A puzzle

- A robot places 2x1 domino tiles on the following 8x8 checkerboard with top-left corner and bottom right corner removed.
- Prove that it will never be the case that all spaces are covered.
A puzzle

- Example state reached
Notation

\[ \mathbb{N} = \{0, 1, 2, 3, \ldots\} \]

\[ \mathbb{Z} = \{0, -1, 1, -2, 2, \ldots\} \]

\[ \mathbb{N}_{>0} = \{1, 2, \ldots\} \]

\[ \mathbb{Z}_{>0} = \]
while programs

\[ S ::= \text{skip} \mid x := e \mid \text{if} \ B \ \text{then} \ S \ \text{else} \ S \mid \]
\[ \text{while} \ (B) \{ S \} \]
\[ \text{assert} \ (B) \]
\[ \text{if} \ B \ \text{then} \ S \ \equiv \ \text{if} \ B \ \text{then} \ S \ \text{else} \ \text{skip} \]
\[ \text{while} \ (B) \{ \} \ \equiv \ \text{while} \ (B) \{ \text{skip} \} \]
while programs

\[ \begin{align*}
\text{input: } & \quad n \\
\text{init: } & \quad i := 1 \\
\text{while } (n > 0) \{ & \\
\quad & \quad i := i \times n; \\
\quad & \quad n := n - 1 \\
\} & \quad \text{assert } n > 0; \\
\text{assert } & \quad \text{assert } n \geq 0 \\
\text{post: } & \quad \text{assert } i = 2; \\
\text{pre: } & \quad \text{assume } i > 0 \\
\text{while } (i > 0) \{ & \\
\quad & \quad i := 1 \\
\quad & \quad m := n \\
\quad & \quad \text{while } (m > 0) \{ \\
\quad & \quad \quad i := i \times m; \\
\quad & \quad \quad m := m - 1 \\
\quad & \quad \} & \quad \text{assert } m = i \\
\} & \quad i := n!
\end{align*} \]
Method foo(int n) {
  \@pre: n > 0
  int i, out;
  i := 1; out := 0;
  while (i <= n) {
    out := out + i;
    i := i + 1;
  }
  assert (out = n(n+1)/2)
}
The image shows a flowchart with labeled nodes and transitions. The flowchart describes a loop with the following invariant statements:

1. \( \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \)
2. \( \eta > 0 \land i = 1 \)
3. \( \eta > 0 \land \text{out} = 0 \)
4. \( \eta > 0 \land \text{out} = \frac{(i-1)i}{2} \)
5. \( \eta > 0 \land \text{out} = \frac{n(n+1)}{2} \)
6. \( \eta > 0 \land \text{out} = \frac{i(i+1)}{2} \)
7. \( \eta > 0 \land \text{out} = \frac{i(i+1)}{2} \land i \leq n+1 \)

The flowchart includes:
- A starting point labeled with an equation: \( i := 1 \)
- A decision node labeled \( i \leq n \)
- Branches for \( Y \) and \( N \)
- A loop with the action \( \text{out} := \text{out} + i \)
- A decision node labeled \( \text{out} \)
- A decision node labeled \( i := i + 1 \)
- A final node labeled \( \text{HALT} \)
\[
\sum_{i=1}^{n} i = \frac{n(n+1)}{2}
\]

Invariants

\[ n > 0 \land i = 1 \land \text{out} = 0 \]

\[ n > 0 \land \text{out} = \frac{(i-1)i}{2} \land i \leq n+1 \]

\[ n > 0 \land \text{out} = \frac{i(i+1)}{2} \land i \leq n \]

HALT

\[ n > 0 \land \text{out} = \frac{n(n+1)}{2} \]

Inductive Invariant

\[ n > 0 \land \text{out} = \frac{(i-1)i}{2} \land i \leq n+1 \]

\[ n > 0 \land \text{out} = \frac{i(i+1)}{2} \land i \leq n \]
i := 1

out := 0

i ≤ n

out := out + i

i := i + 1

HALT
Induction.

\[ \forall n \in \mathbb{N} \ P(n) \]
\[ \forall n \in \mathbb{N} \ I(n) \]
\[ \forall n \ I(n) \Rightarrow P(n) \]

I(0)

\[ \forall k \geq 0 \ (I(k) \Rightarrow I(k+1)) \]
class Example {
  method M(n: int) returns (out: int)
  requires n>0;
  ensures out==n*(n+1)/2;
  {
    var i := 1;
    out:=0;
    while (i <= n)
    {
      invariant out===(i-1)*i/2;
      invariant (i<=n+1)
    }
    out := out + i;
    i := i+1;
  }
}
Part II
Induction

\[ \forall n \in \mathbb{N} \quad P(n) \]

\[ \forall k > 0 \left( (\forall i < k \quad P(i)) \Rightarrow P(k) \right) \]

\[ \Rightarrow \forall n \quad P(n) \]

Base case

Induction Step
Proof

• Color the board with alternating black/white (like a chess board)
• Notice $\#\text{blacks} = \#\text{whites} + 2$
  (since two white squares are removed)

• Invariant:
  No matter what state is reached, the number of white sq covered = number of black squares covered

• Hence all squares can never get covered
Proof

Initially, invariant holds.

From a state where the invariant holds, when a domino is placed, the new state will continue to satisfy the invariant.

The state where all spaces are covered is not included in the invariant.

Hence, by induction on number of steps the robot takes, we can show that it will never reach the state where all spaces are covered (the invariant being the induction hypothesis).
Invariant need not be set of reachable states

• Note that the invariant is not precisely the set of reachable states.

• E.g., the following is satisfied by the invariant but is not reachable
System Invariants

Inductive Invariant: $R$

- $R$
- $I$
- $\Xi$

Init

Bad

Assertion violation
Key properties of inductive invariants

\[
\begin{align*}
\text{init} & \subseteq I \\
\text{bad} \land I & = \emptyset \\
\forall \bar{x}, \bar{x}' \ (I(\bar{x}) \land T(\bar{x}, \bar{x}') \Rightarrow I(\bar{x}') \\
\text{Any such I will include all reachable states} \quad R \subseteq I
\end{align*}
\]
Induction:
Rule out time
Reduce to reasoning of single steps
Inv:
\( pc = 0 \land @pre \lor pc = 1 \land A_1 \lor pc = 2 \land A_2 \lor pc = 3 \land A_3 \lor pc = 4 \land A_4 \lor pc = 5 \land @pre \lor @post \)

@pre = \( A_0 \)

0

\[ i := 1 \]

1

\[ \text{out} := 0 \]

2

\[ i \leq n \]

\( \text{Y} \)

A_3

3

\[ \text{out} := \text{out} + i \]

A_4

4

\[ i := i + 1 \]

\[ \text{N} \]

A_5

5

A_5 = @post

HALT
\[ i := 1 \]

\[ \text{Inv: } \]
\[ pc = 0 \land @p_{\text{pre}} \lor pc = 1 \land A_1 \lor pc = 2 \land A_2 \lor pc = 3 \land A_3 \lor pc = 4 \land A_4 \lor pc = 5 \land @p_{\text{post}} \]

\[ \text{out} := 0 \]

\[ \text{Inv: } @p_{\text{pre}} = A_D \]

\[ i \leq n \]

\[ \text{Y} \]

\[ \text{A}_2 \]

\[ \text{N} \]

\[ \text{A}_3 \]

\[ \text{i} := i + 1 \]

\[ \text{out} := \text{out} + i \]

\[ \text{A}_4 \]

\[ \text{A}_5 \]

\[ @p_{\text{post}} \]

\[ \text{HALT} \]
Verification conditions: what the oracle needs to check

\[
\begin{cases}
\eta > 0 \quad & \text{i := 1} \\
\eta > 0 \land i = 1
\end{cases}
\]

\[
\eta_1 > 0 \land \left( i' = 1 \land \right)
\]

\[
\eta_2 \quad \text{i_2} \quad \text{out}_2
\]
\[
\begin{align*}
i &:= 1 \\
on &:= 0 \\
i &\leq n \\
out &:= out + i \\
i &:= i + 1 \\
@pre & = A_0 \\
@post & = A_5
\end{align*}
\]
\[ \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \]

Algorithm:

1. **i := 1**
2. **out := 0**
3. **i ≤ n**
   - **N** (not true)
     - **HALT**
   - **Y** (true)
     - **out := out + i**
     - **i := i + 1**
     - **n > 0** and **out = \frac{i(i+1)}{2}** and **i ≤ n**
     - **n > 0** and **out = \frac{(i-1)i}{2}** and **i ≤ n+1**
     - **n > 0** and **out = \frac{(i-1)i}{2}** and **i > n**

Invariants:

**Inductive Invariant** vs **Invariant**
Verification conditions:
what the oracle needs to check

Let’s first do \{A3\} \text{out}:=\text{out}+1 \{A4\}

\[
\begin{align*}
\{&n>0 \land \text{out}=\frac{(i-1)i}{2} \land i\leq n \} \\
\text{out} &:= \text{out}+i \\
\{&n>0 \land \text{out} = \frac{i(i+1)}{2} \land i\leq n \}
\end{align*}
\]

Writing in pure logic, using \(n', \text{out}', i'\) for new state:

\[
\begin{align*}
\left[ (n>0 \land \text{out} = \frac{(i-1)i}{2} \land i\leq n) \land \left( \text{out}' = \text{out}+i \right) \right] &\implies \\
\left[ n'>0 \land \text{out}' = \frac{i'(i'+1)}{2} \land i'\leq n' \right]
\end{align*}
\]

Is this always true?

Yes

Semantic & execute statement:
\text{out} := \text{out} + i
Verification conditions: what the oracle needs to check

Let’s do a meta-step \{A3\} out:=out+i; i:=i+1 \{A2\}

\[
\begin{align*}
\{ & n>0 \land \text{out} = \frac{(i-1)i}{2} \land i \leq n \} \\
& \Rightarrow \left[ n_3 > 0 \land \text{out}_3 = \frac{(i-1)i}{2} \land i_3 \leq n_3+1 \right] 
\end{align*}
\]
Hoare triples

• Each triple we need to check:
  like \{A3\} out:=out+1 \{A4\}
  or \{A3\} out:=out+i; i:=i+1 \{A2\}

is called a Hoare triple.

Writing the effect of the statement logically is called the strongest-post (we’ll defined this formally later).

And the whole logical formula we reduce to is called the verification condition.

In order to verify the program, the verification conditions corresponding to all the Hoare triples must be valid (“always true”).
Hoare triples, Strongest-post, VC

Let’s first do \{A3\} \text{out:=out}+1 \{A4\}

\[
\{ n > 0 \land \text{out}\leftarrow \frac{(i-1)i}{2} \land i \leq n \} \quad \text{out:=out}+i \quad \{ n > 0 \land \text{out}\leftarrow \frac{i(i+1)}{2} \land i \leq n \}
\]

Writing in pure logic, using \(n', \text{out}', i'\) for new state:

\[
\left[ (n > 0 \land \text{out}\leftarrow \frac{(i-1)i}{2} \land i \leq n ) \land \left( \text{out}' = \text{out}+i \right) \right] \Rightarrow \left[ n' > 0 \land \text{out}' = \frac{i'(i'+1)}{2} \land i' \leq n' \right]
\]

Verification condition.

\[\text{Strongest post semantics & executing statement} \quad \text{Out := out}+i\]
Recap

• Whole program
  – Write invariants.
    Reduce verification of program to checking one (meta-) step at a time.
• Each meta-step corresponds to a Hoare triple
  Derive the verification condition using strongest post which in turn depends on program semantics.
  Reduce step-verification to pure logical reasoning.
  – Prove verification condition using logical methods
    (automated using logic solvers or manually)
DAFNY

• Expects you to write specifications and inductive invariants

• Automates verification condition generation (through a lower level language called Boogie)

• Tries to automate validating the verification conditions using a logic engine (SMT solvers)
Functional vs Imperative programs

- Functional programs:
  No state; no notion of time;
  Closer to mathematics already.
- Imperative programs
  We need to eliminate time (using induction) and state (using strongest post/VC Gen)
  What results is a “functional” version amenable to verification.

For functional programs, one can immediately write down translate the property required to logic.
Complex specifications/ complex logical reasoning

• How do we specify properties of arrays?
  (e.g., an array is sorted)
• How do we specify complex properties
  (e.g., gcd(a,b)=1 or m= factorial(n) )
• How do we specify properties of dynamically manipulated data-structures (heaps)
  (e.g., x points to a tree ; x points to a bst)
• How do we write abstraction-refinements of datatypes?
  (e.g., the linked list represents a set)
• How do we help the verifier when it cannot verify?
  (ghost code; lemmas)
• How do we refine high-level specifications to low-level specs?
  (e.g., “memory given to processes is disjoint” \[\rightarrow\] disjointness of ranges stored in a linked list)
• How do we talk about objects? Class invariants?
Need to understand logic!

• Propositional logic, proof systems for it, axiom systems, completeness theorem, strong completeness, compactness.
• First-order logic: Axiom systems, completeness, strong completeness, compactness
• First order theories: Incompleteness theorem, various theories useful for verification.