n }
        …
}
…
```

Assignment 1 – weakest precondition proofs

```
…
while n != 33
        invariant s == n*(n-1)/2
{
        ghost var WP: bool, WP_s: bool;
        WP_s := s == n*(n-1)/2 && n != 33;    // strengthening
        WP := s == n*(n-1)/2;                 // arithmetic
        WP := s + n == n*(n-1)/2 + n;
        s := s + n;
        WP := s == n*(n-1)/2 + n;
        …
}
…
```

Assignment 1 – Checker file

```
... // supporting functions

class Checker {
    var ν  // input, output and local variables

    ... // checking method goes here
}
```

Equivalence

```
WP := ϕ;
WP := ψ;
```

Also, while and if-else conditions, e.g.,

```
WP := ϕ;
if B {WP := P; ...} else {WP := Q; ...}

calc {ϕ; == ((B ==> P) && (!B ==> Q));}
```

```
method Check() {
        assume α;    // α is teaching-staff provided assumptions
        calc { ϕ; == ψ; }
}
```

Assignment 1 – Checker file

```
... // supporting functions

class Checker {
    var ν  // input, output and local variables

    ... // checking method goes here
}
```

Strengthening WP_s := ϕ;
WP := ψ;

```
    method Check() {
        assume α;
        calc { φ; ==> ψ; }
    }
```

```
method Check() {
    assume α;
    calc { ψ; ==> φ; }          // should fail
}
```

Assignment 1 – Checker file

WP := ϕ;
x := E;
WP := ψ;

```
method Check()
        modifies this
{
        assume α;
        x := E;
        assert ψ <==> old(φ);
}
```

Proof:

{$\psi[x\backslash E]$ <==> ϕ}	(at start of method)
{$\psi[x\backslash E]$ <==> old(ϕ)}	(old values are constant)
x := E;	
{ψ <==> old(ϕ)}	(wp(assert P, Q) = P && Q)
assert ψ <==> old(ϕ);	
{true}	(no explicit postcondition)

Assignment 2 – Specifying and implementing Dafny methods

Specification compared against teaching-staff provided solution

i. Full student pre or postcondition must imply each conjunct of the solution (not too weak)

ii. The full solution implies each of the conjuncts of the student pre or postcondition (not too strong)

iii. The full student pre or postcondition is not false!

Assignment 2 – Checking preconditions

```
... // supporting functions (from assignment)
... // additional functions (from student)

class Checker {
    ... // checking methods go here
}
```

Preconditions

```
method Check(v_in) {
    assume α;
    calc { Ψ; ==> false; }
}     // should fail
```

```
method Check(v_in) {
    assume {:axiom} α;
    calc { Ψ; ==> φ_i ; }
}
```

```
method Check(v_in) {
    assume {:axiom} α;
    calc { Φ; ==> ψ_i ; }
}
```

Ψ is student precondition, ψ_i is its i-th conjunct
Φ is solution precondition, ϕ_i is its i-th conjunct

Assignment 2 – Checking postconditions

Postconditions

```
method Change(v_in)
    modifies δ

method Check(ν)
    modifies δ
{
    assume pre;
    Change(μ_in);
    assume α;
    calc { Ψ; ==> false; }
}     // should fail
```

```
method Change(v_in)
    modifies δ

method Check(ν)
    modifies δ
{
    assume {:axiom} pre;
    Change(μ_in);
    assume {:axiom} α;
    calc { Ψ; ==> φ_i; }
}
```

```
method Change(v_in)
    modifies δ

method Check(ν)
    modifies δ
{
    assume {:axiom} pre;
    Change(μ_in);
    assume {:axiom} α;
    calc { Φ; ==> ψ_i ; }
}
```

Ψ is student postcondition, ψ_i is its i-th conjunct
Φ is solution postcondition, ϕ_i is its i-th conjunct

Assignment 2 – Checking invariants

- See if the student's code verifies for each correct conjunct of the student's postcondition
  - verification succeeds if they have an invariant which ensures that postcondition

// additional methods and lemmas (from student)

method Check(v_{in}) returns (v_{out})
    requires pre
    modifies δ
    ensures ψ_{ensi}
{
    $code$;
}

No assumptions included, student's must ensure their code verifies

Evaluation

1. By how much is manual marking time reduced?

Year	Assignment 1	Assignment 2
2024	20.08	21.92
2025	9.87 (50%)	5.05 (77%)

Mean manual marking time (minutes per submission)

2. By how much is marking accuracy increased?

Year	Assignment 1	Assignment 2
2024	22%	44%
2025	11%	29%

Number of re-mark requests resulting in changes to marks

Conclusion

Dafny autograder for

- fine-grained marking of wp proofs, pre/post specifications and invariants
- allow maximum flexibility in writing code
- easy to reuse for new assignments

Extended for a third assignment using Dafny classes for data structures

Successfully trialled on ~380 students in 2025 and 2026

Future directions: tutoring system for feedback to short-answer questions