UNIVERSITY OF MARYLAND

SAT / SMT Solving

Tautology-proving in Dafny

- Dafny proves tautologies when verifying code
  - Needs to prove that method preconditions imply the weakest precondition of method postconditionsfollowing statements
- Uses “SMT” (= “Satisfaction Modulo Theories”) solvers
- We will see how SMT solvers work….

Refresher: Weakest Preconditions

- Weakest preconditions start from code S and postcondition Q!
  - If Q is a postcondition and S is code, then P is the *weakest precondition for S and Q* if and only if:
  - $\{P\}\ S\ \{Q\}$ is valid
  - P is the "most general" among all preconditions P' such that $\{P'\}\ S\ \{Q\}$ is valid

    "Most general" means that for all P' such that $\{P'\}\ S\ \{Q\}$ is valid, $P' \Rightarrow P$
- Some facts
  - For traditional imperative languages: weakest preconditions always exist!
    - Regardless of form of S and Q, weakest precondition can be written down as a formula
    - Notation: $wp(S, Q)$ used for weakest precondition of S, Q
  - $wp(S, Q)$ can (often) be computed syntactically!

Computing $wp(S, Q)$: Assignment

- Suppose S is `x := t`. What is $wp(S, Q)$?
  - $\boldsymbol{wp(S, Q) = Q[x \coloneqq t]}$
- Example:

$$\{\,?\,\}$$

```
x := x + 1;
```

$$\{x = 42\}$$

Computing $wp(S, Q)$: Assignment

- Suppose S is `x := ` t. What is $wp(S, Q)$?
  - $\boldsymbol{wp(S, Q) = Q[x \coloneqq t]}$
- Example:

$$\{\mathrm{x} + 1 = 42\}$$

```
x := x + 1;
```

$$\{x = 42\}$$

Computing $wp(S, Q)$: Assignment

- Suppose S is `x := t`. What is $wp(S, Q)$?
  - $\boldsymbol{wp(S, Q) = Q[x \coloneqq t]}$
- Example:

$$\{\,?\,\}$$

```
x := y * y;
```

$$\{x \geq 0 \;\&\&\; \mathrm{y} = \mathrm{z}\}$$

Computing $wp(S, Q)$: Assignment

- Suppose S is `x := t`. What is $wp(S, Q)$?
  - $\boldsymbol{wp(S, Q) = Q[x \coloneqq t]}$
- Example:

$$\{\mathrm{y} * \mathrm{y} \geq 0 \;\&\&\; \mathrm{y} = \mathrm{z}\}$$

```
x := y * y;
```

$$\{x \geq 0 \;\&\&\; \mathrm{y} = \mathrm{z}\}$$

Computing wp(S, Q): Statement Blocks

```
assert P;
s1; s2;
assert Q;
```

What is the weakest P?

Computing wp(S, Q): Statement Blocks

$\{\ ?\ \}$

```
x := y * y;
x := x + 1;
```

$\{x \geq 0 \ \&\& \ \mathrm{y} = \mathrm{z}\}$

Computing wp(S, Q): Statement Blocks

$$\{\,?\,\}$$

```
x := y * y;
```

$$\{x + 1 \geq 0 \;\&\&\; \mathrm{y} = \mathrm{z}\}$$

```
x := x + 1;
```

$$\{x \geq 0 \;\&\&\; \mathrm{y} = \mathrm{z}\}$$

Computing wp(S, Q): Statement Blocks

$\{y * y + 1 \geq 0 \; \&\& \; \mathrm{y} = \mathrm{z}\}$

```
x := y * y;
```

$\{x + 1 \geq 0 \; \&\& \; \mathrm{y} = \mathrm{z}\}$

```
x := x + 1;
```

$\{x \geq 0 \; \&\& \; \mathrm{y} = \mathrm{z}\}$

Computing $wp(S, Q)$: Statement Blocks

- Suppose S is $S_1; S_2; \cdots S_n;$
- $wp(S, Q)$ is computed starting at the end of the block and working forward

  $wp(P, S) = P_1$, where:

$$\begin{aligned} P_n &= wp(S_n, Q) \\ P_{n-1} &= wp(S_{n-1}, P_n) \\ &\vdots \\ P_1 &= sp(S_1, P_{n-1}) \end{aligned}$$

Computing $wp(P, S)$: if-then-else

```
assert P;
if b {
    s1;
} else {
    s2;
}
assert Q;
```

What is the weakest P?

Computing $wp(P, S)$: if-then-else

- Suppose $S = \texttt{if}\, B\ \{\, S' \,\}\ \texttt{else}\ \{\, S'' \,\}$, where B is condition and S, S' are blocks of statements. What is $wp(S, Q)$?
  - Suppose we compute $P_1 = wp(S', Q)$, $P_2 = wp(S'', Q)$
  - This gives the preconditions under the assumption that B is true (P_1)and under the assumption that B is false (P_2)
  - So $wp(S, P) = (B \Rightarrow P_1) \wedge (\neg B \Rightarrow P_2)$!

Computing $wp(P, S)$: if-then-else

$\{?\}$

```
if x < y {

    min := x;

} else {

    min := y;

}
```

$\{\min \leq x\}$

Computing $wp(P, S)$: if-then-else

$$\{\,?\,\}$$

```
if x < y {
```

$$\{\,?\,\}$$

```
    min := x;
```

$$\{\text{min} \leq \text{x}\}$$

```
} else {
```

$$\{\,?\,\}$$

```
    min := y;
```

$$\{\text{min} \leq \text{x}\}$$

```
}
```

$$\{\text{min} \leq \text{x}\}$$

Computing $wp(P, S)$: if-then-else

$\{\,?\,\}$

```
if x < y {
```

$\{\, x \leq x\}$

```
    min := x;
```

$\{\text{min} \leq \text{x}\}$

```
} else {
```

$\{\, y \leq \text{x}\}$

```
    min := y;
```

$\{\text{min} \leq \text{x}\}$

```
}
```

$\{\text{min} \leq \text{x}\}$

Computing $wp(P, S)$: if-then-else

$\{ x < \mathrm{y} \Rightarrow \mathrm{x} \leq \mathrm{x} \;\&\&\; !(\mathrm{x} < \mathrm{y}) \Rightarrow \mathrm{y} \leq \mathrm{x}\}$

```
if x < y {
```

$\{ x \leq x\}$

```
min := x;
```

$\{\mathrm{min} \leq \mathrm{x}\}$

```
} else {
```

$\{ y \leq \mathrm{x}\}$

```
min := y;
```

$\{\mathrm{min} \leq \mathrm{x}\}$

```
}
```

$\{\mathrm{min} \leq \mathrm{x}\}$

Computing $wp(P, S)$: while loops

```
assert P;
while b
{
    s;
}
assert Q;
```

What is the weakest P?

Computing $wp(P, S)$: while loops

```
??
while b
  invariant I
{
    s;
}
assert Q;
```

Use the invariant

Computing $wp(P, S)$: while loops

$$\{\ ?\ \}$$

```
while x > 0
  invariant x >= 0
{
    x := x - 1;
}
```

$$\{\mathrm{min} \leq \mathrm{x}\}$$

Computing $wp(P, S)$: while loops

$$\{x \geq 0\}$$

```
while x > 0
  invariant x >= 0
{
    x := x - 1;
}
```

$$\{\min \leq x\}$$

Why?

```
method Min(x:int,y:int) returns (min : int)
  requires true
  ensures min <= x
{
  if x < y {
    min := x;
  } else {
    min := y;
  }
}
```

```
method Min(x:int,y:int) returns (min : int)
  requires true
  ensures min <= x
{
  if x < y {
    min := x;
  } else {
    min := y;
  }
}
```

```
method Min(x:int,y:int) returns (min : int)
  requires true
  ensures min <= x
{

if x < y {
    min := x;
  } else {
    min := y;
  }
```
$\{min \leq x\}$
```
}
```

```
method Min(x:int,y:int) returns (min : int)
  requires true
  ensures min <= x
{
```

$\{x < y \Rightarrow x \leq x \;\&\&\; !(x < y) \Rightarrow y \leq x\}$

```
  if x < y {
    min := x;
  } else {
    min := y;
  }
```

$\{min \leq x\}$

```
}
```

Weakest Precondition

```
method Min(x:int,y:int) returns (min : int)
  requires true
  ensures min <= x
{
```

$$\{x < y \Rightarrow x \leq x \,\&\&\, !(x < y) \Rightarrow y \leq x\}$$

```
  if x < y {
    min := x;
  } else {
    min := y;
  }
```

$$\{min \leq x\}$$

```
}
```

Does this…

…imply this?

Verification Conditions: while loops

$\{P\}$

```
while b
  invariant I
{
    s
}
```

$\{Q\}$

Verification Conditions: while loops

$\{P\}$

$\{I\}$

```
while b
  invariant I
{
     s
}
```

$\{Q\}$

Weakest Precondition

Verification Conditions: while loops

$\{P\}$

Does this…

$\{I\}$

…imply this?

```
while b
    invariant I
{
        s
}
```

$\{Q\}$

Verification Conditions: while loops

$$\{ P \} \rightarrow \{ I \}$$

```
while b
  invariant I
{
```

$$\{ I \&\& b\} \rightarrow wp\ (s, I)$$

```
    s
```

$$\{ I \}$$

```
}
```

$$\{I \&\& !\mathrm{b}\} \rightarrow \{Q\}$$

Tautology-proving in Dafny

- Dafny proves tautologies when verifying code
  - Needs to prove that method preconditions imply the weakest precondition of method postconditionsfollowing statements
- Uses “SMT” (= “Satisfaction Modulo Theories”) solvers
- We will see how SMT solvers work….

SMT Solving Uses SAT Solving

- SMT solvers rely on “SAT solvers”
- SAT solvers determine if propositional formulas are satisfiable
- Propositional formulas consist of variables (p, q, etc.) and propositional operators ($\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$, etc.).

SMT Solving

- Generalizes SAT solving to data theories!
- The SMT problem for data theory $\mathcal{D}$
  - Given: quantifier-free formula (no $\forall, \exists$) predicate calculus formula φ

    φ can involve atomic predicates from $\mathcal{D}$, e.g. $2x + y \leq 0$, as well as propositional connectives $\neg, \vee, \wedge$, etc.
  - Determine: is φ satisfiable?

SMT Solving an Active Theory of Research!

- Some SMT solvers: Z3, CVC4, Boolector, …
- Current work focuses on decision procedures for basic data theories, engineering aspects of efficient SMT solving, new applications, …