

# Trigger Selection Strategies to Stabilize Program Verifiers

K. Rustan M. Leino (MSR), Clément Pit-Claudel (MIT CSAIL)

CAV 2016 – July 21, 2016





# What is Dafny?

- Dafny is a **verification-aware programming language**.
- Like many other tools, Dafny is based on **Boogie and Z3**.
- It runs on most platforms, and has advanced editor support in Visual Studio and (now!) in Emacs.

Dafny hands-on: finding the maximum of a sequence (solution)

# What problem are we trying to solve?

- Dafny is very snappy on small programs
- On larger programs it suffers from **butterfly effects**:
  - Verification performance is **chaotic** (unstable and unpredictable)
  - Insignificant changes cause verification failures

---

```
var x := y;
assert ...;
```

---

✓ Verifies :)

---

```
var x := y + 0;
assert ...;
```

---

✗ Fails to verify ?!

This work focuses on this issue in Dafny, but we expect our techniques to apply **to other verifiers**.

# What causes instability?

## ■ Translation

- Similar Dafny programs can look very different at the Z3 level

## ■ Undecidable/Semi-decidable domains:

- Non-linear arithmetic
- **First-order logic (quantifier instantiations) ← Our focus**
  - Costly instantiations
  - Matching loops

Solution: pick **better triggers**

# The Dafny pipeline

- 0 Parse
- 1 Type-check
- 2 Transform the AST ← this project happens here
- 3 Translate to Boogie
- 4 Translate to Z3
- 5 Verify

## How does Z3 handle quantifiers?

- Z3 relies on **triggers** (matching patterns) to instantiate quantifiers. Every time Z3 comes across a new term, it instantiates all quantifiers whose triggers match the new term. For example:

---

IsHuman(Socrates)

$\forall h \{ \text{IsMortal}(h) \} \cdot \text{IsHuman}(h) \implies \text{IsMortal}(h)$

Goal: IsMortal(Socrates)

---

- Bad trigger choices cause **verification failures**, **matching loops**, and **costly instantiations**.
- Z3 knows how to pick good triggers for **clean formulas**.

# Why isn't this enough?

Z3 produces **excessively liberal triggers** on Dafny programs.

- Dafny produces large formulas with many **parasitic terms**, due to its internal encoding.
  - Dafny: `s[x]`
  - Boogie: `$Unbox(read($Heap, s#0, IndexField(x#1)))`
  - Z3: `(U_2_int ($Unbox intType (MapType1Select $Heap1 |s#00| ...)))`
- Debugging and understanding trigger choices is hard (triggers are Z3 terms, not Dafny terms!).

We need to **pick triggers at the Dafny level**.

# How do we generate good triggers?

- 0 Walk the AST below a quantifier. Annotate each term as
  - A trigger head, if it can act as a trigger:  
 $f(x) \quad \text{old}(h(x, y)) \quad x \text{ in multiset}\{\dots\}$
  - A trigger killer, if it prevents parent nodes from being heads:  
 $x+1 \quad \neg y \quad x \text{ in multiset}\{\dots\}$
- 1 Collect all trigger heads
- 2 Enumerate subsets to generate candidates
- 3 Filter



# Trigger generation example

Quantifier:  $\forall x \cdot P(x) \wedge (Q(x) \implies P(x+1))$

Subexpressions:  $x \ P(x) \ Q(x) \ 1 \ x+1 \ P(x+1) \ (Q(x) \implies P(x+1)) \ \dots$

Killers:  $x+1 \ P(x+1) \ (Q(x) \implies P(x+1)) \ \dots$

Heads:  $P(x) \ Q(x)$

# What do we gain?

- Triggers now come from actual **Dafny terms**: we can show them to the user directly
- **Parasitic** terms are not chosen as triggers anymore: **less costly instantiations**
- We can show warnings when we can't find good triggers
- And we can start **looking for matching loops!**

# What are matching loops?

- Matching loops occur when instantiating a quantifier produces terms that directly or indirectly **cause it to be instantiated again**, repeatedly:

---


$$\forall x \{f(x)\} \cdot f(x) \leq f(f(x))$$


---

$$f(\theta) \rightsquigarrow f(f(\theta)) \rightsquigarrow f(f(f(\theta))) \rightsquigarrow f(f(f(f(\theta)))) \rightsquigarrow f(f(f(f(f(\theta))))) \rightsquigarrow \dots$$


---

$$\forall x \{P(x)\} \cdot P(x) \wedge (Q(x) \implies P(x+1))$$


---

$$P(x) \rightsquigarrow P(x+1) \rightsquigarrow P(x+2) \rightsquigarrow P(x+3) \rightsquigarrow P(x+4) \rightsquigarrow \dots$$

# Detecting and suppressing matching loops

- 0 For every candidate trigger, compute the set of **matching terms** in the body of the quantifier.
- 1 For each matching term, decide whether it **might create a loop**:
  - $\{f(x)\} \approx f(x)$ ? **Safe**
  - $\{f(x)\} \approx f(x+1)$ ? **Loops**
  - $\{f(x)\} \approx f(f(x))$ ? **Loops**
  - $\{f(x, y)\} \approx f(y, x)$ ? **Safe**
- 2 **Suppress triggers** that could lead to matching loops
- 3 **Report information** to the user

# Overly enthusiastic loop suppression causes a loss of expressive power

- Cycle detection acts on a **full quantifier**, while loops often only involve **parts of it**:

---


$$\forall x \{??\} \cdot P(x) \wedge (Q(x) \implies P(x+1))$$


---

- Suppressing loops **costs us too much** expressiveness: we don't learn  $P(x)$  anymore!

# Splitting quantifiers regains some expressiveness

We extended Dafny to **split quantifiers** before checking for loops:

$$\frac{\forall x \{Q(x)\} \cdot P(x) \wedge (Q(x) \implies P(x+1))}{\implies} \frac{\forall x \{P(x)\} \cdot P(x) \quad \forall x \{Q(x)\} \cdot Q(x) \implies P(x+1)}{\implies}$$

- Each quantifier gets its own triggers.
- This fixes some of our issues, but we **lose a different type of expressiveness**: learning  $Q(x)$  doesn't teach us  $P(x)$  anymore!

# Triggers sharing further recovers expressive power

- Triggers do not need to appear **in the body** of a quantifier.
- Dafny can **share triggers** across all terms of a split quantifier:

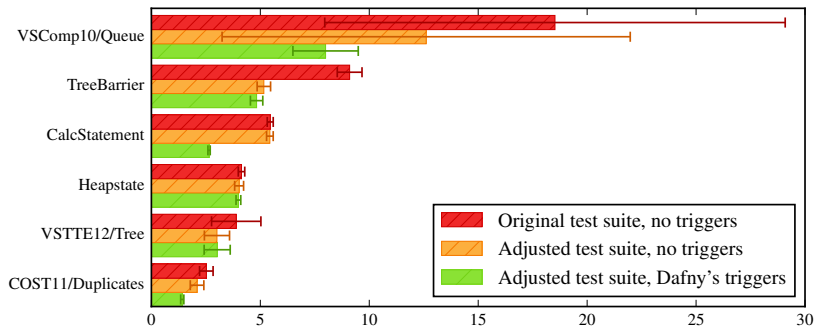
---


$$\begin{array}{l} \forall x \{P(x)\} \{Q(x)\} \cdot P(x) \\ \forall x \qquad \qquad \{Q(x)\} \cdot Q(x) \implies P(x+1) \end{array}$$


---

# Variability and performance on Dafny's test suite

The effect on most tests is small, but some tests do benefit significantly.

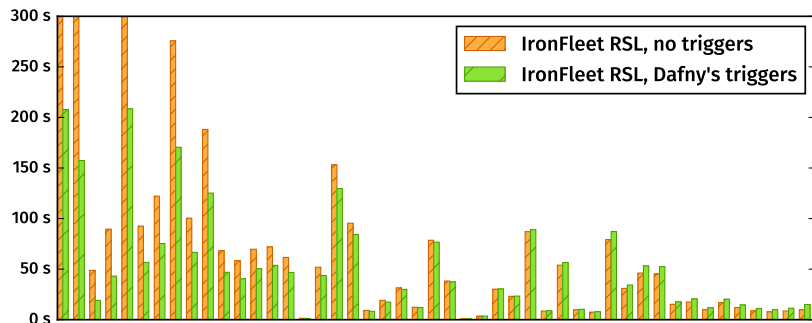


Verification times (seconds) for six of the affected test programs.



In the real world

## IronFleet RSL



Verification times in seconds for the 48 programs composing the implementation layer of IronRSL. Overall: 30% faster.

# Usability results

Dafny now **picks triggers** and reports them directly in the editor.

```
method Main() {
  assume  $\forall x \cdot P(x) \wedge (Q(x) \implies P(x+1))$ ;
}
```

For expression "Q(x) ==> P(x + 1)":  
 Selected triggers: {Q(x)}  
 Rejected triggers: {P(x)} (may loop with "P(x + 1)")

For expression "P(x)":  
 Selected triggers:  
 {Q(x)}, {P(x)}

# Conclusion

Trigger generation, quantifier splitting, and matching loop elimination offer new, exciting opportunities to improve the **performance** and **predictability** of tools based on SMT solvers.

# Thanks!

- Check out the **updated Dafny** (on GitHub! MIT-licensed!):

<https://github.com/Microsoft/dafny>

Emacs mode: <https://github.com/boogie-org/boogie-friends/>

- Implement this and try it in your own solver

(and let us know how well it works!)

- Talk to me!

[clement@pit-claudel.fr](mailto:clement@pit-claudel.fr)

<http://pit-claudel.fr/clement/>