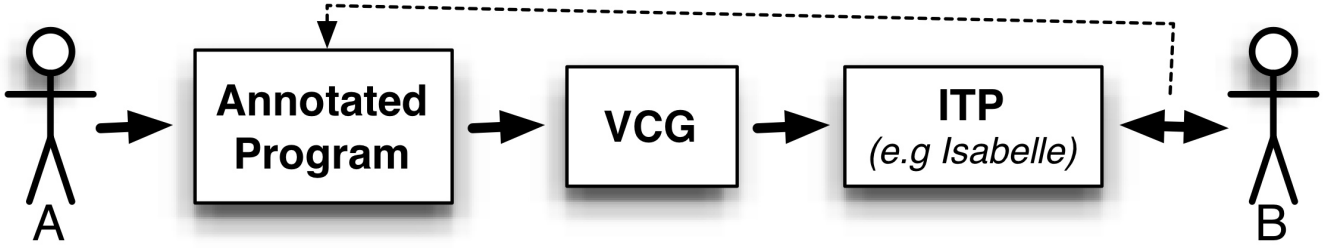
A new project
Tactics for the Dafny Program Verifier

Gudmund Grov

Induction workshop, Chalmers, March 2015

A wee bit of history...

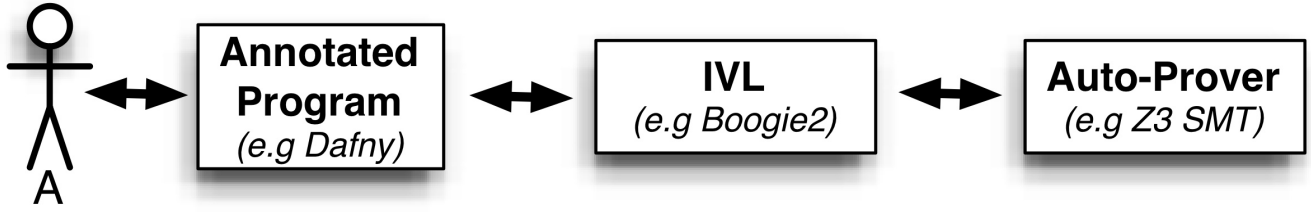
- Traditionally, program verifiers used Interactive Theorem Provers (ITPs)



- Requires programming/specification + ITP expertise
- Automation via proof **tactics**
  - General proof patterns
  - User can encode (problem specific) proof patterns
- Relate ITP failure with program text
  - e.g. strengthen loop invariant

New wave of program verifiers

- New wave of program verifiers
  - Dafny, VCC, Verifast, Spark 2014, ...
- Highly automated theorem provers/SMT
- All user interaction via program text



- Crucially, **proof guidance** in program text
  - auto-active
  - user does not need to learn additional system (ITP)

A challenge

- Proof automation in hands of underlying prover
  - unless user willing to change VC/IVL generation...
- Repetitive "proof steps" has to be provided each time
- No notion of user-defined tactics as with ITPs

Hypothesis

Can we develop a notion of "tactics" for program verifiers?

- User encodes proof patterns in the program text
- .... using familiar programming constructs
  - avoids having to learn new languages/systems/...
- .... and applies them to similar problems

Hypothesis

Can we develop a notion of "tactics" for program verifiers?

- User encodes proof patterns in the program text
- .... using familiar programming constructs
  - avoids having to learn new languages/systems/...
- .... and applies them to similar problems
- Liberate user from
  - tedious repetitive tasks
  - low-level search tasks
- ... enables them to focus on key proof steps

ITP tactics

- Proof steps transform *proof state*
  - keeps track of open sub-goals
- **ITP tactic** = proof state ⇒ proof state

ITP tactics

- Proof steps transform *proof state*
  - keeps track of open sub-goals
- **ITP tactic** = proof state ⇒ proof state
- Consider conj-I tactic

$$\frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$$

- applied to sub-goal 1 of *proof state*

```
Sub-goal 1: x ∧ z
Sub-goal 2: d
```

ITP tactics

- Proof steps transform *proof state*
  - keeps track of open sub-goals
- **ITP tactic** = proof state ⇒ proof state
- Consider conj-I tactic

$$\frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$$

- applied to sub-goal 1 of *proof state*

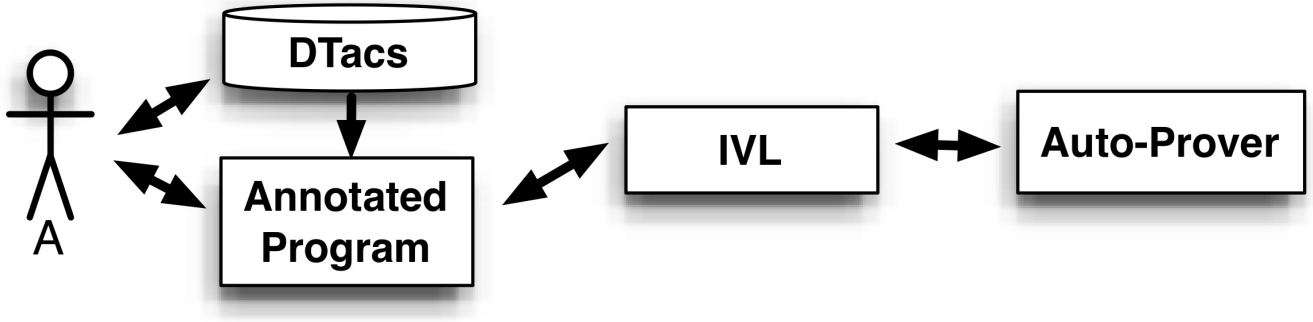
```
Sub-goal 1: x ∧ z
Sub-goal 2: d
```

- should generate *proof state*:

```
Sub-goal 1: x
Sub-goal 2: z
Sub-goal 3: d
```

Towards Dafny tactics

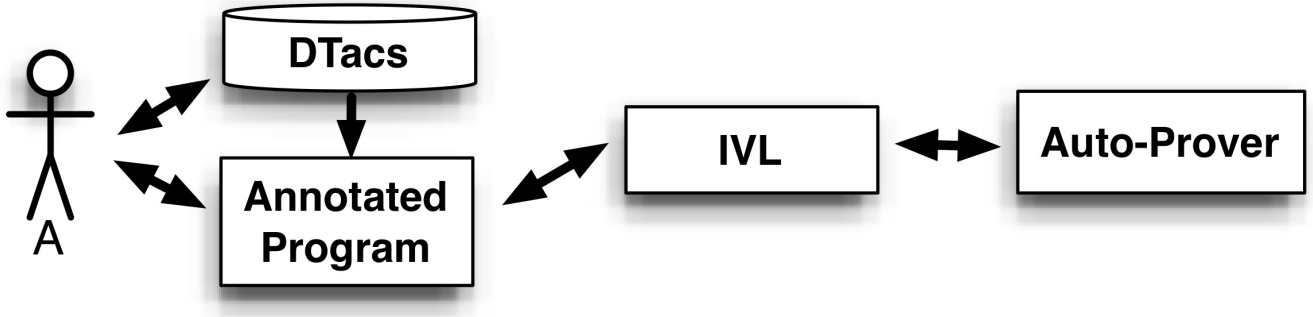
- Use Dafny tactics (DTacs) to support proof process



- Proof state $\sim$ program text

Towards Dafny tactics

- Use Dafny tactics (DTacs) to support proof process



- Proof state $\sim$ program text
- Proof steps transform *program text*
- **Dafny tactic** = program text $\Rightarrow$ program text
- ... or a form of **program refactoring**:
  Correct Dafny program $\Longrightarrow$ **provably** correct Dafny programs

Towards Dafny tactics

- An analogous Dafny tactic to

$$\frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$$

- would split a lemma

```
lemma BigGoal() ensures A ∧ B {..}
```

Towards Dafny tactics

- An analogous Dafny tactic to

$$\frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$$

- would split a lemma

```
lemma BigGoal() ensures A ∧ B {..}
```

- .. into two lemmas:

```
lemma SubGoalA() ensures A {..}
lemma SubGoalB() ensures B {..}
lemma BigGoal() ensures A ∧ B
{ SubGoalA(); SubGoalB();}
```

Example Dafny tactics

- Proof by cases
- Apply lemma
- Introduce Skolem function/constants
- Add assertion

Proof by cases example

```
lemma Length(n : int)
   requires n ≥ 0;
   ensures ∃ xs • length(xs) = n;
{
}
```

Proof by cases example

```
lemma Length(n : int)
   requires n ≥ 0;
   ensures ∃ xs • length(xs) = n;
{
}
```

$$\xRightarrow{case-I}$$

```
lemma Length(n : int) ..{
  if(n = 0){
  }else{
}
```

Apply lemma (induction hypothesis)

```
lemma Length(n : int)..{
  if(n = 0){
  }else{
}
```

Apply lemma (induction hypothesis)

```
lemma Length(n : int) ..{
  if(n = 0){
  }else{
}
```

$$\xRightarrow{IH-I}$$

```
lemma Length(n : int) ..{
  if(n = 0){
  }else{
        Length(n-1);
}
```

Introduce Skolem constant

```
lemma Length(n : int) ...
  ... ensures ∃ xs • length(xs) = n;{
  if(n = 0){
  }else{
        Length(n-1);
}
```

Introduce Skolem constant

```
lemma Length(n : int) ...
  ... ensures ∃ xs • length(xs) = n;{
  if(n = 0){
  }else{
        Length(n-1);
}
```

$$\xRightarrow{ex-E}$$

```
lemma Length(n : int) ..{
  if(n = 0){
  }else{
        Length(n-1);
        var xs :| length(xs) = n-1;
}
```

Add assertion

```
lemma Length(n : int)..{
  if(n = 0){
  }else{
        Length(n-1);
        var xs :| length(xs) = n-1;
}
```

Add assertion

```
lemma Length(n : int) ..{
  if(n = 0){
  }else{
        Length(n-1);
        var xs :| length(xs) = n-1;
}
```

$$\xRightarrow{asset-I}$$

```
lemma Length(n : int) ..{
  if(n = 0){
  }else{
        Length(n-1);
        var xs :| length(xs) = n-1;
        assert length(Cons(1,xs)) = n;
}
```

Complete proof

```
lemma Length(n : int)
   requires n ≥ 0;
   ensures ∃ xs • length(xs) = n;
{
  if(n = 0){
  }else{
        Length(n−1);
        var xs :| length(xs) = n−1;
        assert length(Cons(1,xs)) = n;
}
```

Some more DTacs...

- Generate lemma
- Add/strengthen loop invariant
- Add/strengthen post-condition
- Add/strengthen pre-condition
- Add ghost constructs
  - including body (proof) of lemma
- Trigger/instantiate quantifier

Atomics: towards a trusted "kernel"

- We cannot arbitrarily apply tactics
  - E.g. strengthen pre-conditions changes contract
  - However, there are cases where we can
    - e.g. on lemmas generated and applied
- Develop a "kernel" of trusted **atomic** Dafny tactics
- ... with suitable pre-conditions
- Comparable to ITP kernels
- All other tactics: composition of the atomics

Tactic language

- Need tactic language to combine
  - atomic tactics
  - previously defined tactics
- This is really a programming language
  - loops, alternation, branching,...
- We already have a fully fledged programming language in Dafny
  - use (variant of) Dafny...
  - try to keep language "natural" for Dafny users

A Dafny Tactics

```
tactic length_tac()
{
    var f,n := meth_name(),meth_arg();
    if(n = 0){
    }else{
            call(f(n-1));
            ...
    }
}
```

Workplan

- Re-engineer existing Dafny developments + develop new
  - Extract proof strategies
  - Identify "kernel" of "atomic tactics"
- Define Dafny subset as tactic languge to combine them
- Develop "refactoring" framework to apply tactics
- Implement proof patterns and evaluate

Challenges

- Each tactic **refactor** program text
  - Will generate new Dafny programs
  - Where to add generated program text?
  - How to define progress?
  - Speed/response rate – close unnecessary branches
- Right level of abstraction
  - tactic as part of proof process instead?
  - close integration with underlying prover?
- How to define tactic for e.g. loop invariant
  - use of theory schemas? or just arguments?
- ...

Post-doc position available

```
method post_doc(p : Person)
  requires p has PhD in relevant area
  requires August ≤ p starts ≤ October
  ensures 12 months position for p
  ensures Dafny tactics as Tacny system
{
   // to be completed by p
}
```

https://sites.google.com/site/avocs15/

AVOCS'15

1 - 4 Sept, Edinburgh, UK

15th International Workshop on Automated Verification of Critical Systems

Home | Call for Papers | Important Dates | Organisation | Submission | Registration | Student Grants | Programme | Accommodation | Local Information

The 15th International Workshop on Automated Verification of Critical Systems (AVoCS 2015) will take place 1-4 September 2015 in Edinburgh, UK. The workshop is organised by the Computer Science department within the School of Mathematical and Computer Sciences at Heriot-Watt University and will be held in the ICMS building found in the historic old town of the Edinburgh, and will be collocated with the 6th AI4FM workshop, which will be held on 1 September.

Highlights

- **Full paper deadline 12th June (abstract due 5th June)**
- **Special research ideas session: short papers due 7th August**
- **Invited talks by**
  - **Colin O'Halloran (D-RisQ/Oxford)**
  - **Don Sannella (Contemplate/Edinburgh)**
- **Proceedings to be published by EASST**
- **Special issues of Science of Computer Programming**

Paper deadline: 12th June