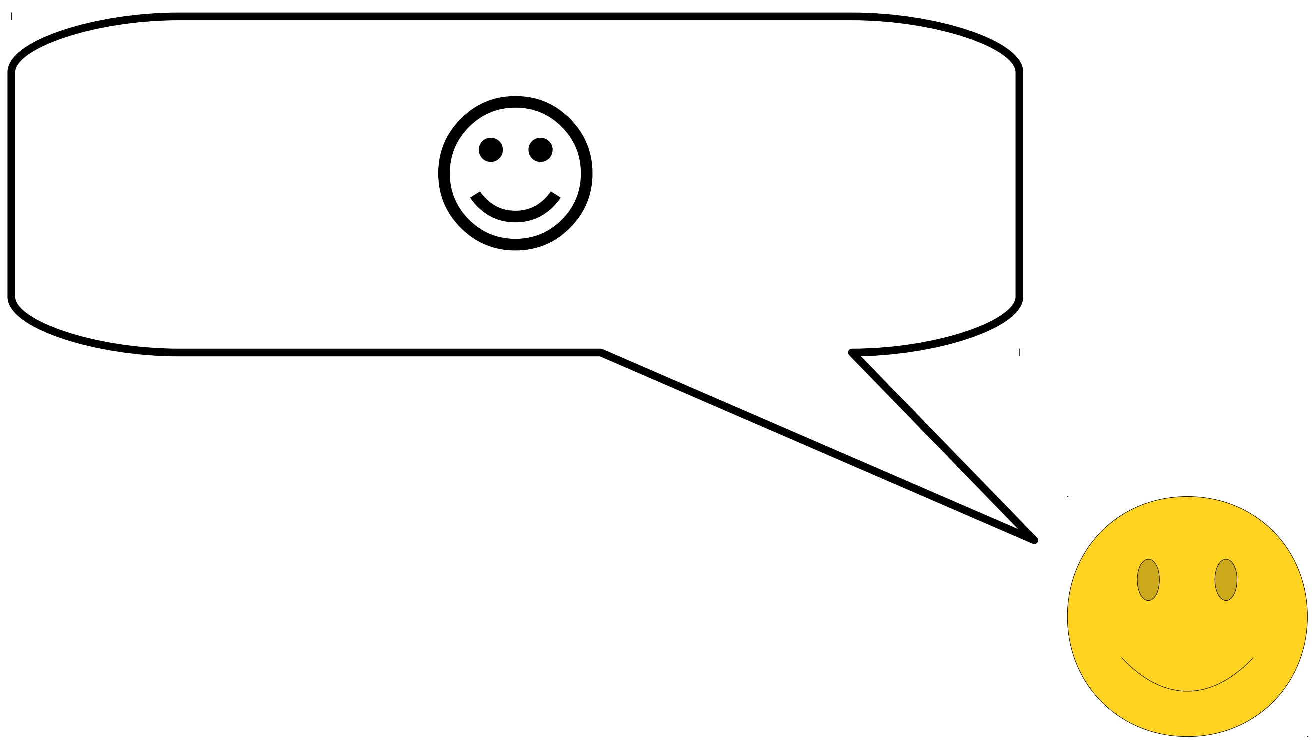
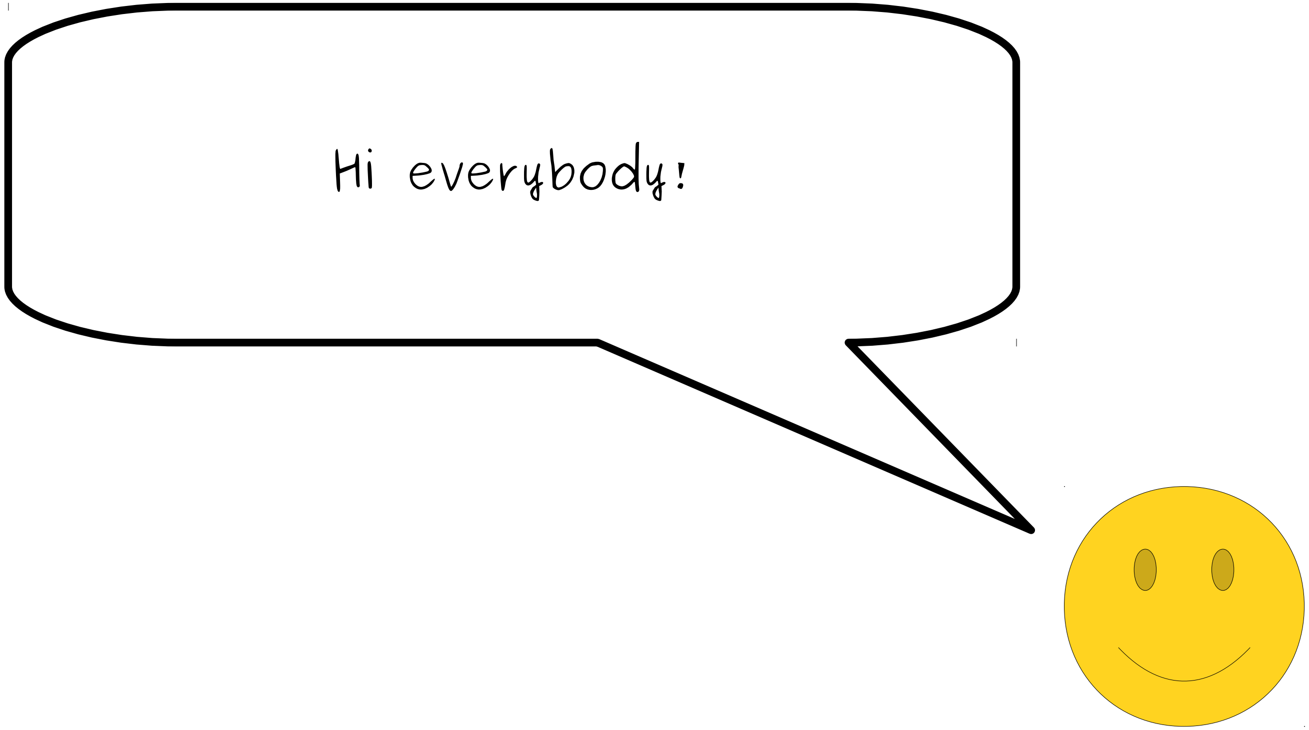
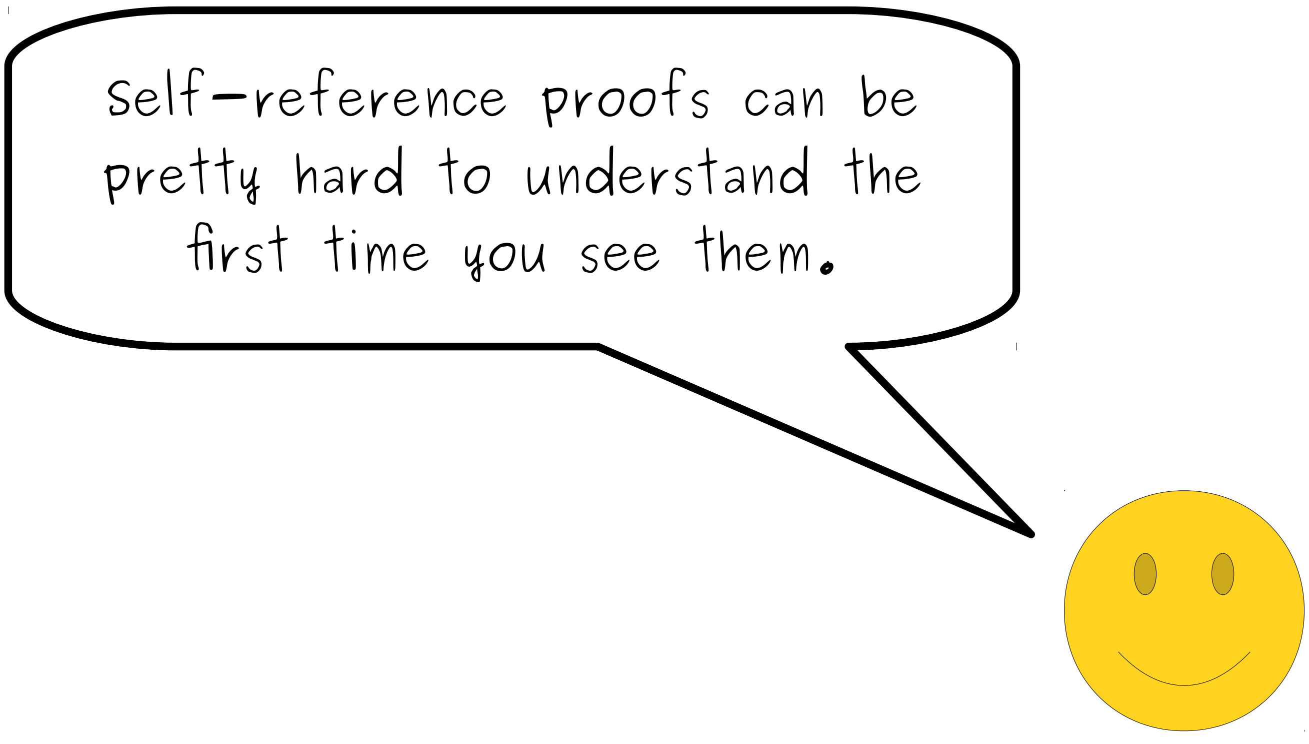
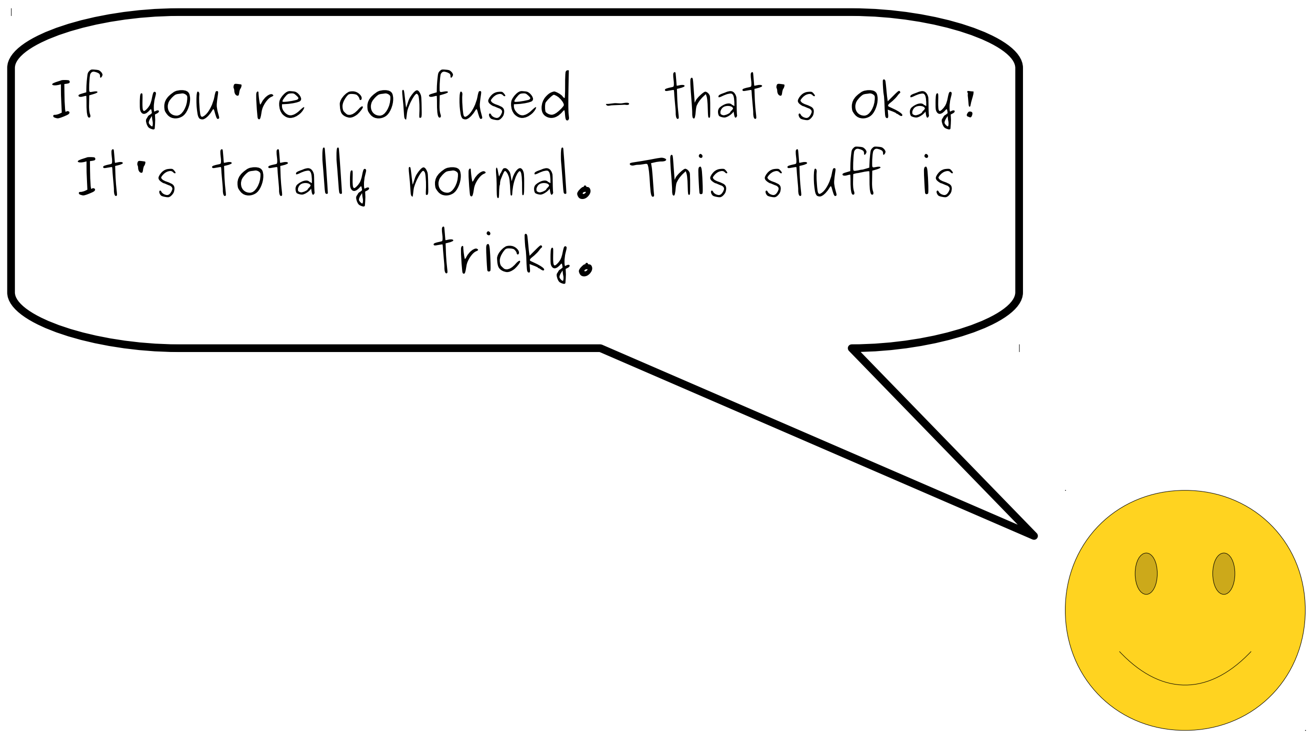
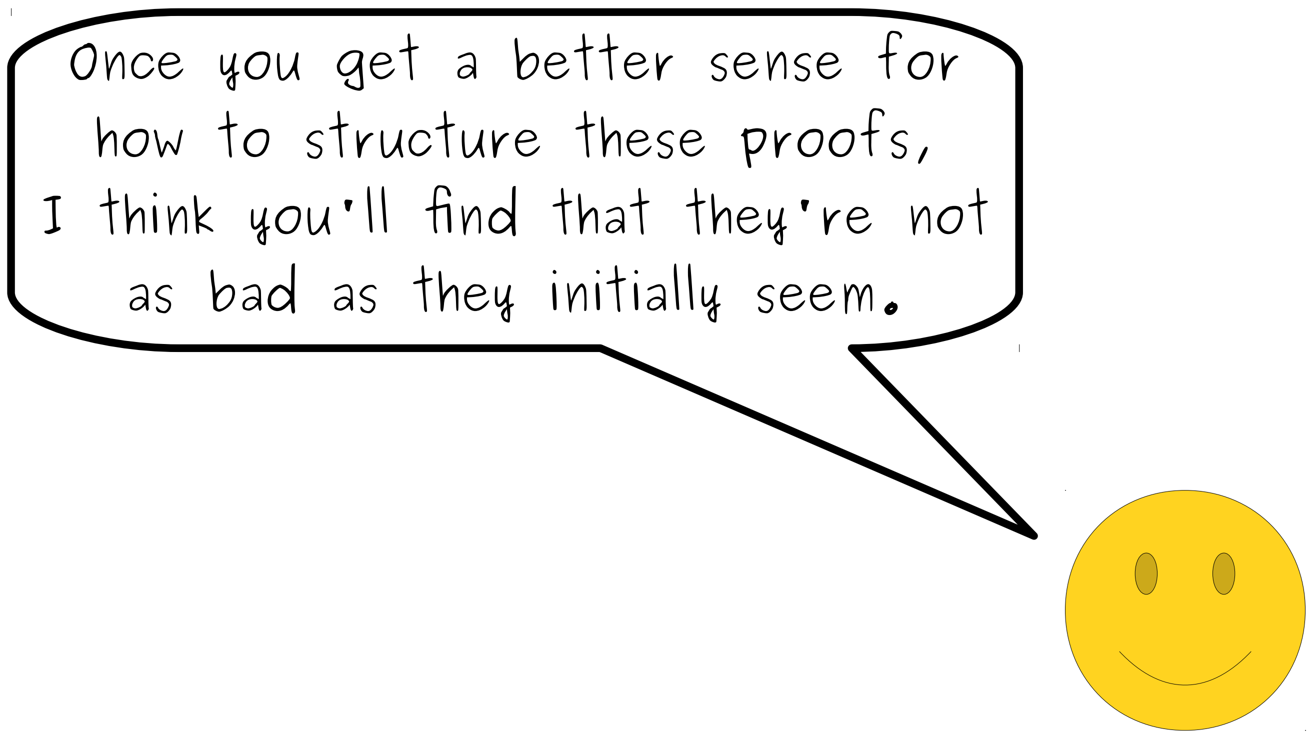
The Guide to Self-Reference




Hi everybody!


Self-reference proofs can be pretty hard to understand the first time you see them.

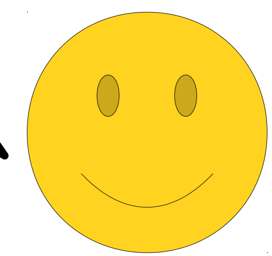

If you're confused - that's okay! It's totally normal. This stuff is tricky.


Once you get a better sense for how to structure these proofs, I think you'll find that they're not as bad as they initially seem.

What does this program do?

```
bool willAccept(st
   /* … some im
}

int main() {
   string me = 
   string input

   if (willAccept(me, input)) {
      reject();
   } else {
      accept();
   }
}
```

This lecture slide was the first time that we really saw self-reference, and a lot of you got pretty tripped up by what was going on.



Try running this program on any input.
What happens if

… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!

What does this program do?

```
bool willAcc
    /* … some
}

int main() {
    string me
    string inpu

    if (willAccept(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

Part of the reason why this can be tricky is that what you're looking at is a finished product. If you don't have a sense of where it comes from, it's really hard to understand!

Try running this program on any input.
What happens if

… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!

What does this program do?

```
bool willAccept(string program, string input) {
    /* … some implementation … */
}

int main() {
    string me = mySource();
    string input = getInput();

    if (willAccept(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

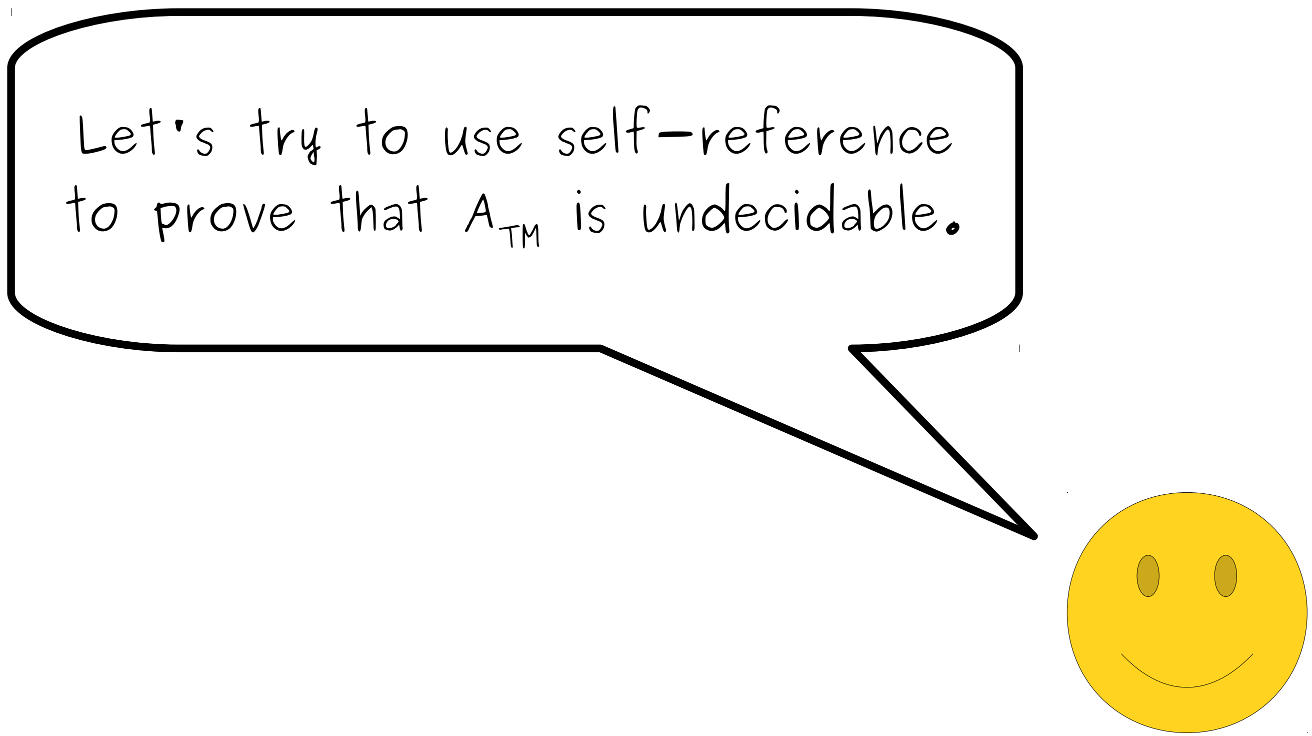
Let's see where it comes from!

We'll take it from the top.

Try running this program on any input.
What happens if

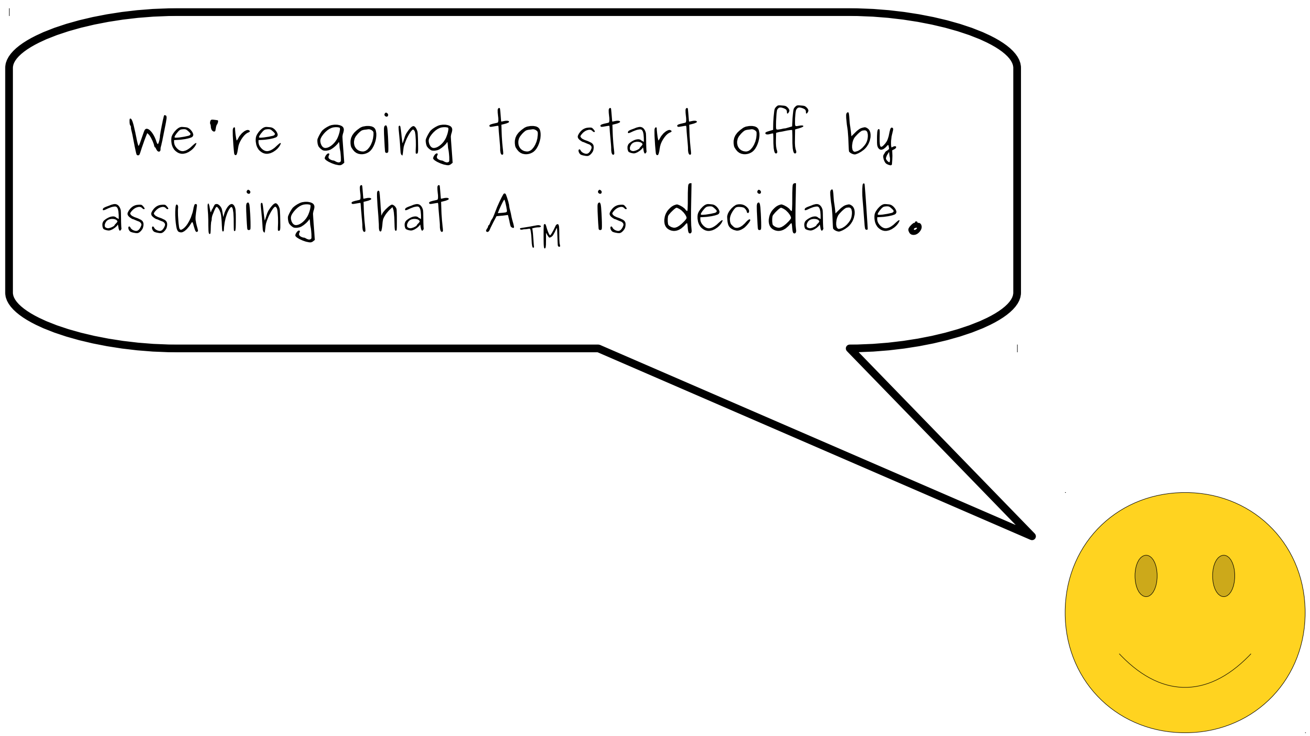
… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!


Let's try to use self-reference to prove that A_{TM} is undecidable.


At a high level, we're going to do a proof by contradiction.

$A_{TM} \in \mathbf{R}$



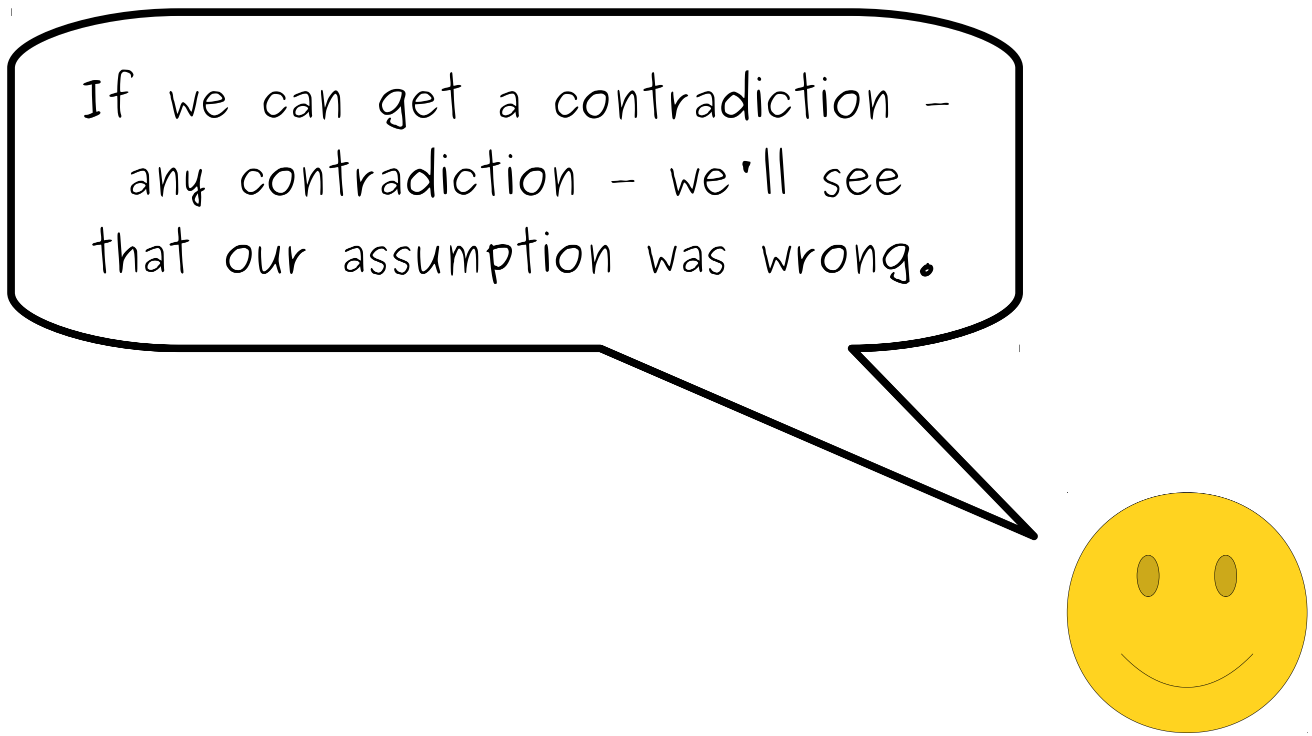
$A_{TM} \in \mathbf{R}$

↓

Contradiction!

$A_{TM} \in \mathbf{R}$

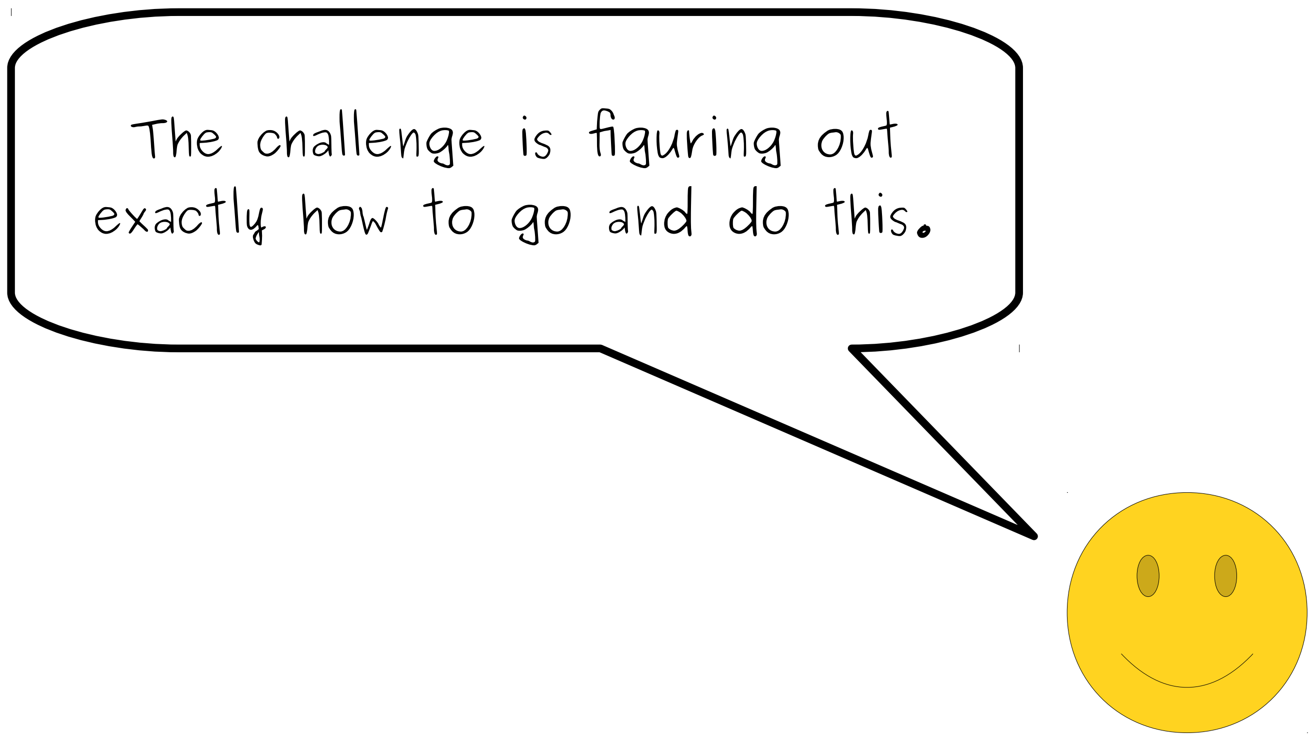
Contradiction!



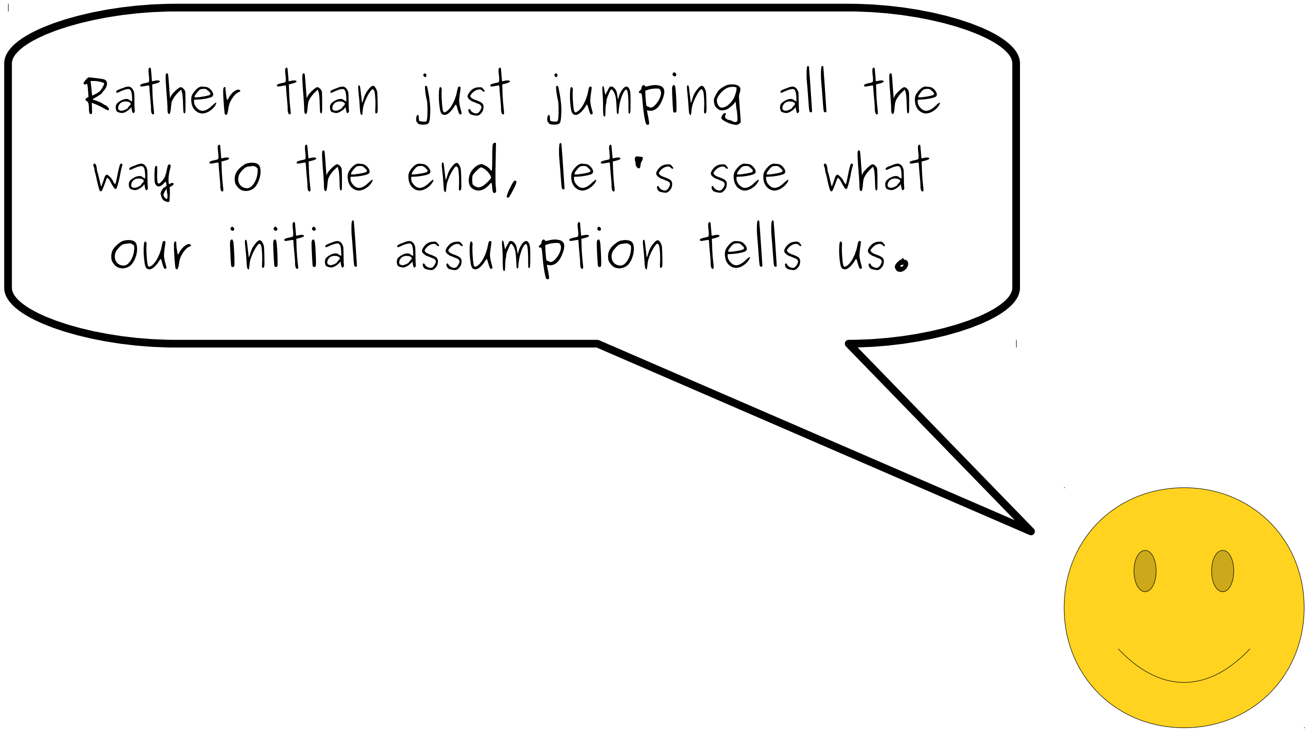
$A_{TM} \in \mathbf{R}$

↓

Contradiction!

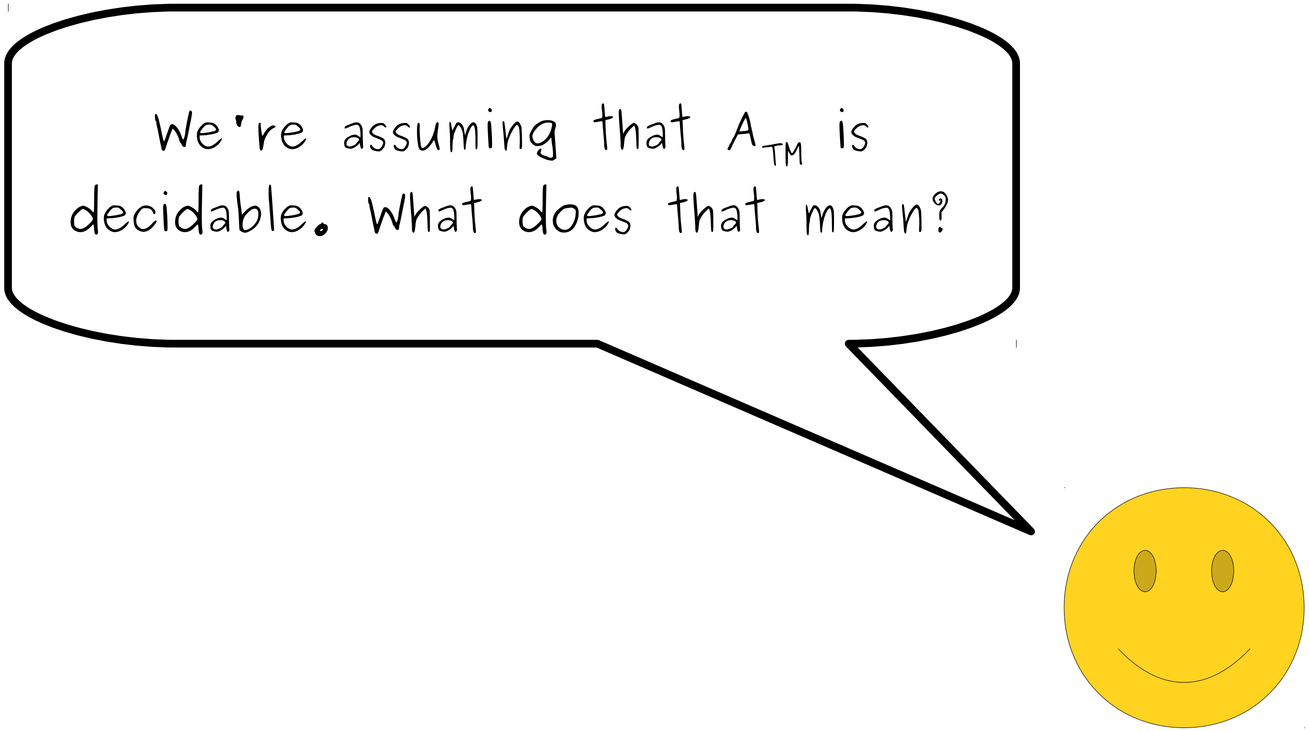


$A_{TM} \in$ **R**



Contradiction!

$A_{TM} \in \mathbf{R}$



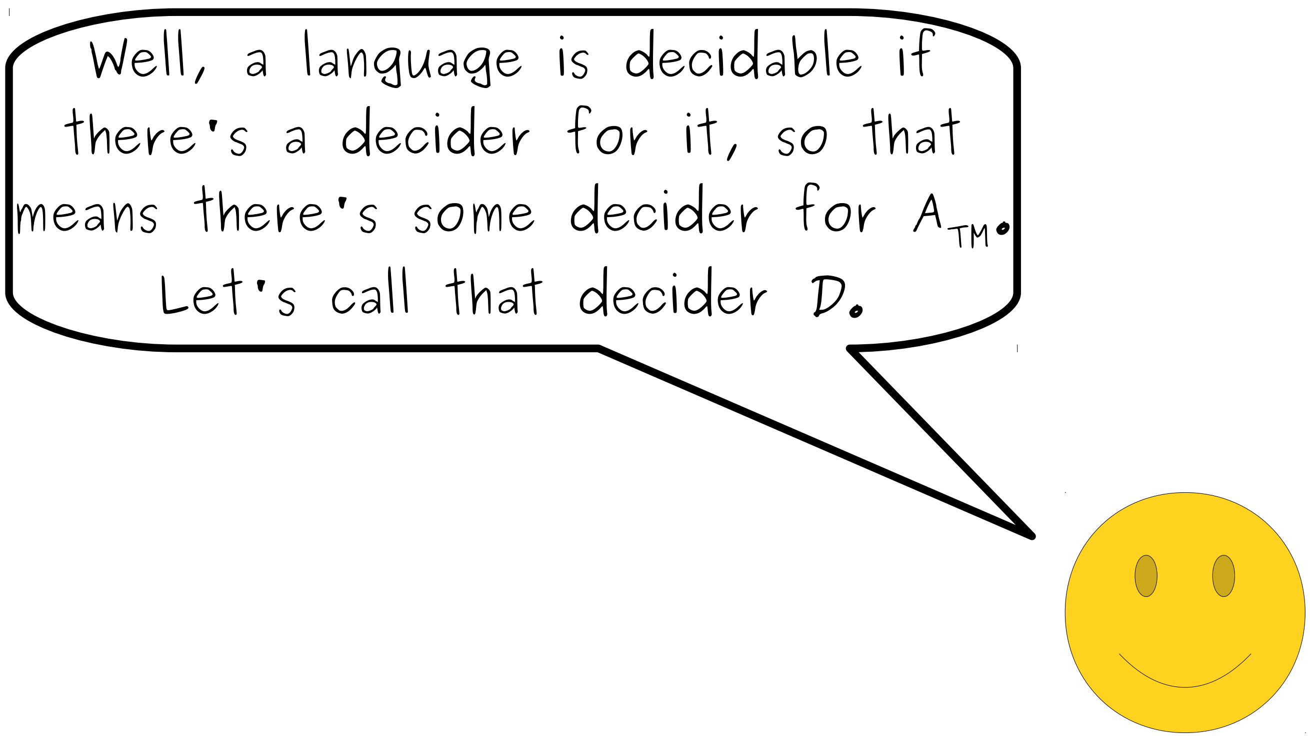
Contradiction!

$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

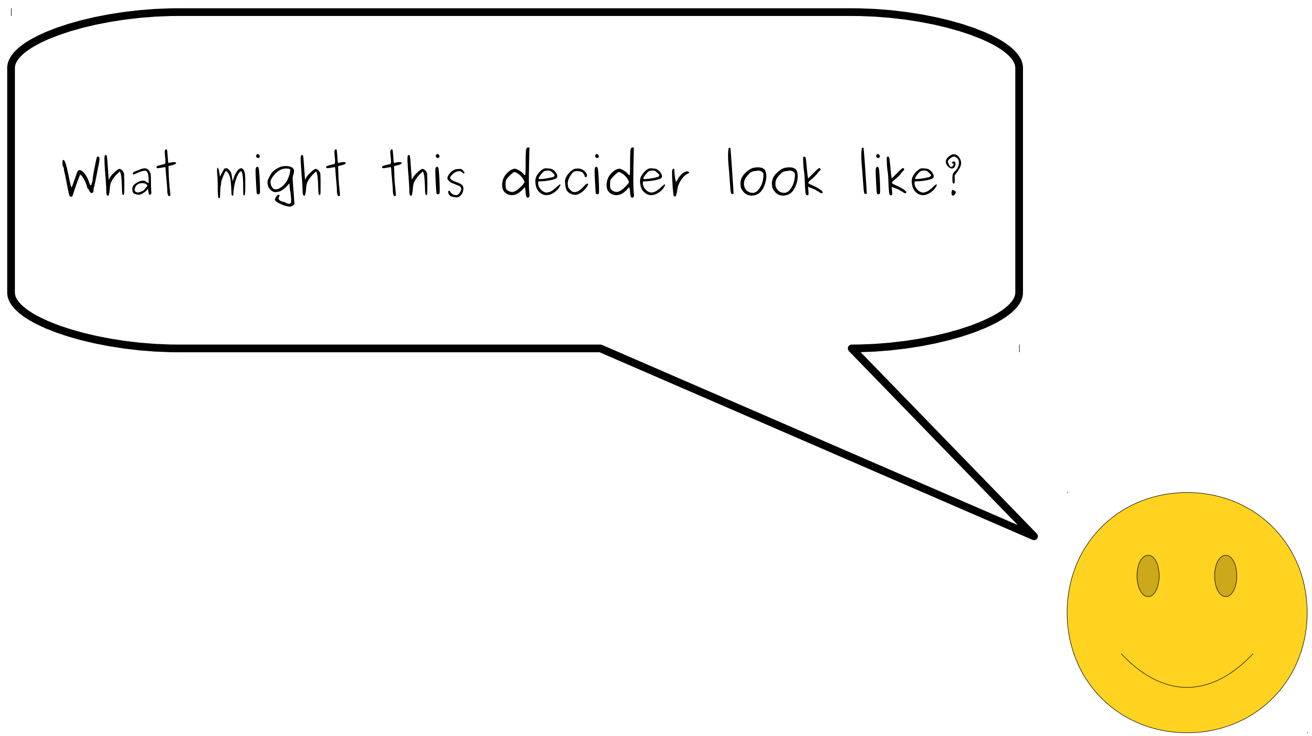
Contradiction!



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}



Contradiction!

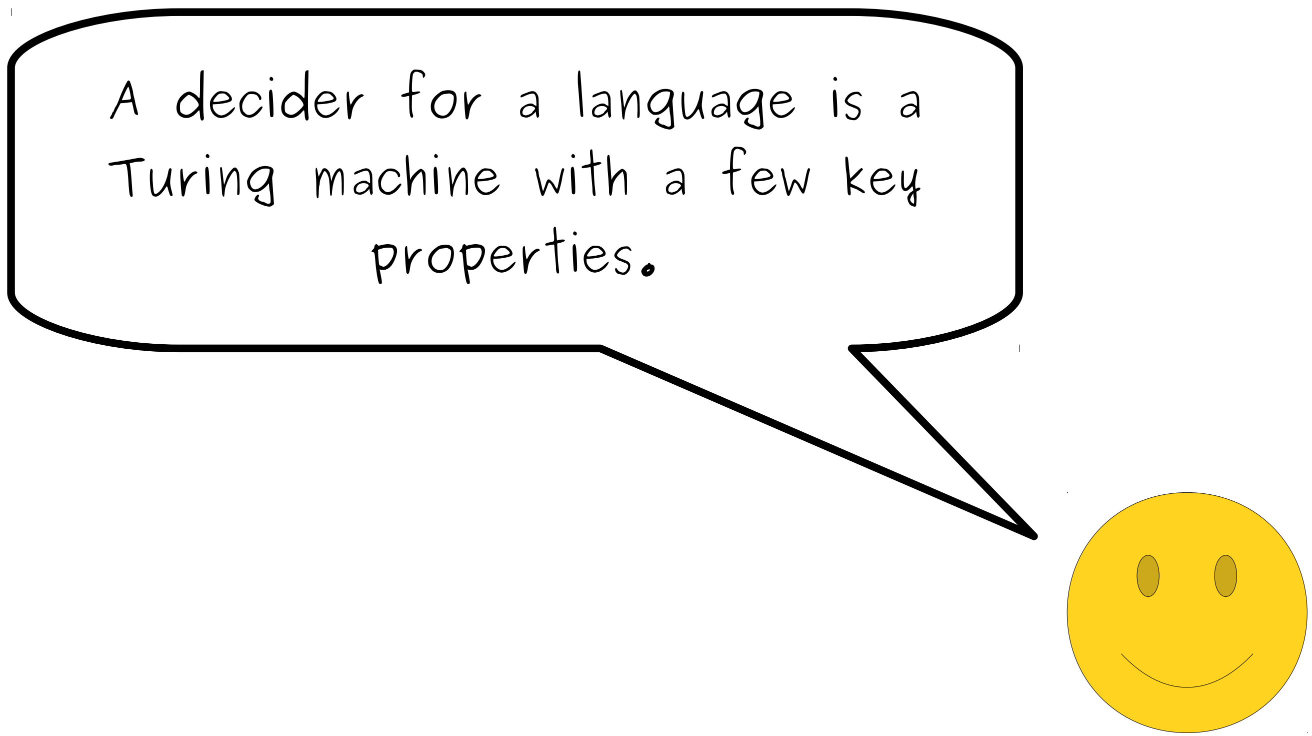
$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

Contradiction!

Decider D for A_{TM}



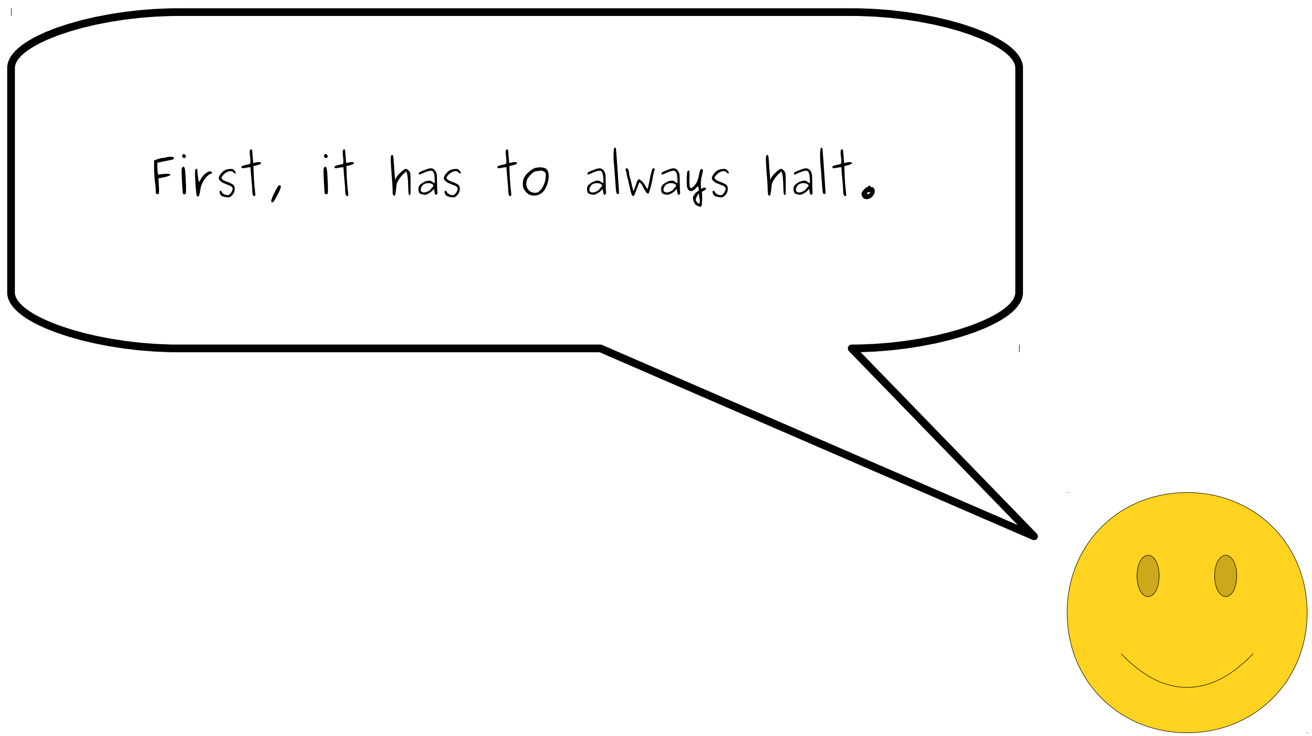
$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

Contradiction!

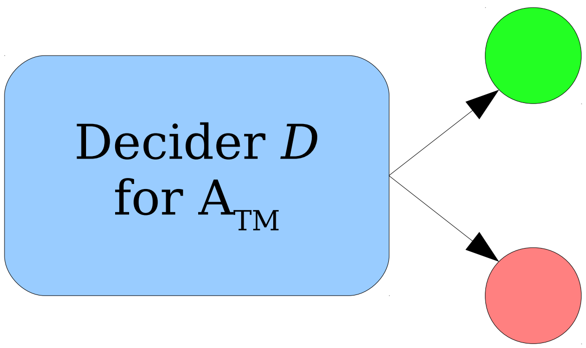
Decider *D* for A_{TM}

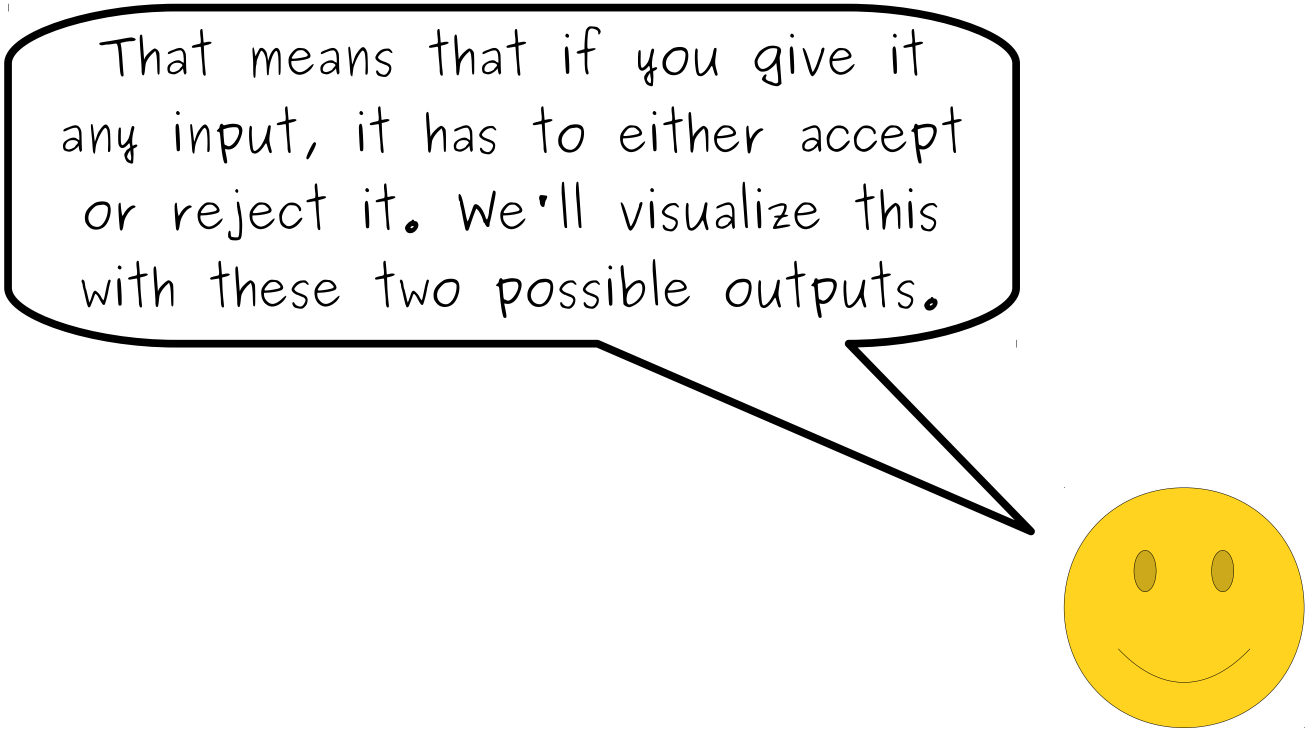


$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

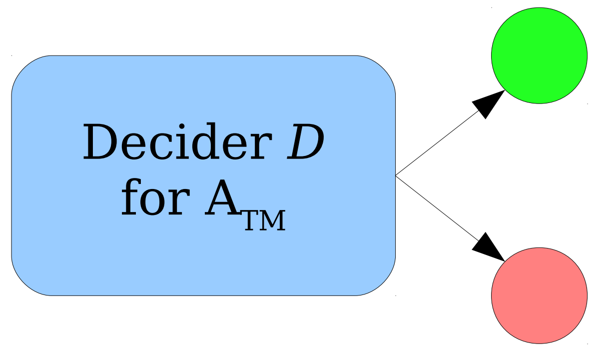


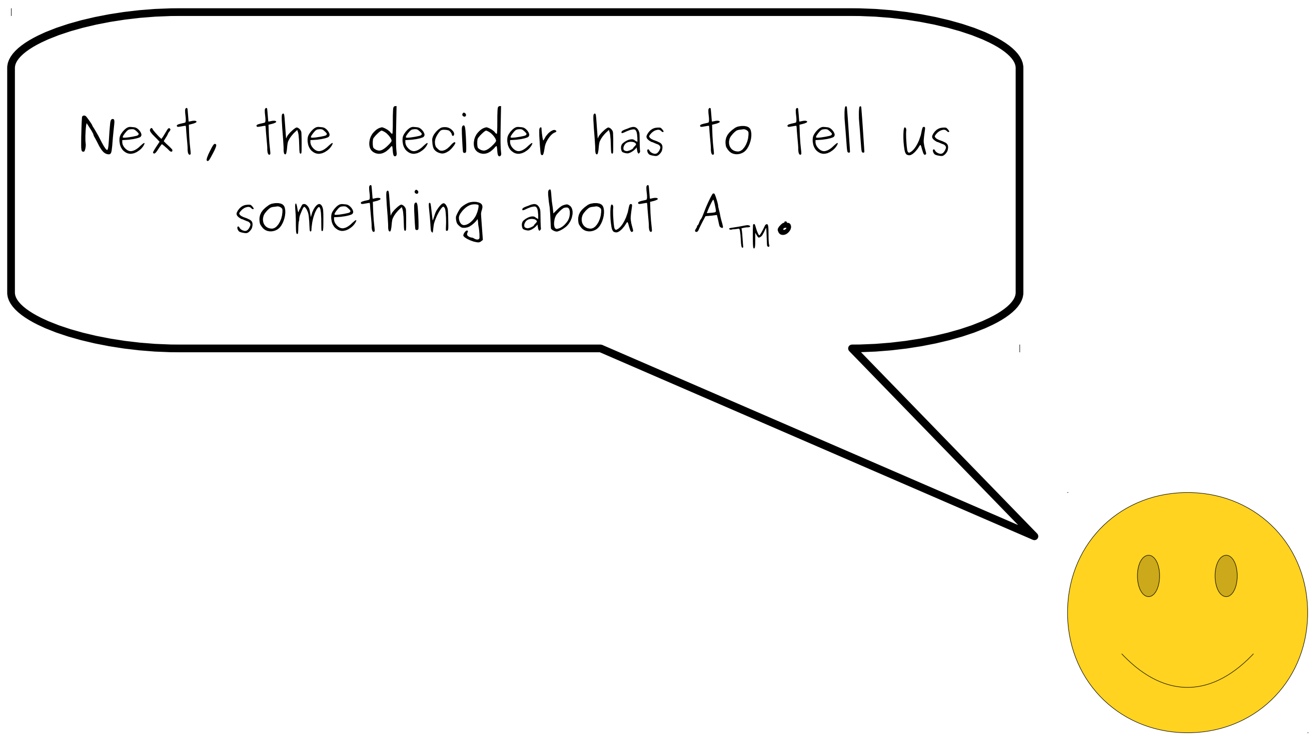


Contradiction!

$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}



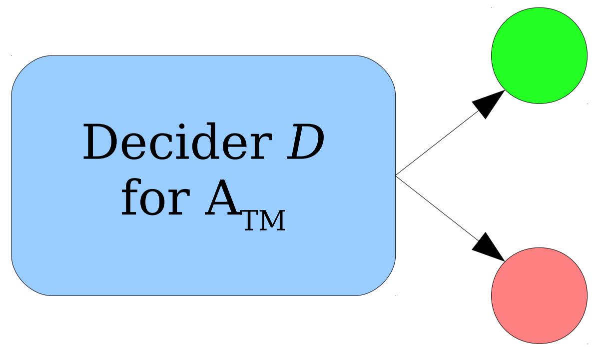


Contradiction!

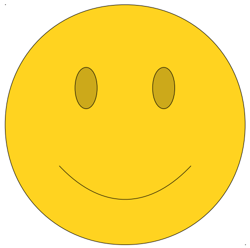
$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

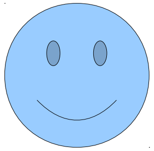


Next, the decider has to tell us something about A_{TM}.



As a reminder, A_{TM} is the language

$\{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$



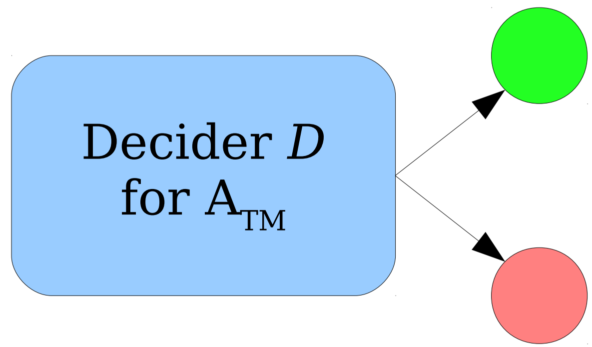
Contradiction!

$A_{TM} \in \mathbf{R}$

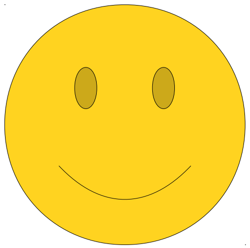
↓

There is a decider D for A_{TM}

Contradiction!

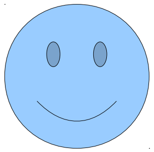


Specifically, the decider D needs to take in an input and tell us whether that input is in A_{TM}



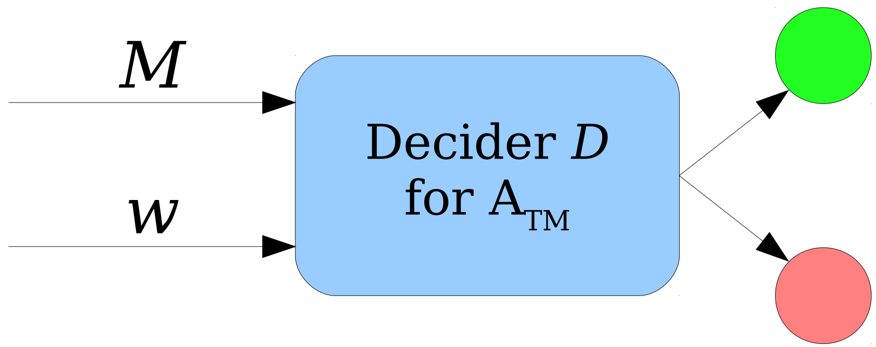
As a reminder, A_{TM} is the language

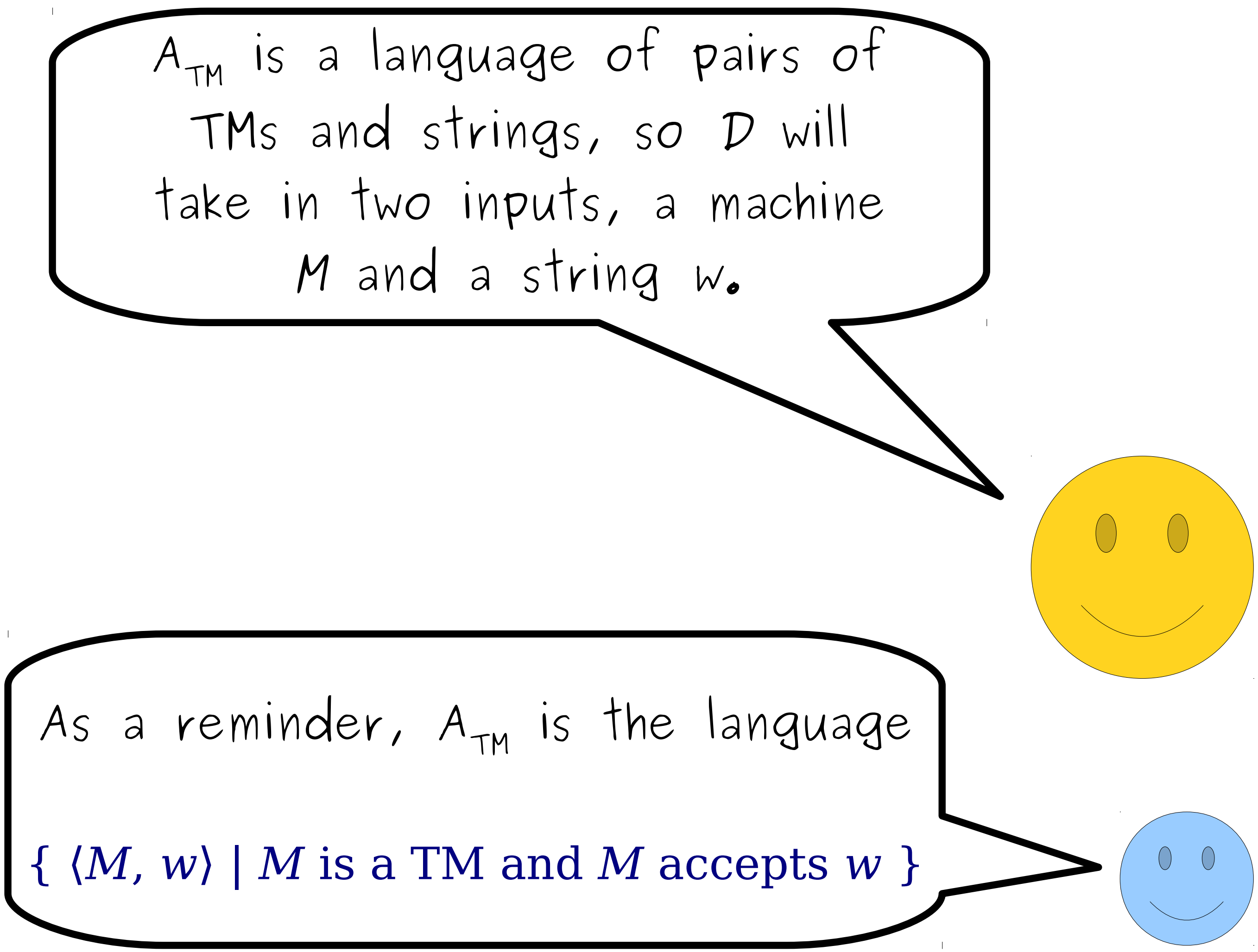
$\{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$



$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}



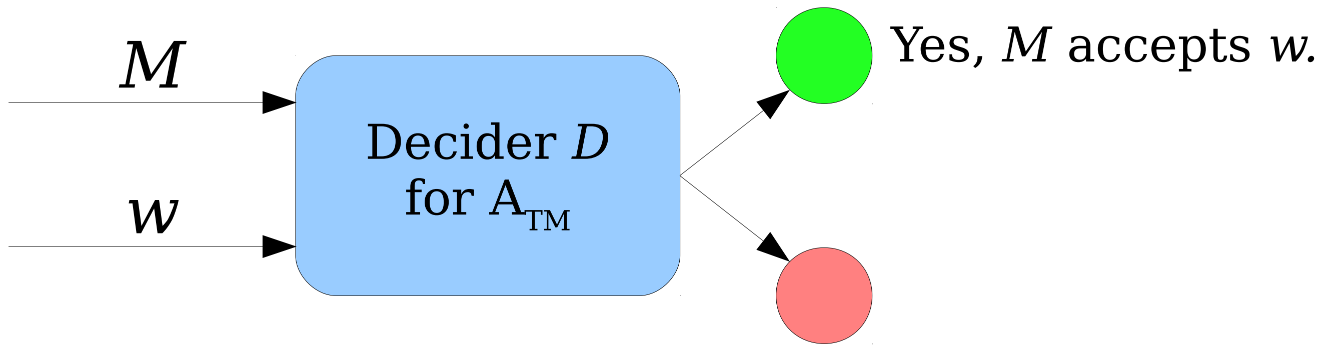


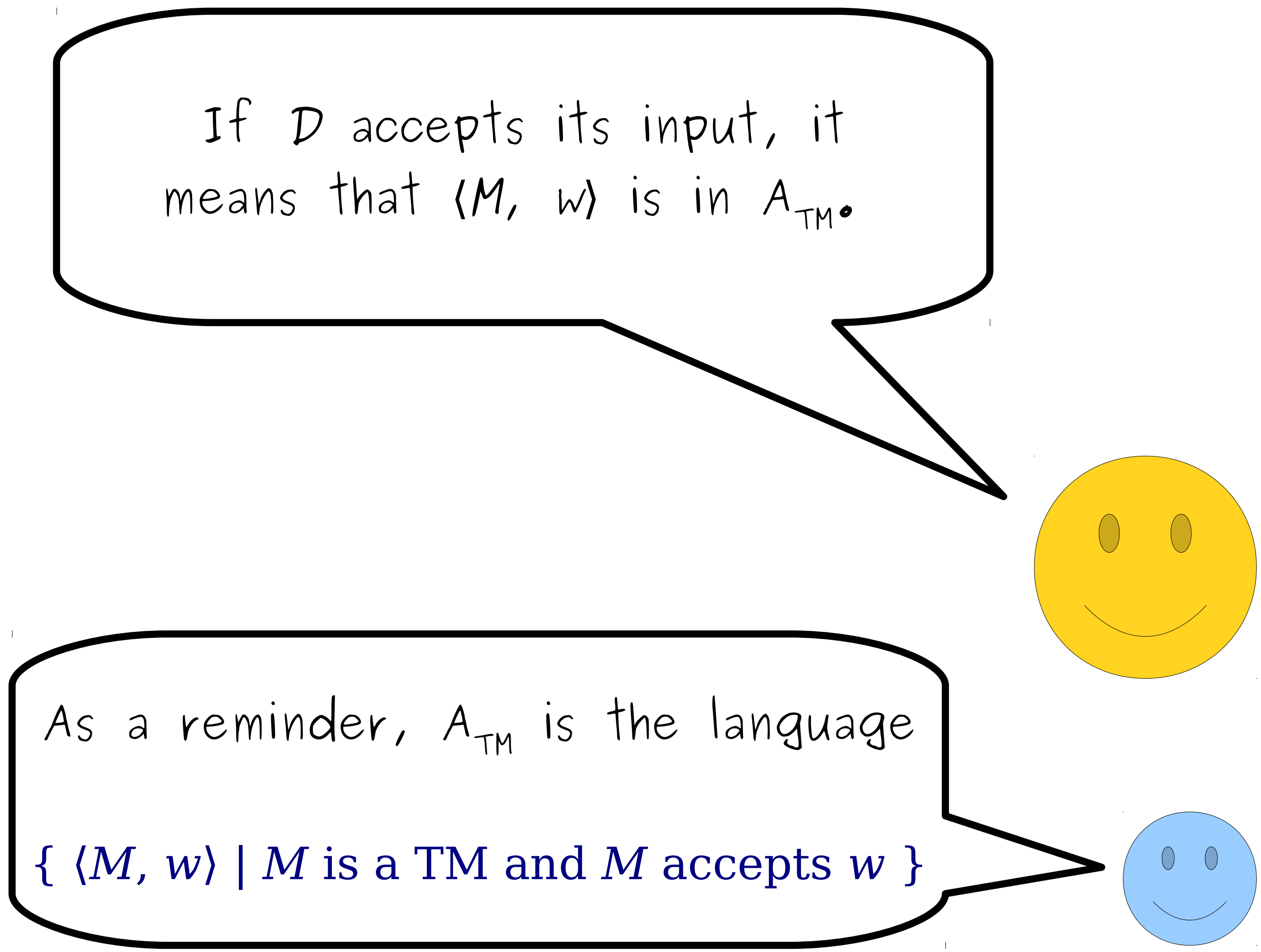
Contradiction!

$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}



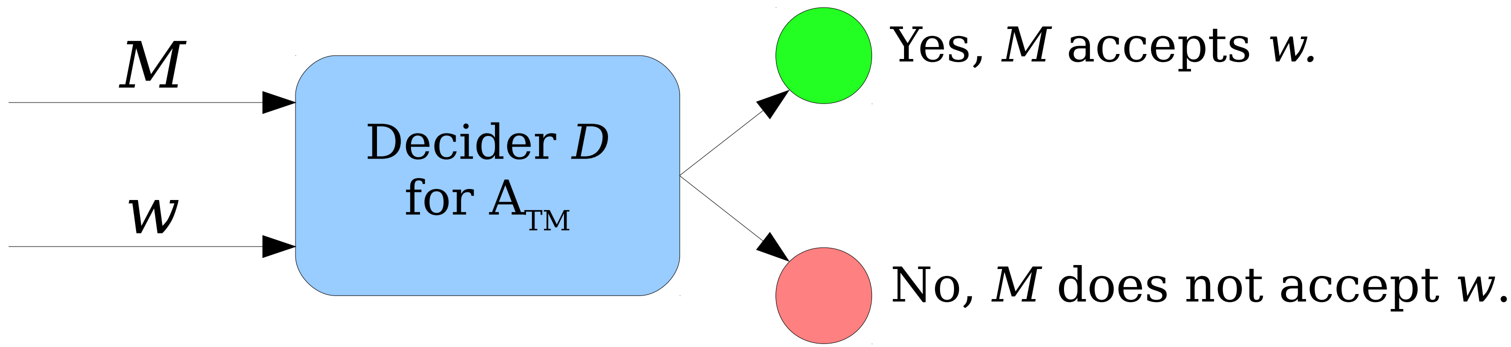


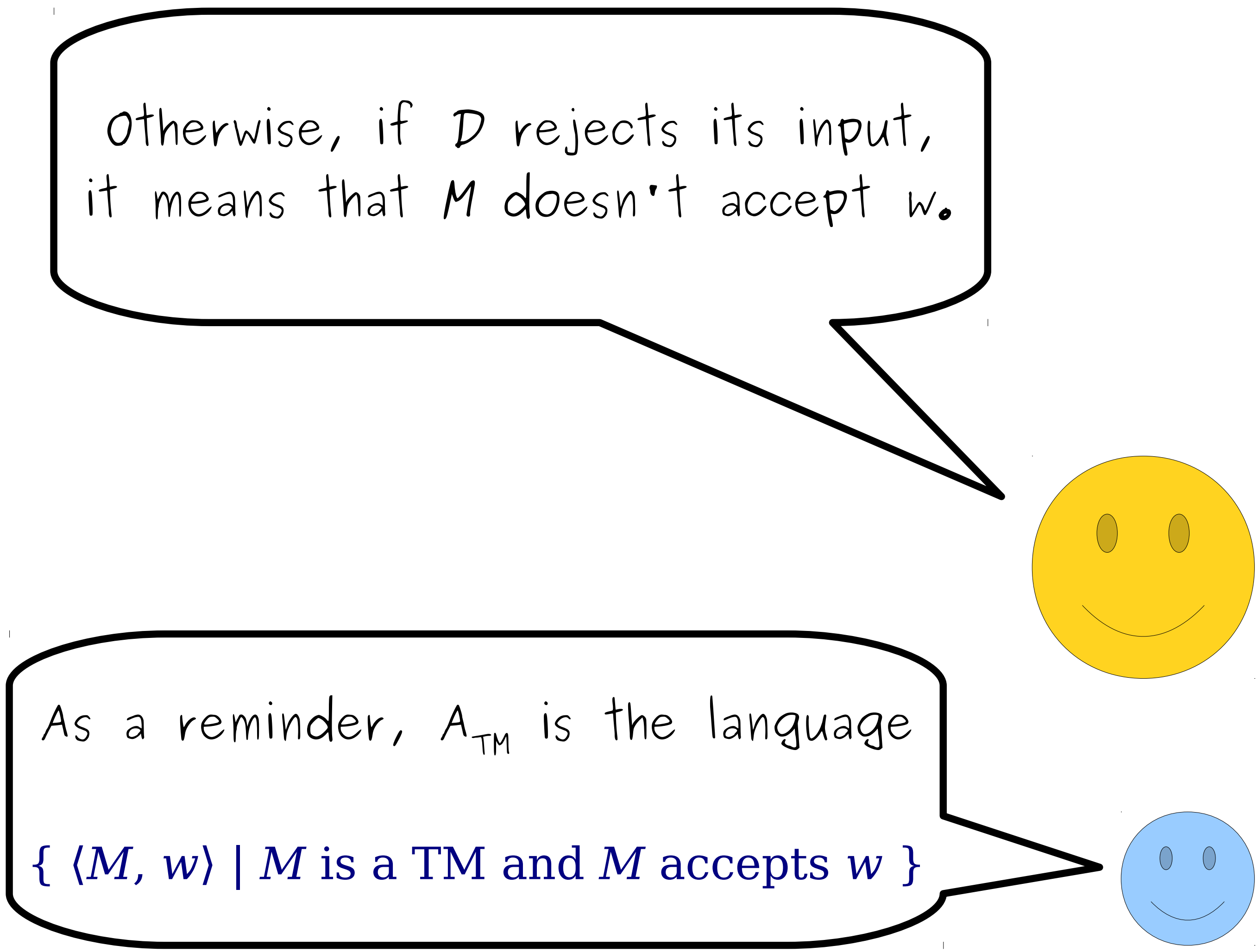
Contradiction!

$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}



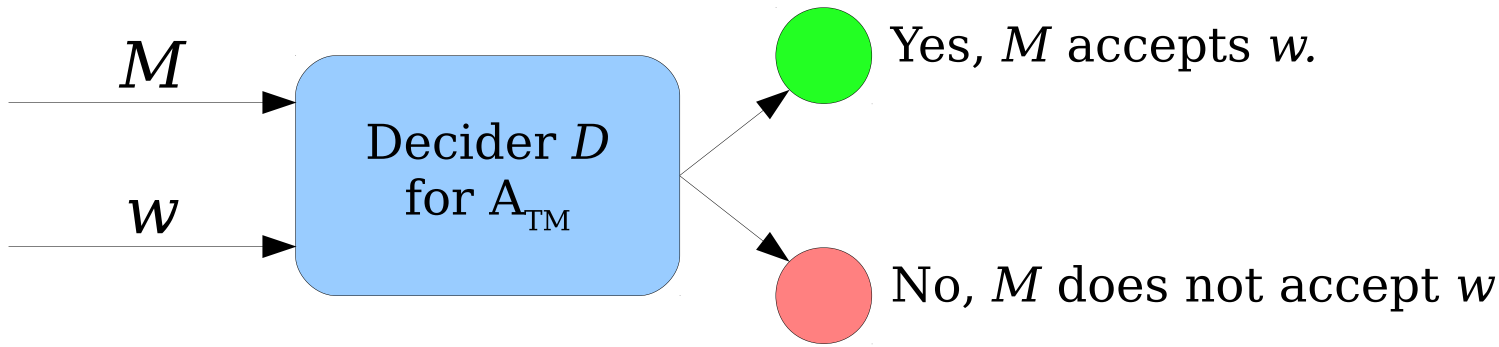


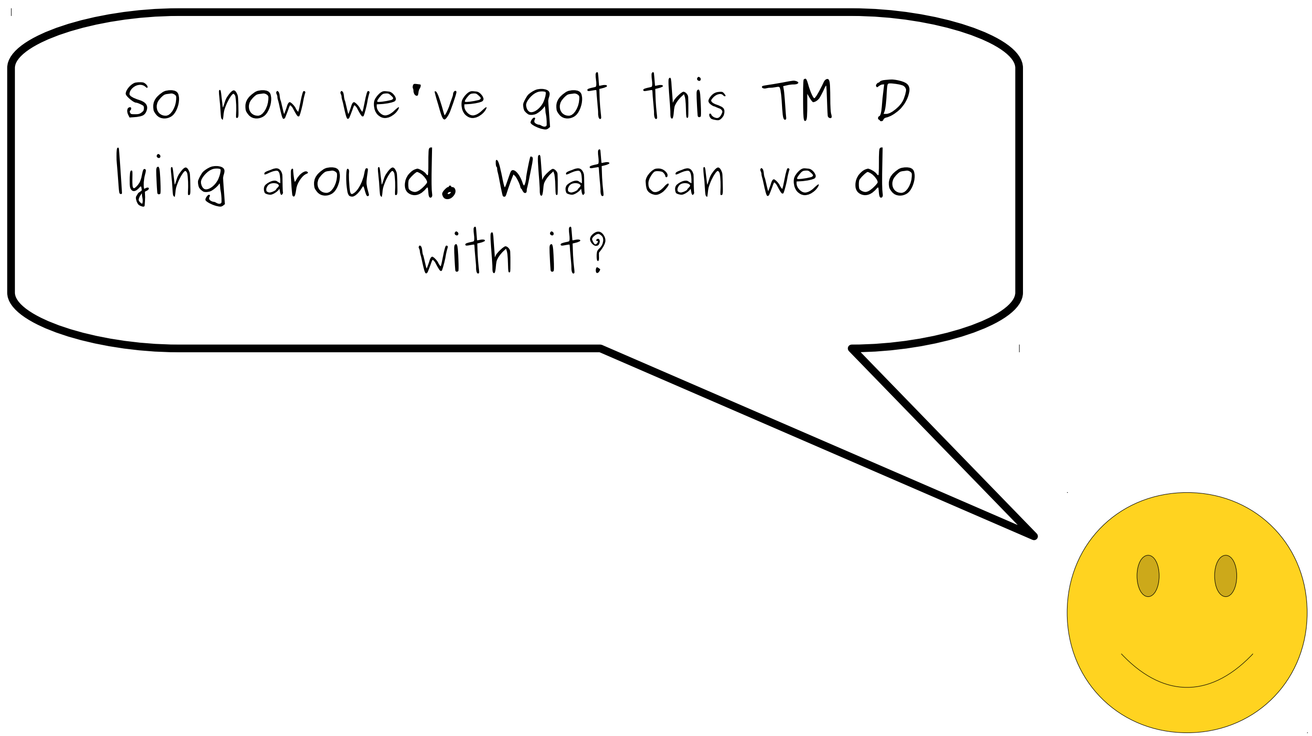
Contradiction!

$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}





Contradiction!

$A_{TM} \in \mathbf{R}$

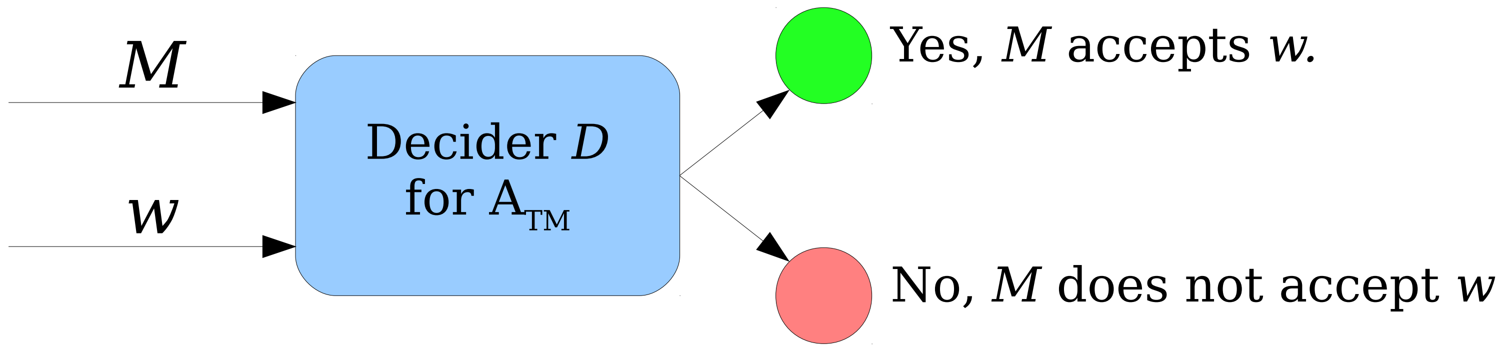
↓

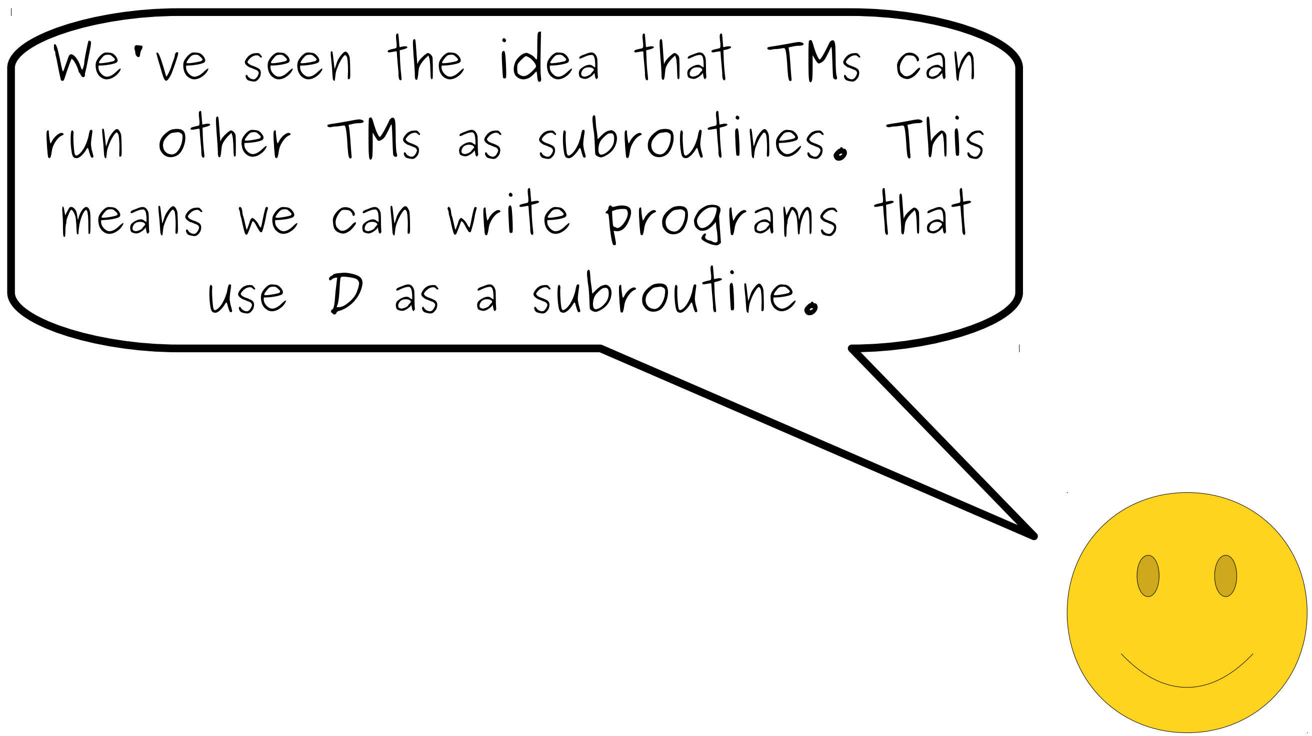
There is a decider *D* for A_{TM}

↓

We can write programs that use *D* as a helper method

Contradiction!





$A_{TM} \in \mathbf{R}$

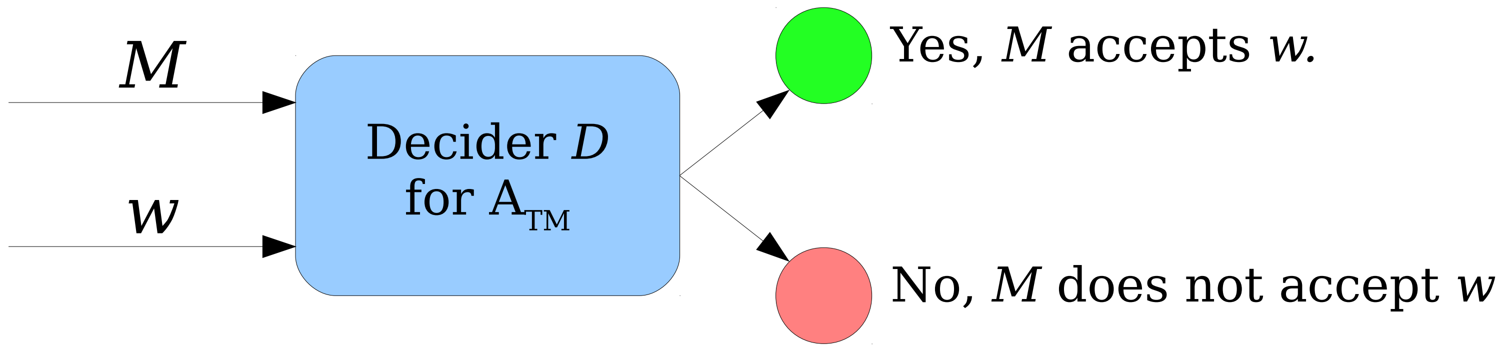
↓

There is a decider D for A_{TM}

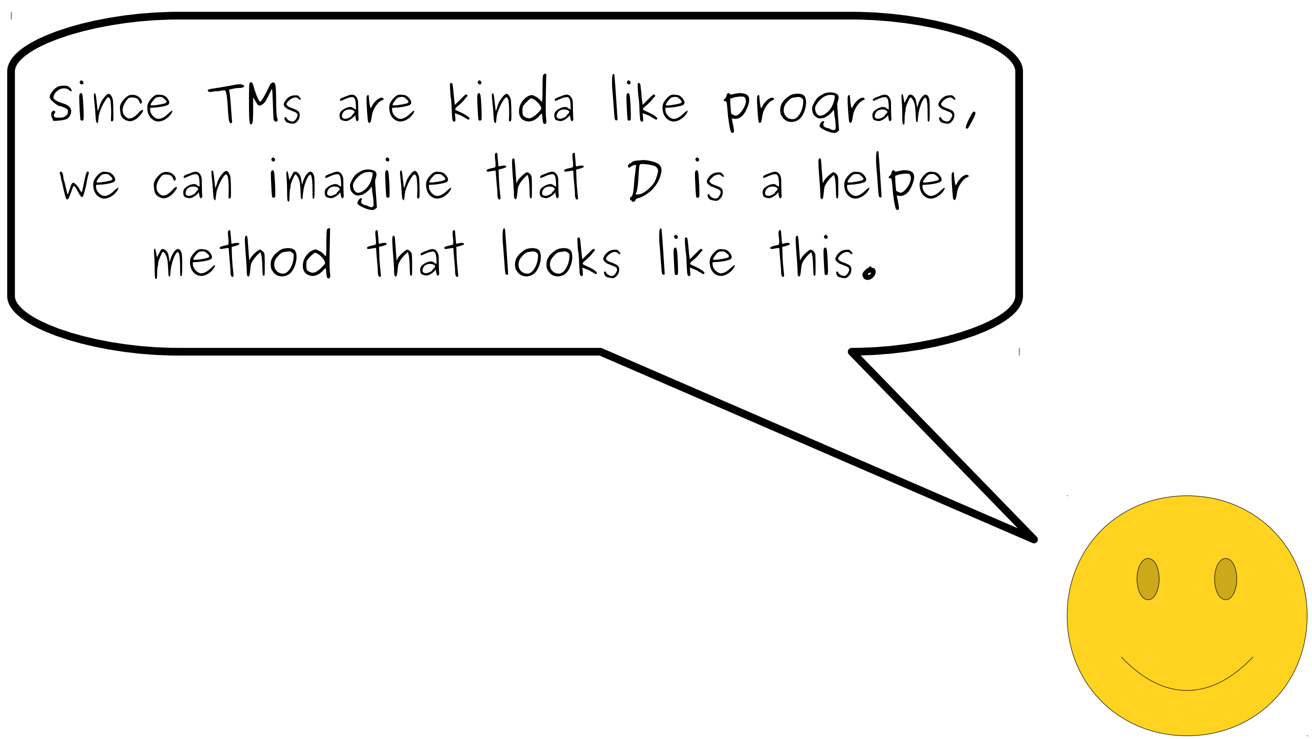
↓

We can write programs that use D as a helper method

Contradiction!




```
bool willAccept(string program, string input)
```

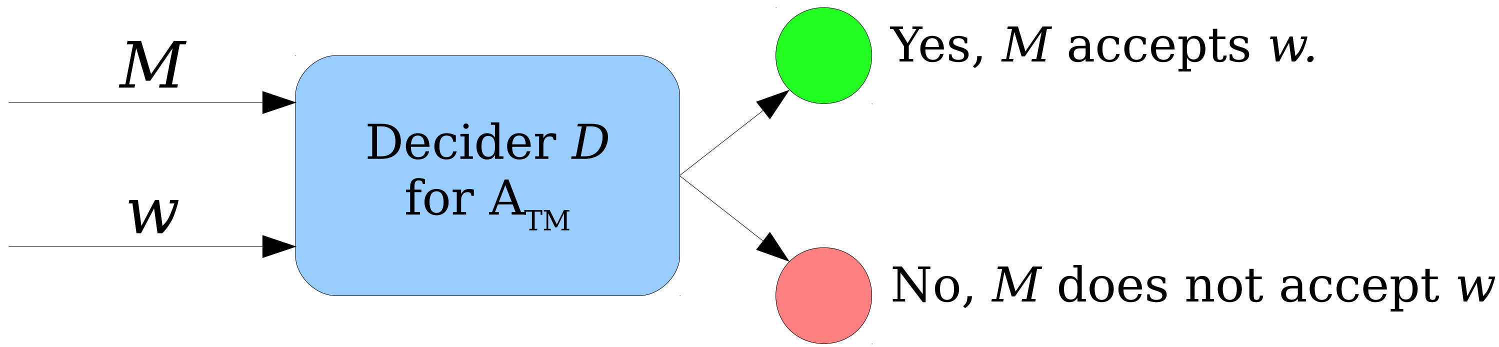


$A_{TM} \in \mathbf{R}$

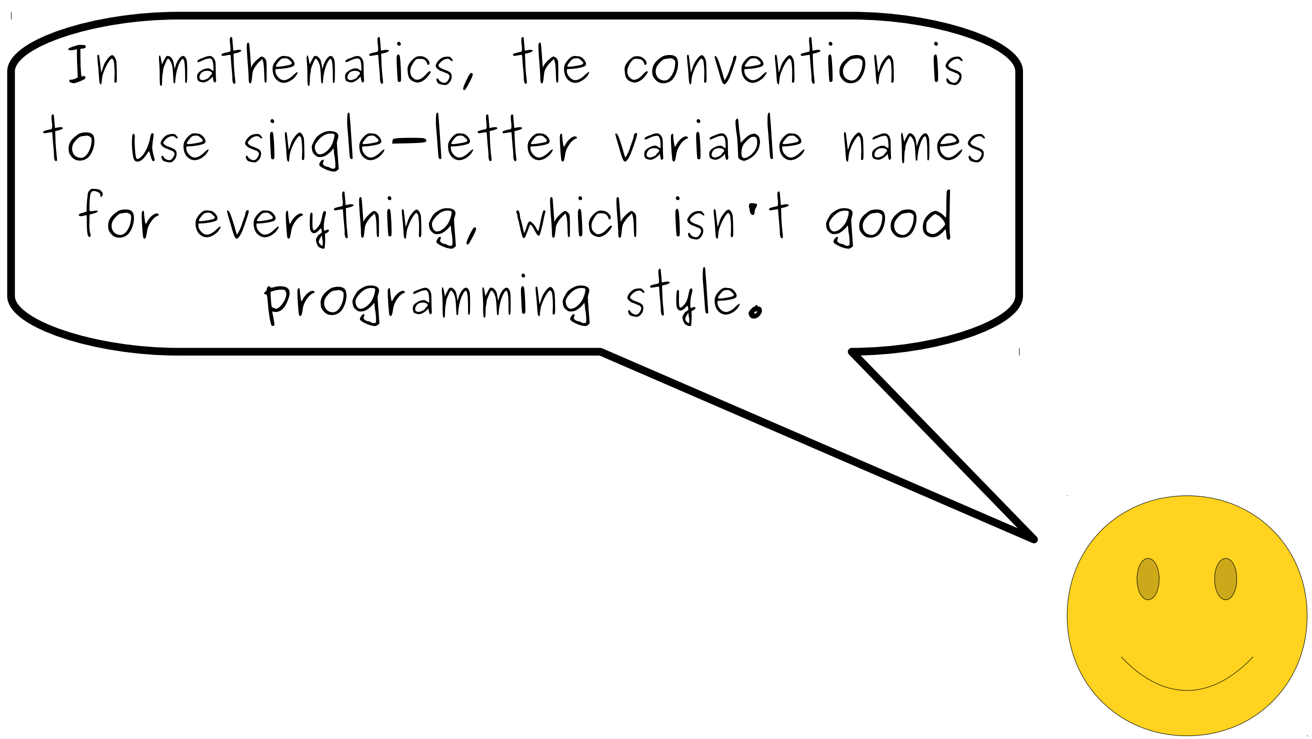
There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Contradiction!




```
bool willAccept(string program, string input)
```

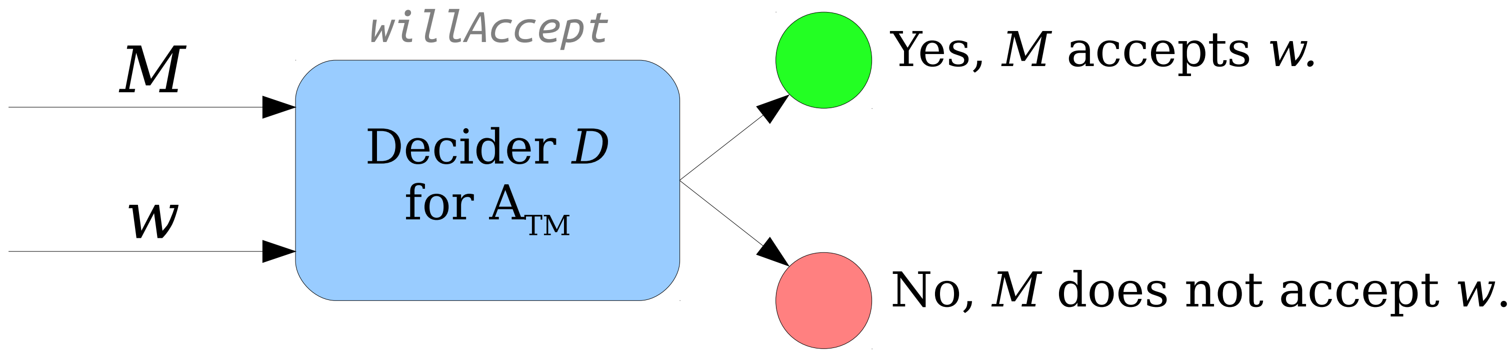


$A_{TM} \in \mathbf{R}$

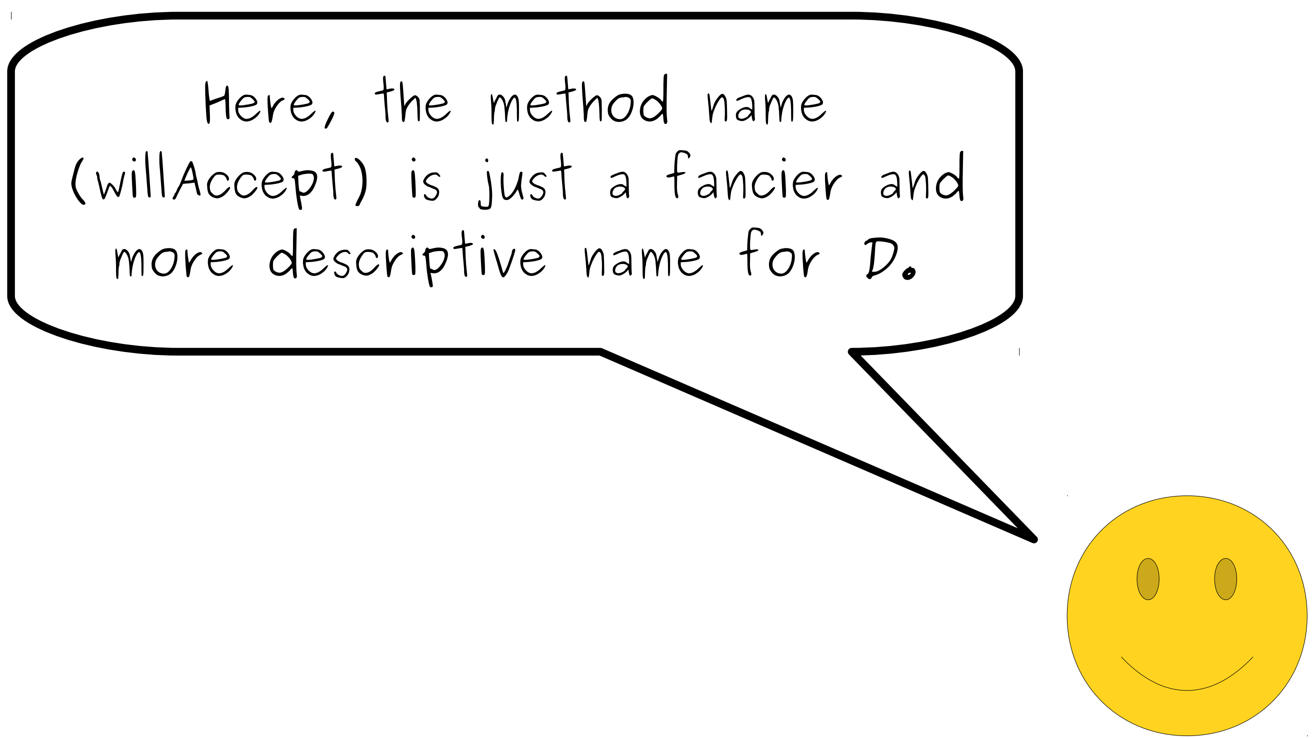
There is a decider D for A_{TM}

We can write programs that use D as a helper method

Contradiction!




```
bool willAccept(string program, string input)
```



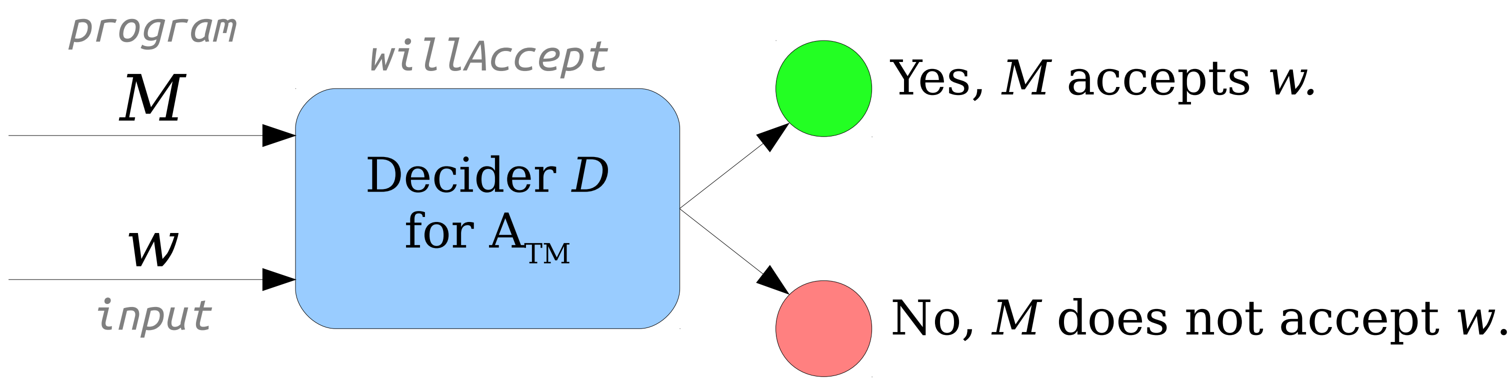
$A_{TM} \in \mathbf{R}$

↓

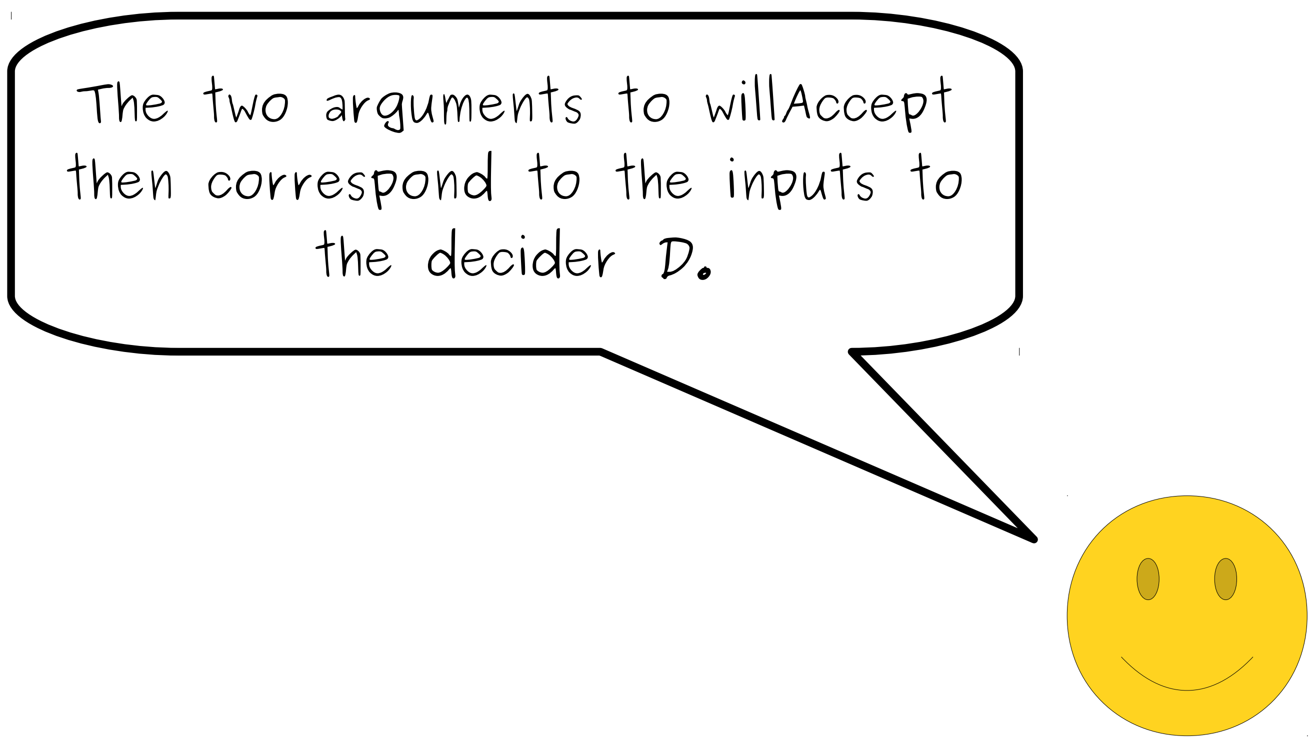
There is a decider *D* for A_{TM}

↓

We can write programs that use *D* as a helper method




```
bool willAccept(string program, string input)
```



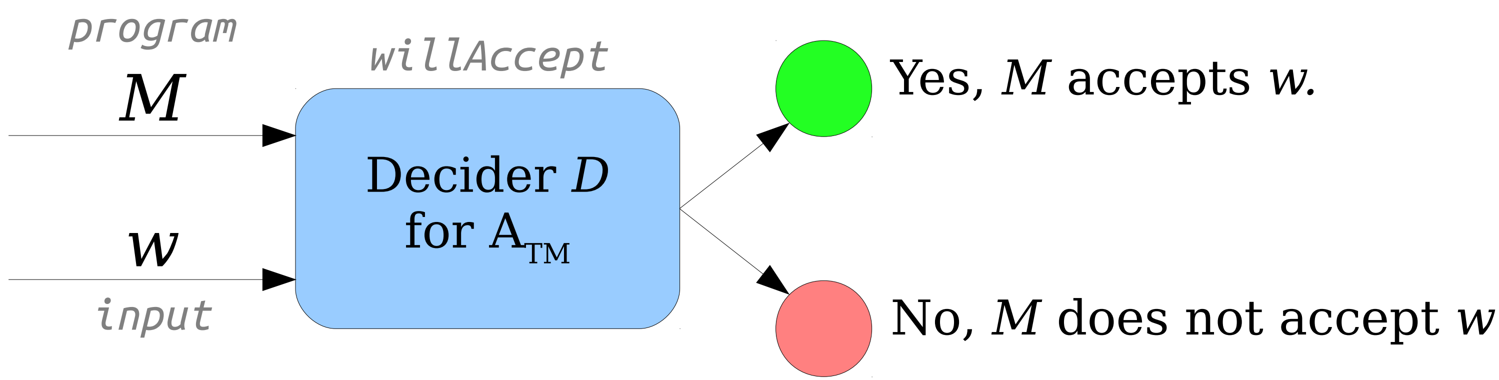
Contradiction!

$A_{TM} \in \mathbf{R}$

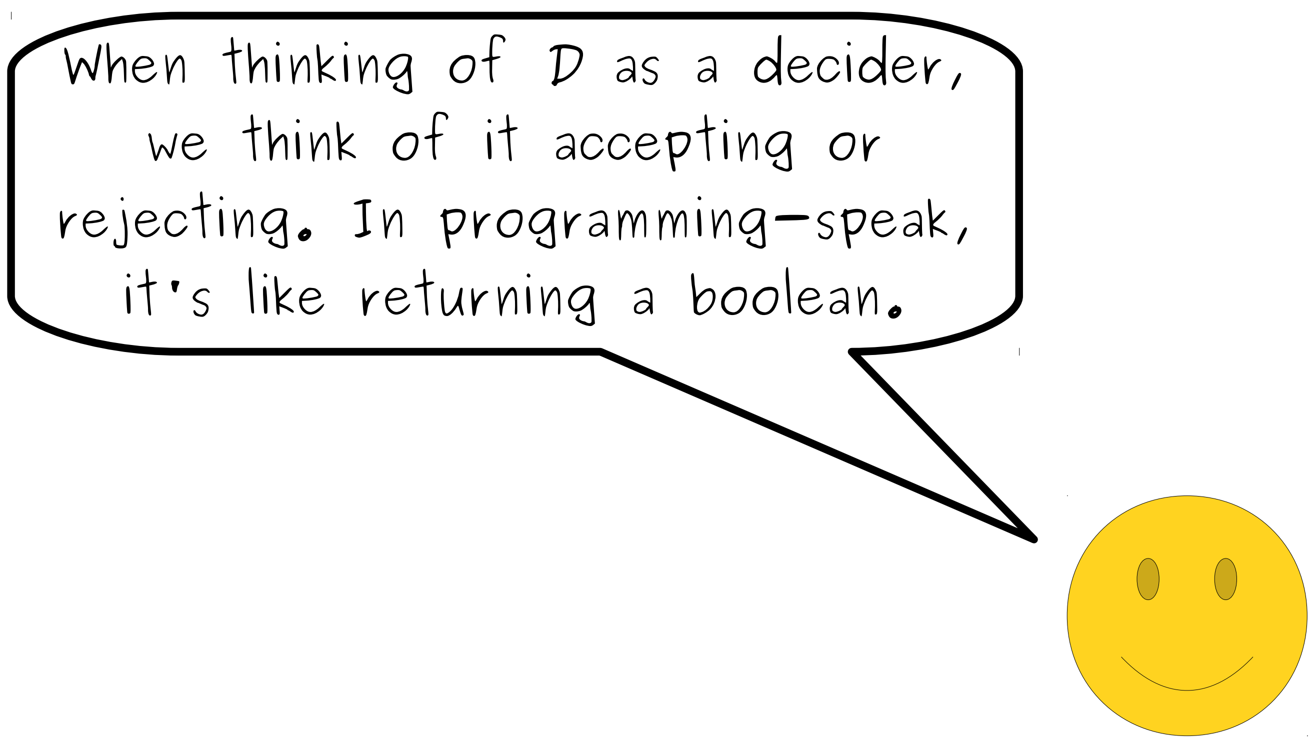
There is a decider D for A_{TM}

We can write programs that use D as a helper method

Contradiction!




```
bool willAccept(string program, string input)
```



$A_{TM} \in \mathbf{R}$

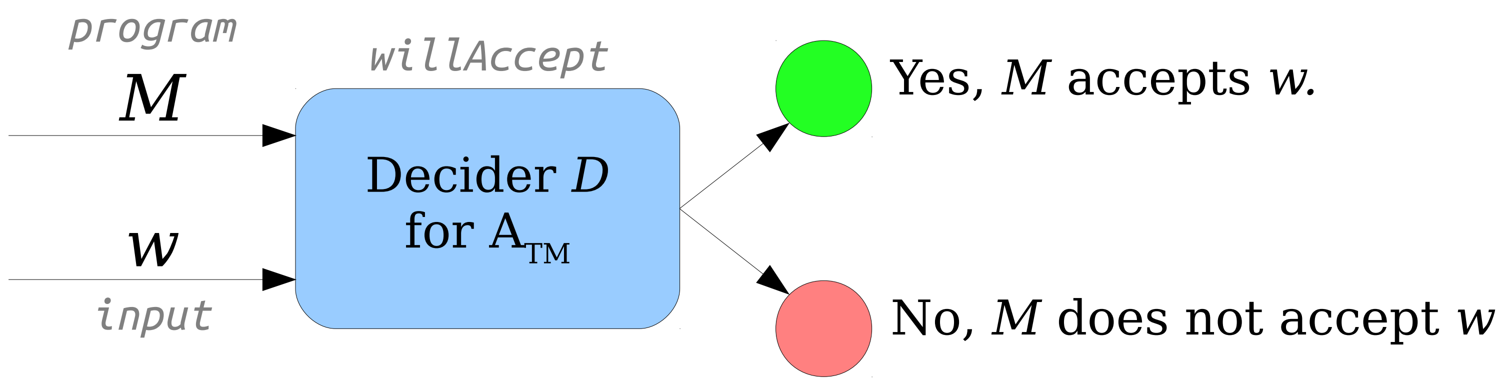
↓

There is a decider *D* for A_{TM}

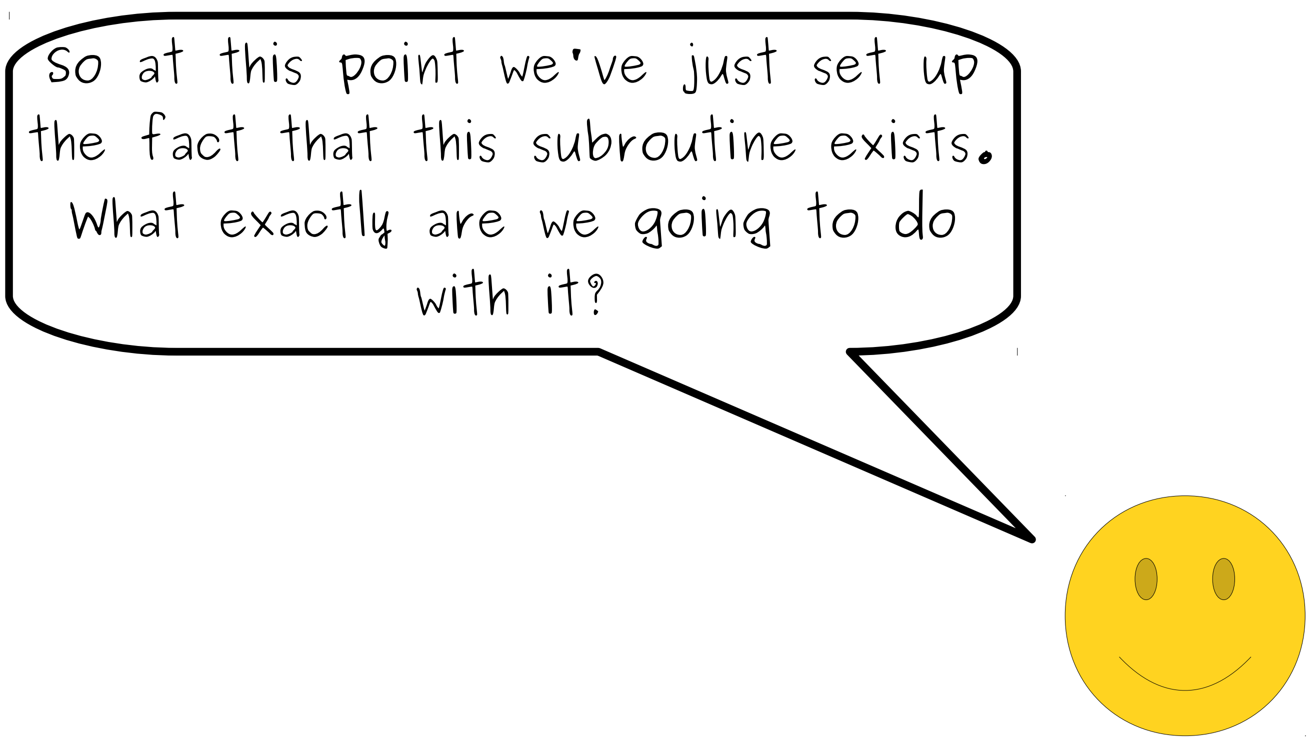
↓

We can write programs that use *D* as a helper method

Contradiction!




```
bool willAccept(string program, string input)
```

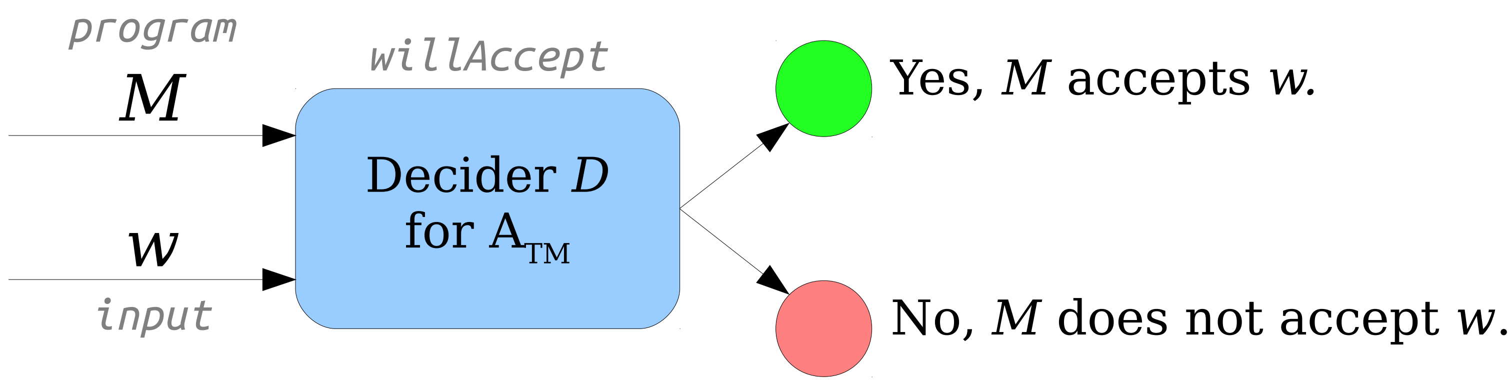


$A_{TM} \in \mathbf{R}$

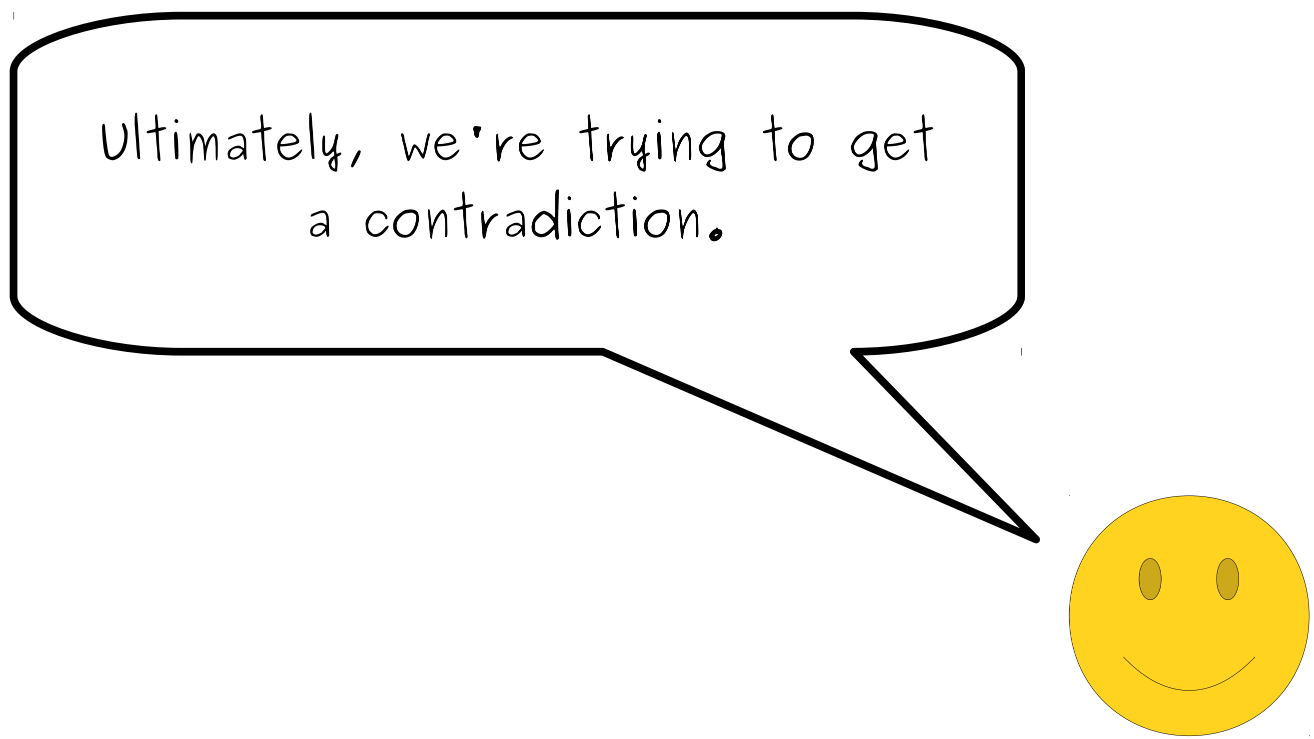
There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Contradiction!




```
bool willAccept(string program, string input)
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

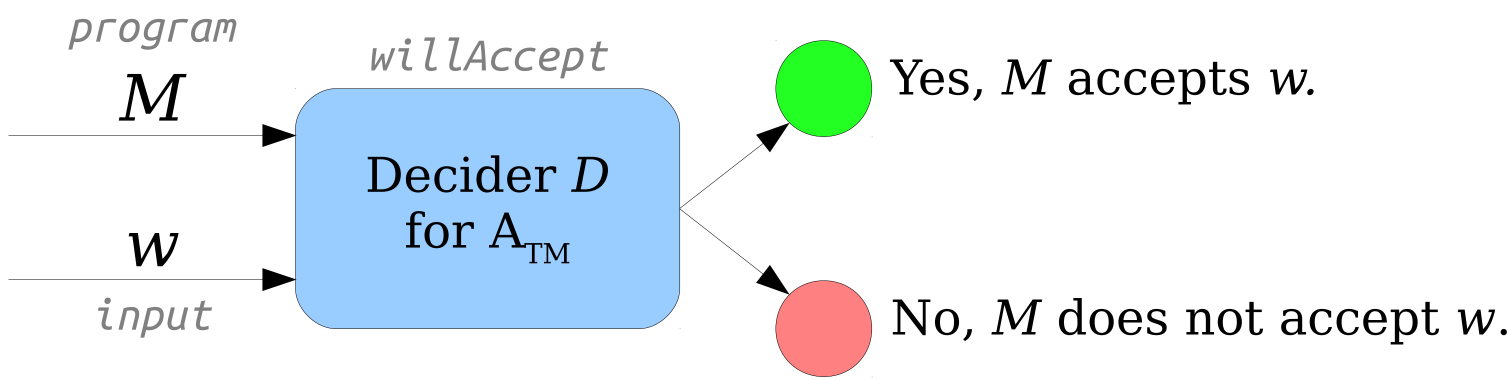
↓

We can write programs that use *D* as a helper method

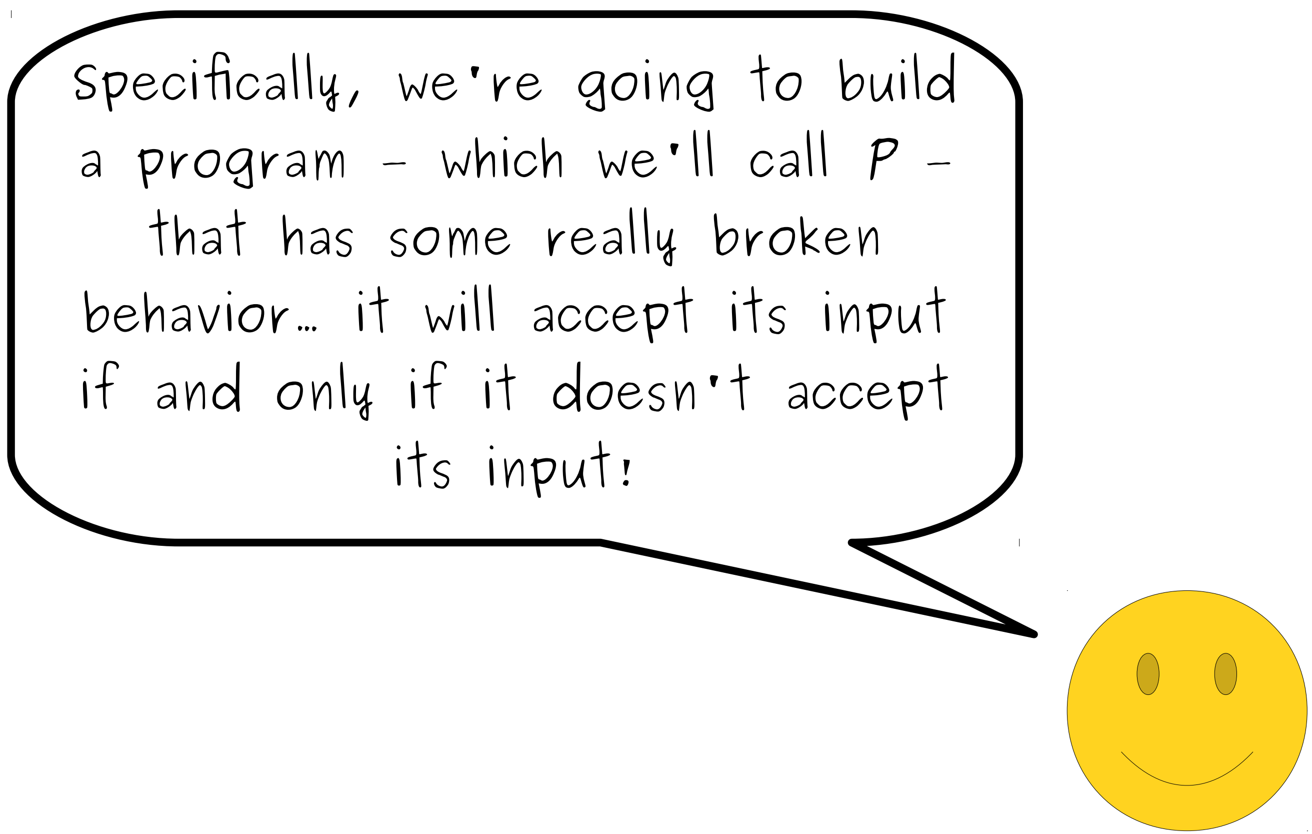
↓

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!




```
bool willAccept(string program, string input)
```



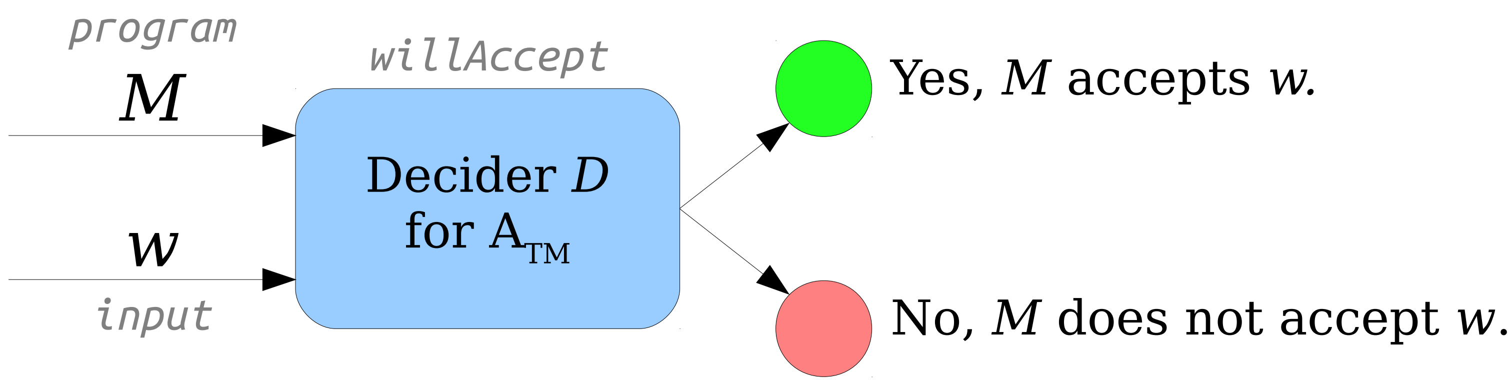
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

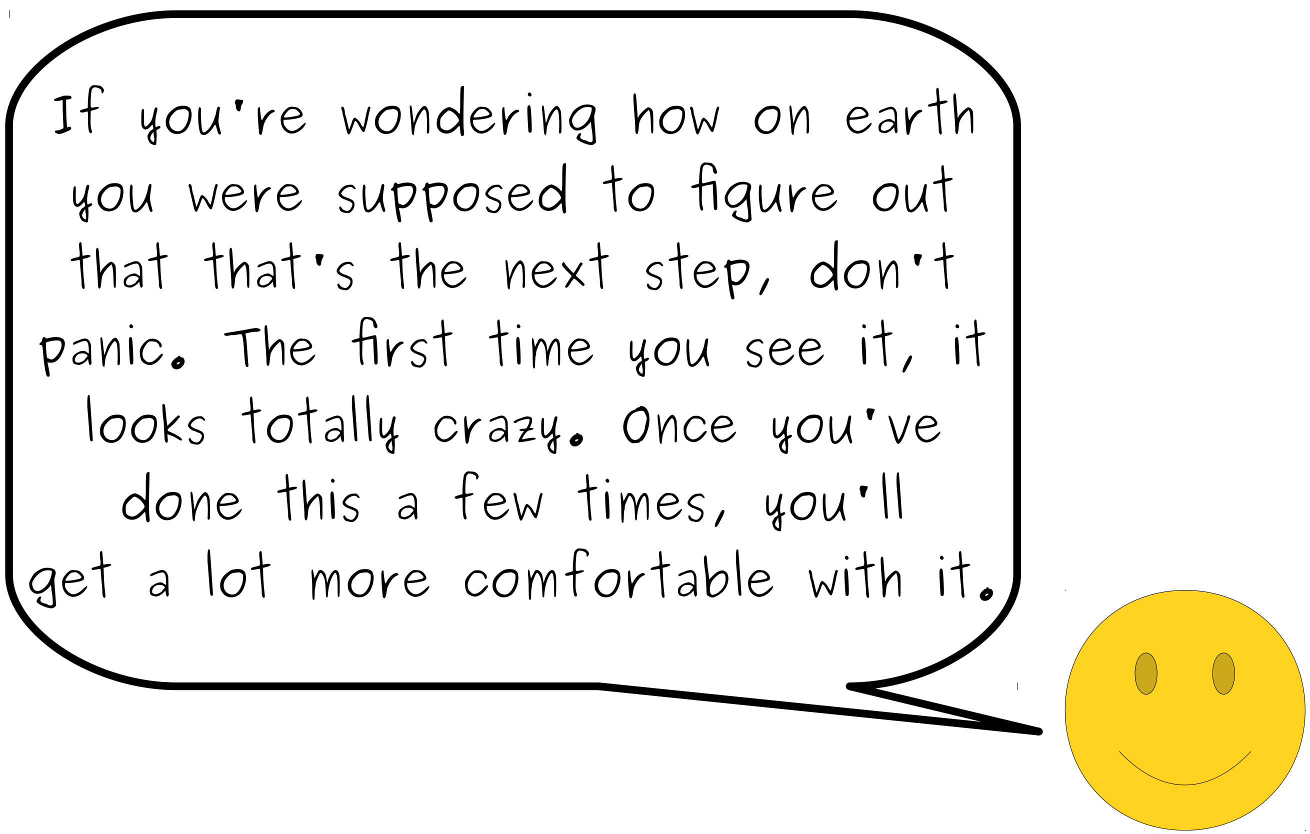
We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!




```
bool willAccept(string program, string input)
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

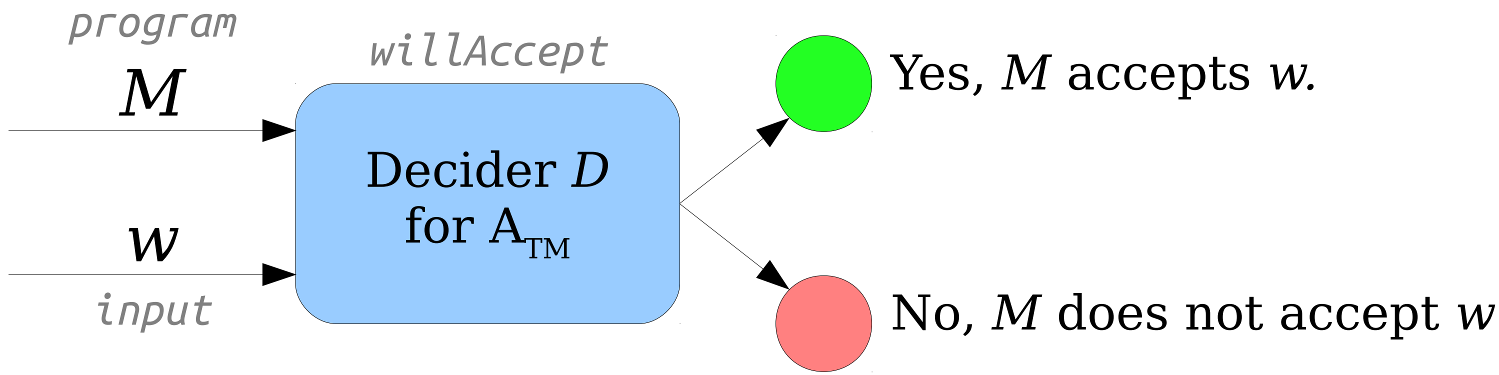
↓

We can write programs that use D as a helper method

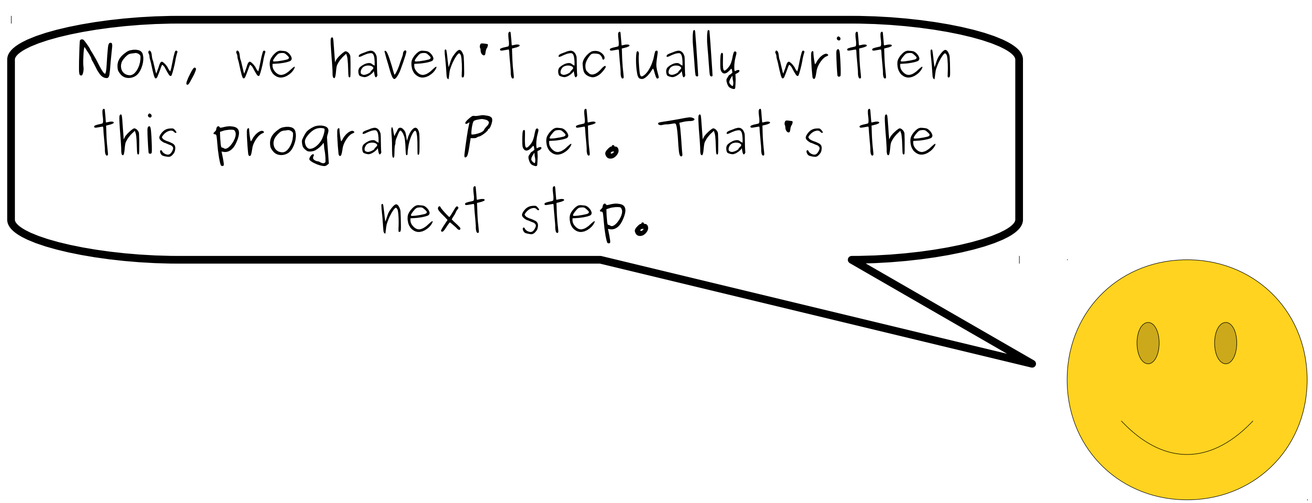
↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!




```
bool willAccept(string program, string input)
```



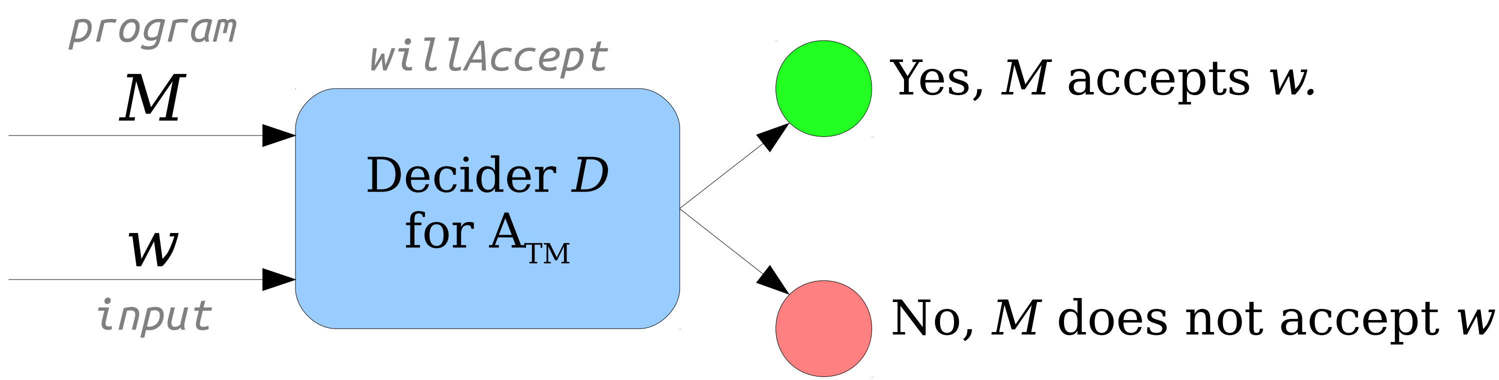
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

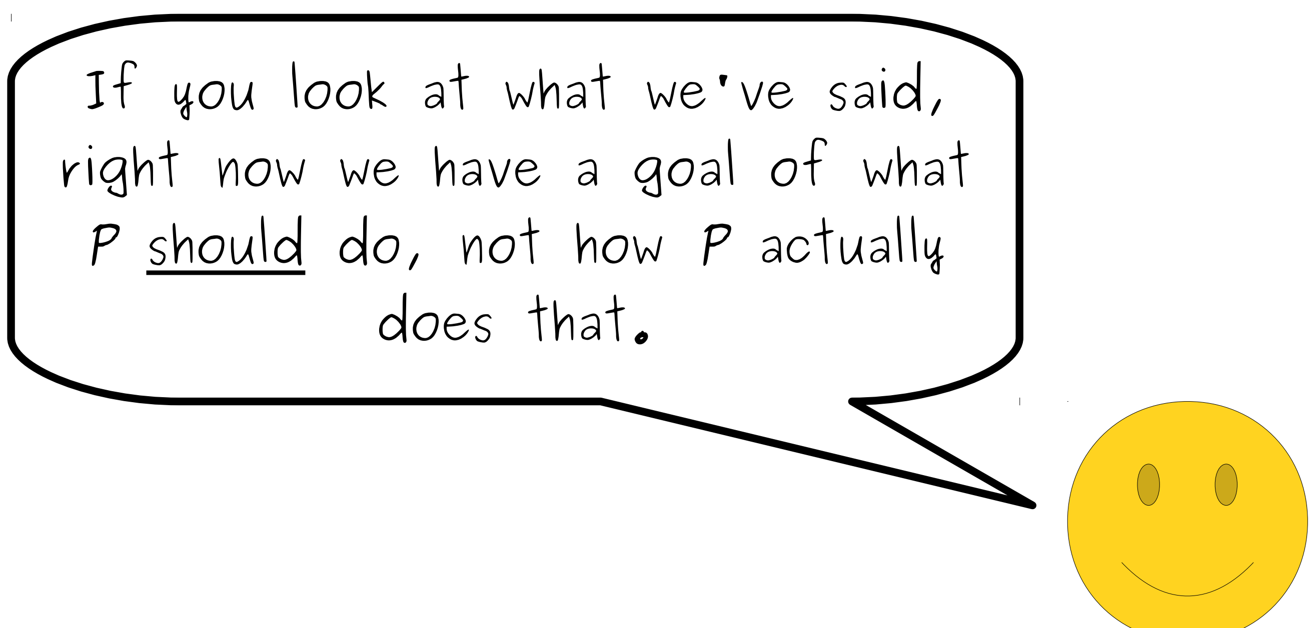
We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!




```
bool willAccept(string program, string input)
```



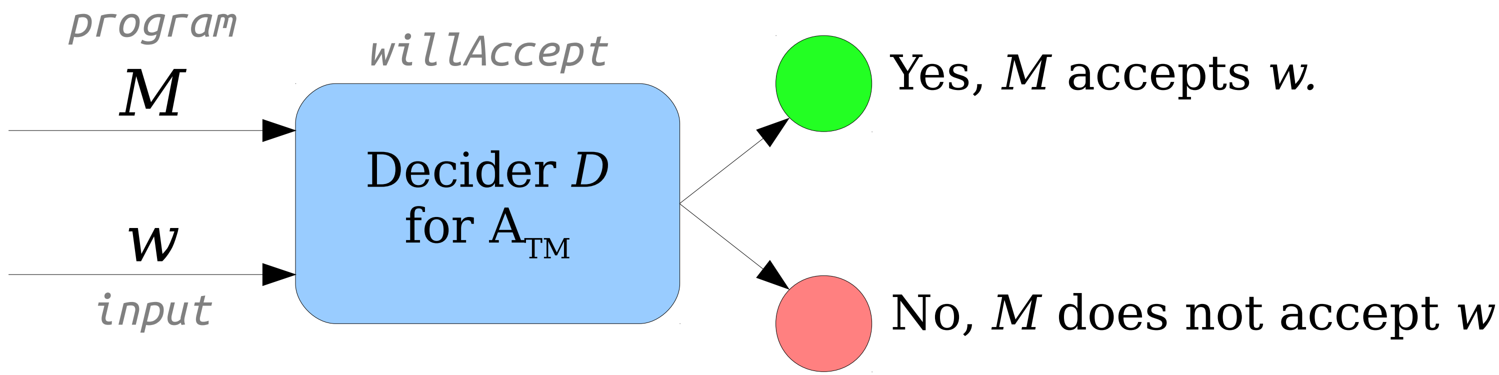
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

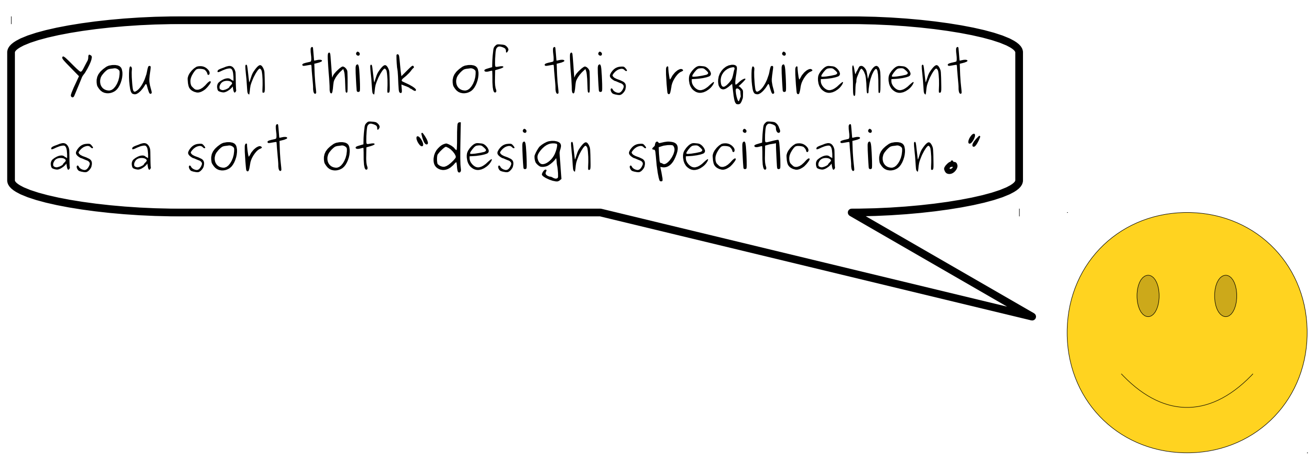
We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!




```
bool willAccept(string program, string input)
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

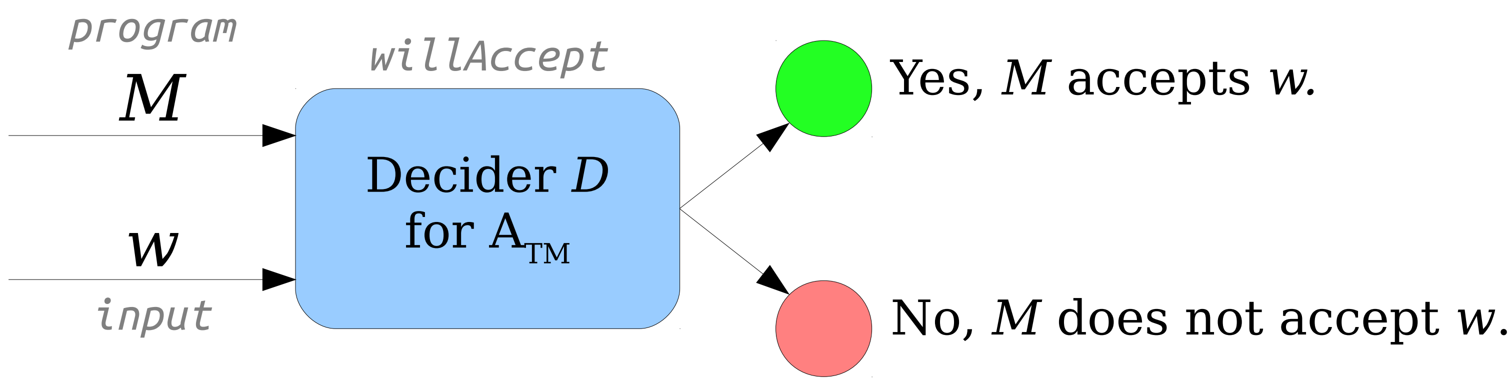
↓

We can write programs that use *D* as a helper method

↓

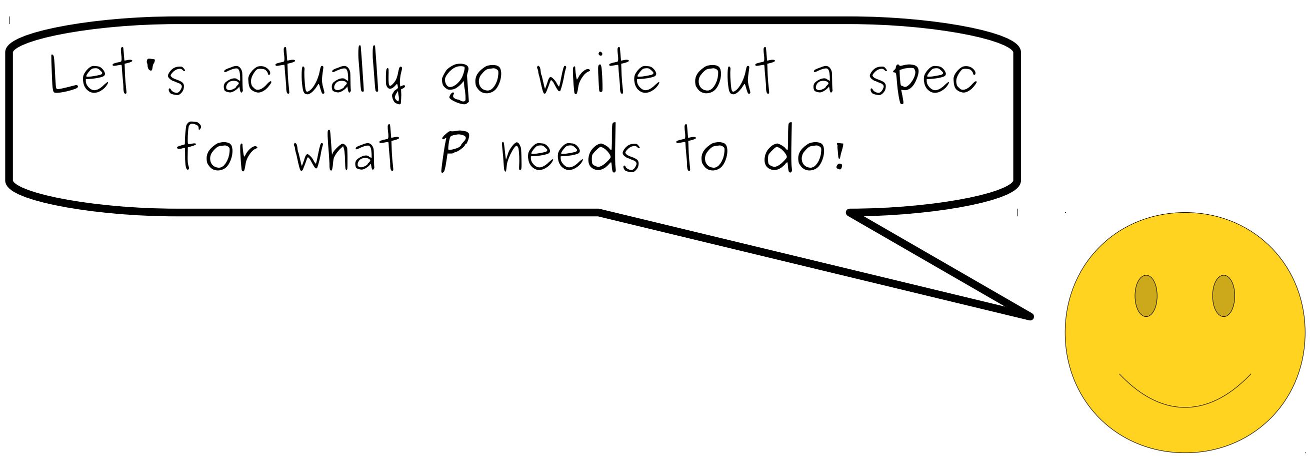
Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!




```
bool willAccept(string program, string input)
```

Program *P* design specification:



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

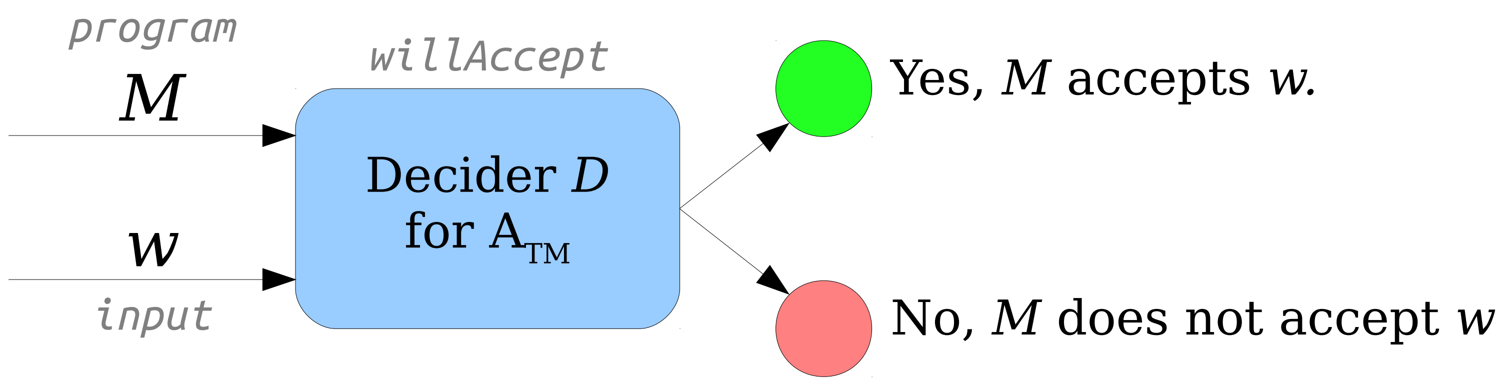
↓

We can write programs that use *D* as a helper method

↓

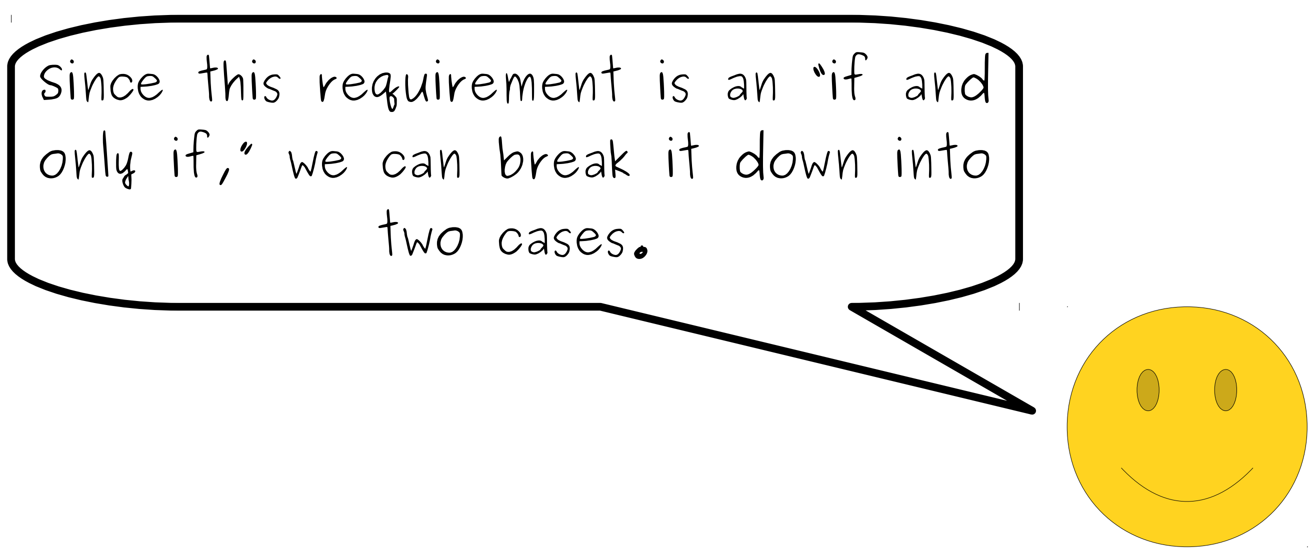
Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!




```
bool willAccept(string program, string input)
```

Program *P* design specification:



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

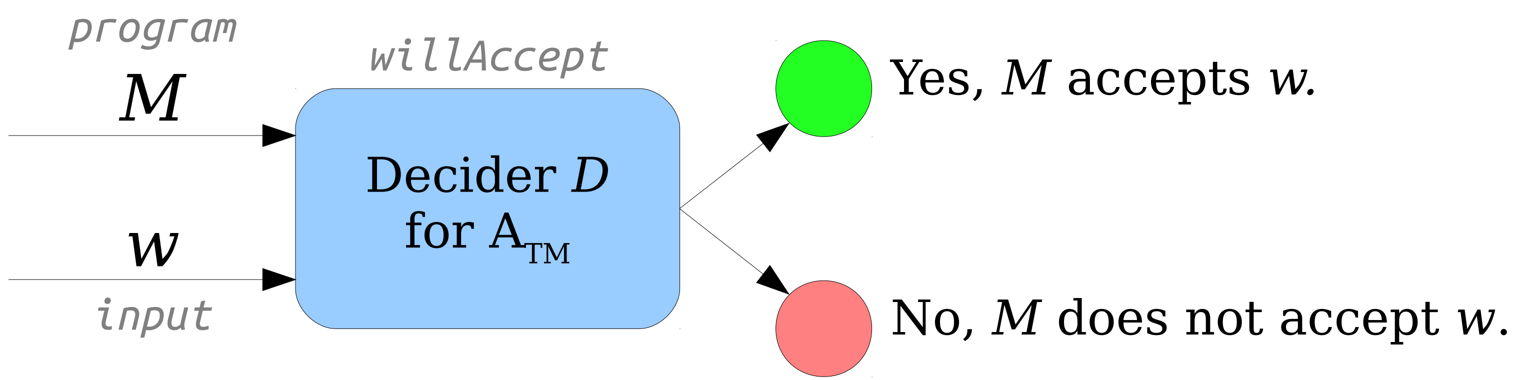
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

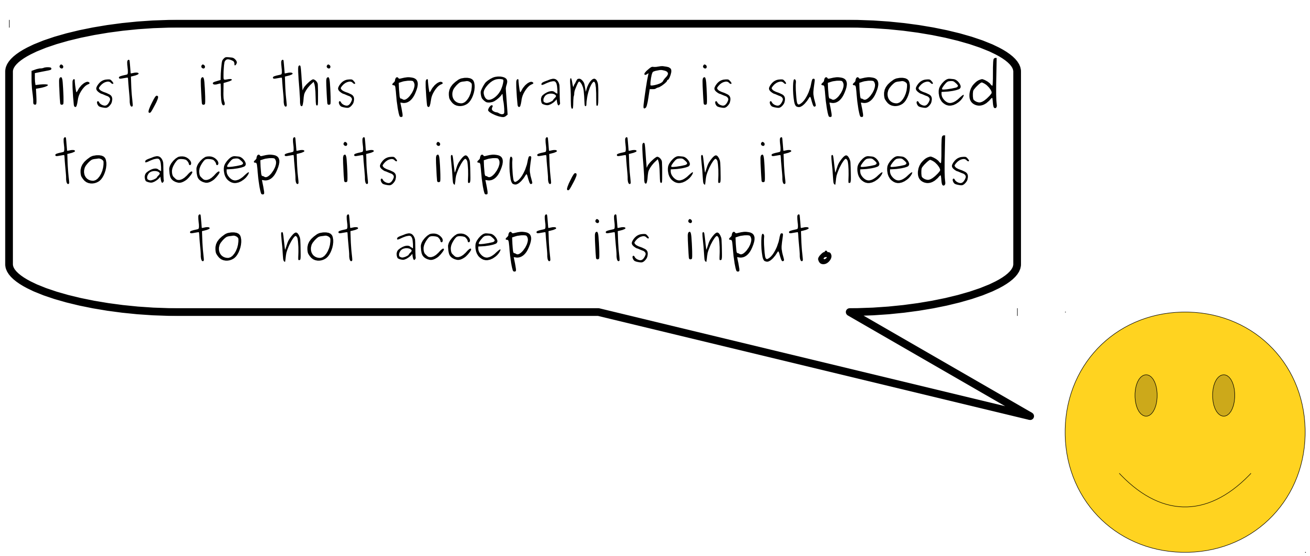
Contradiction!




```
bool willAccept(string program, string input)
```

Program P design specification:

If P accepts its input, then
P does not accept its input.



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

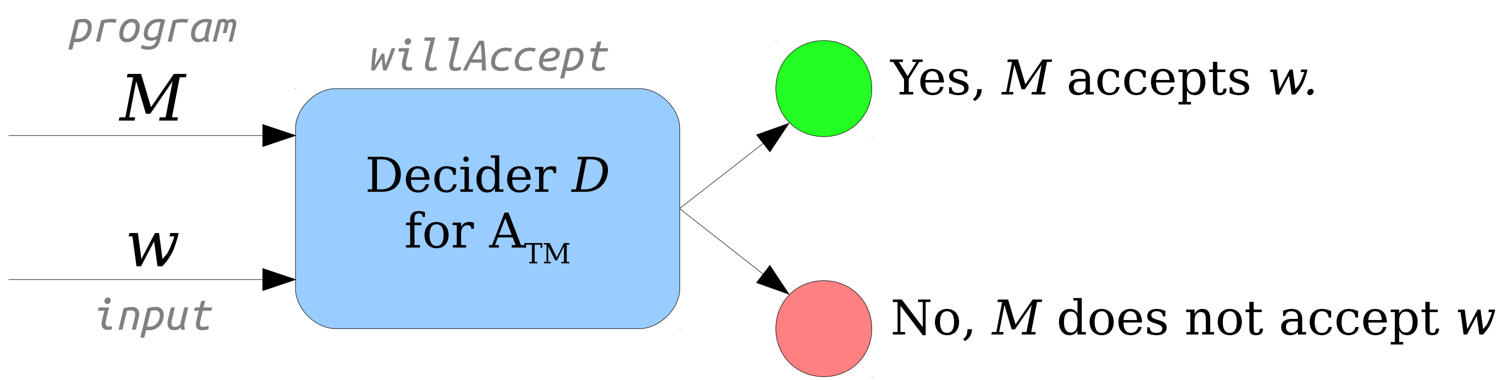
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

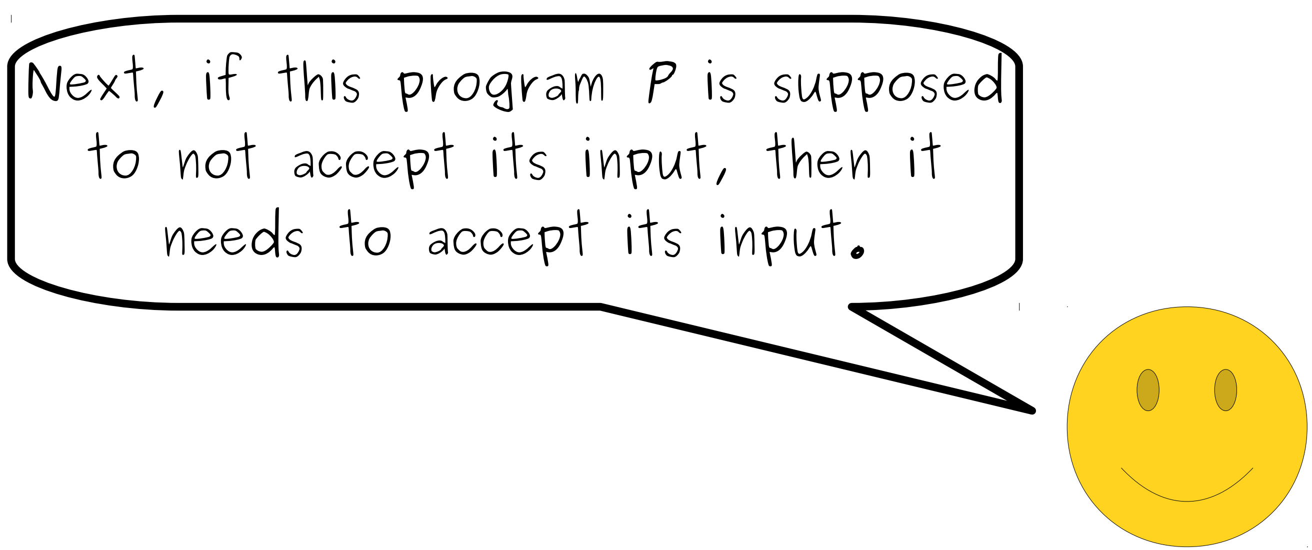
Contradiction!




```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

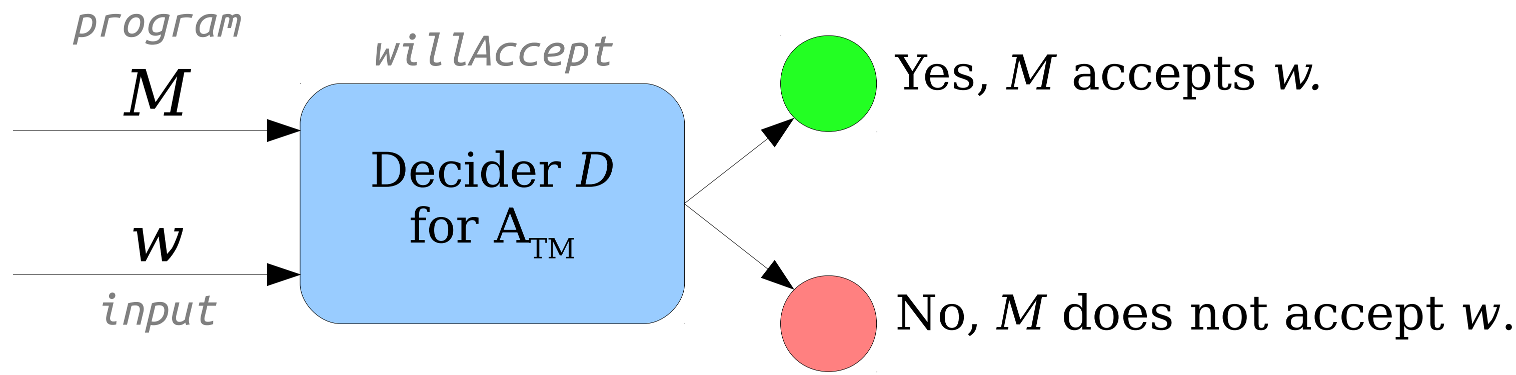
↓

We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

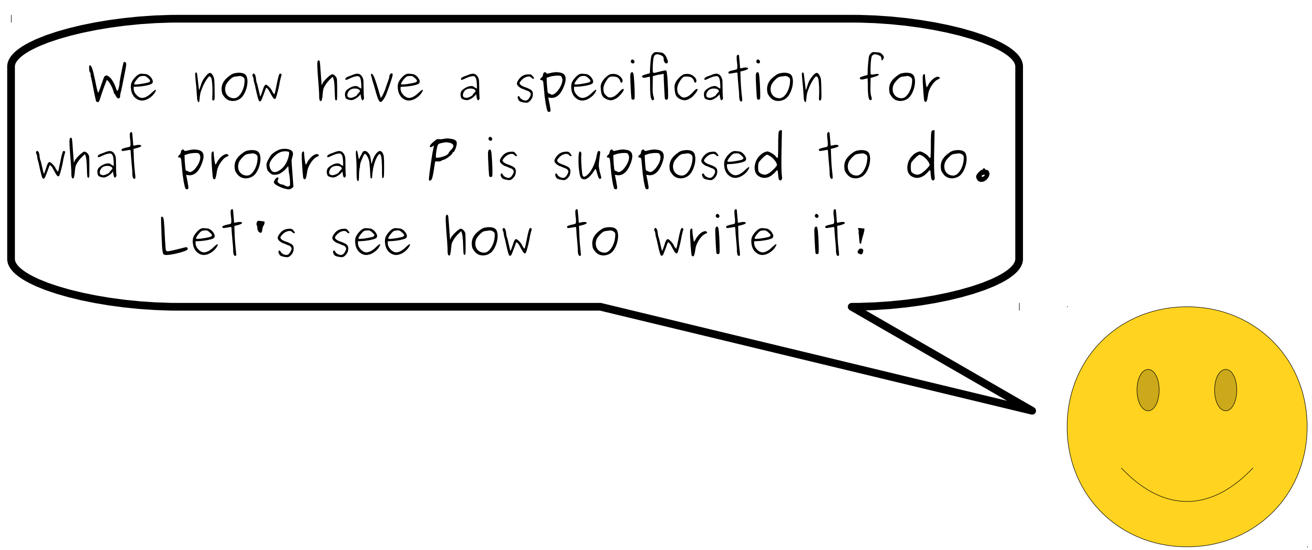
Contradiction!




```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.



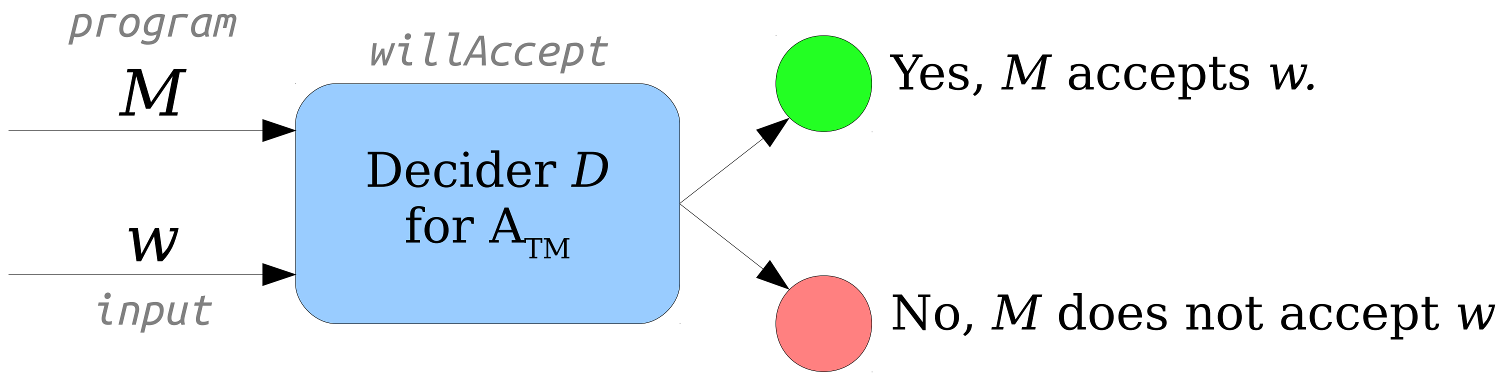
$A_{TM} \in$ **R**

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

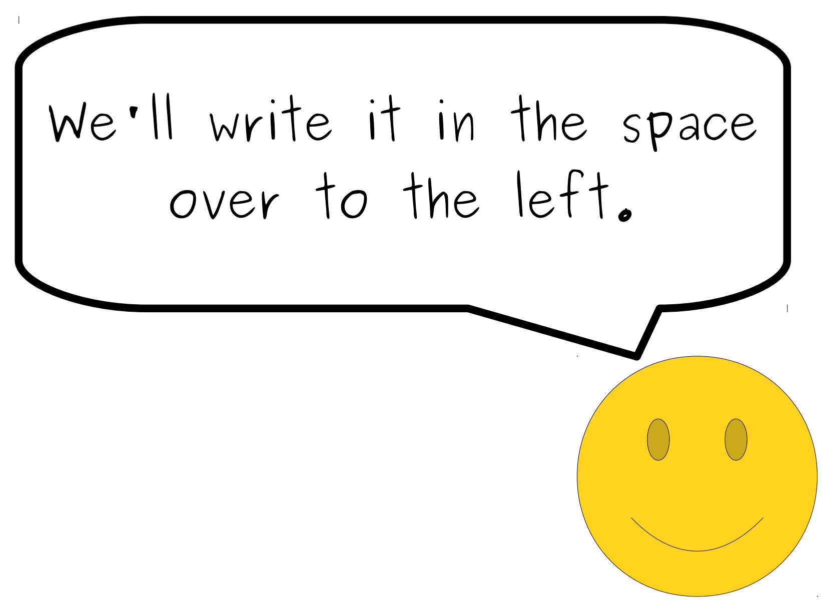



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

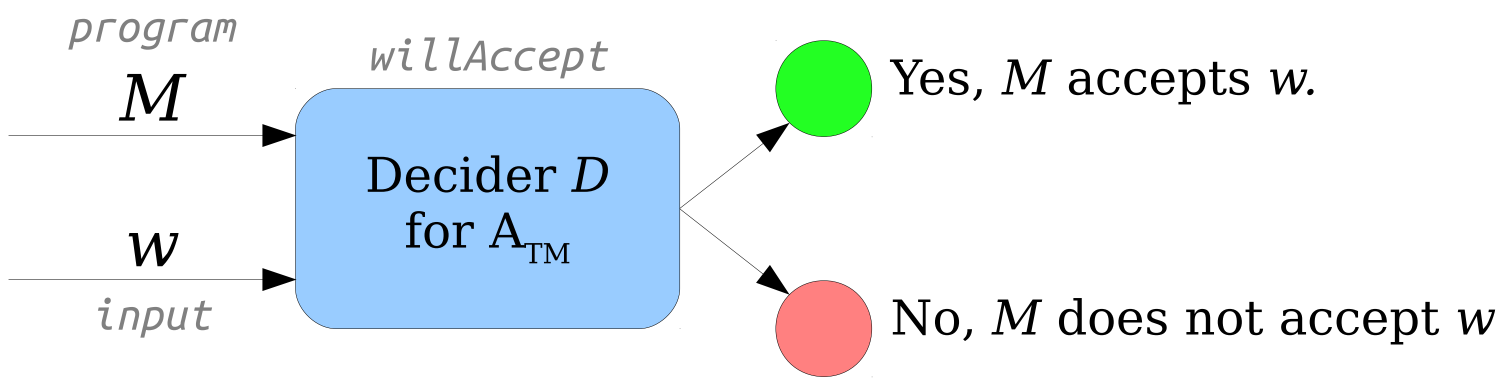
↓

We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

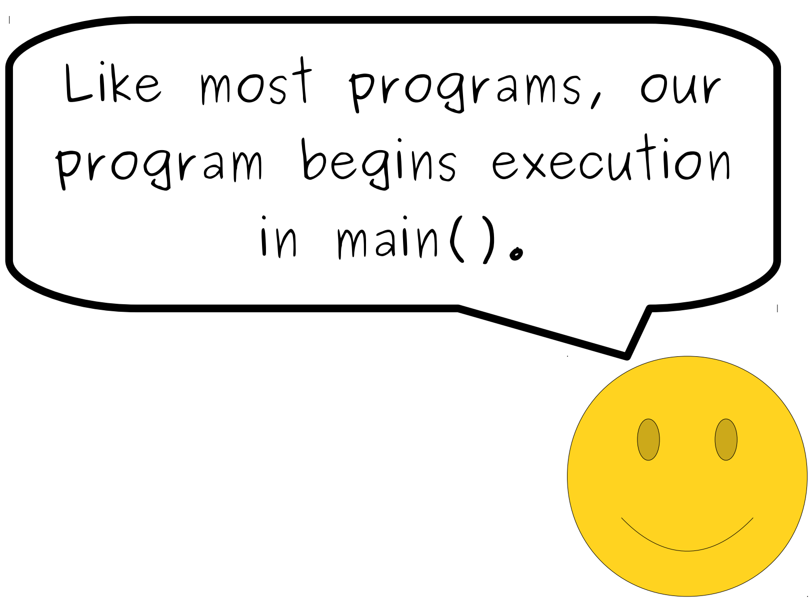



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {



}
```



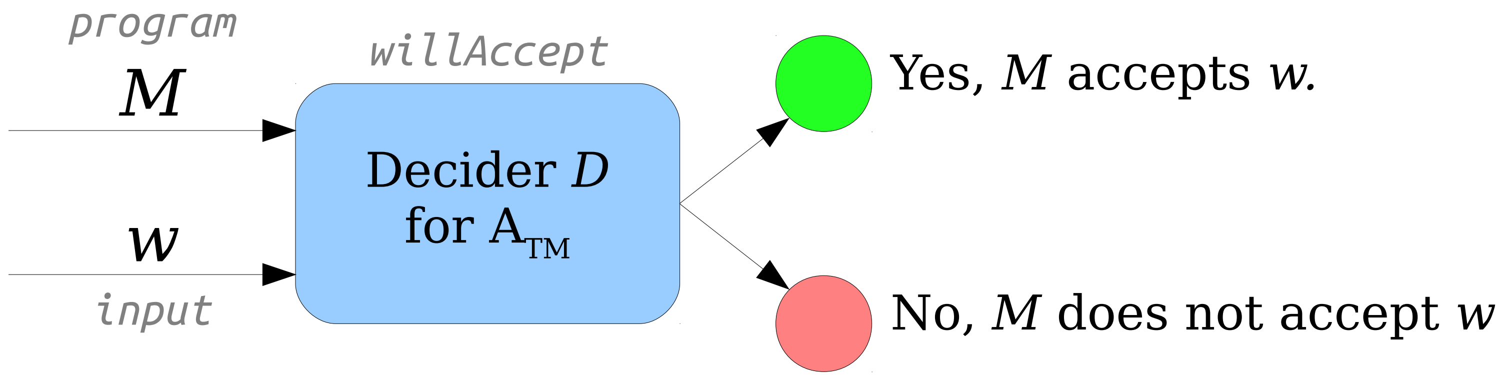
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!

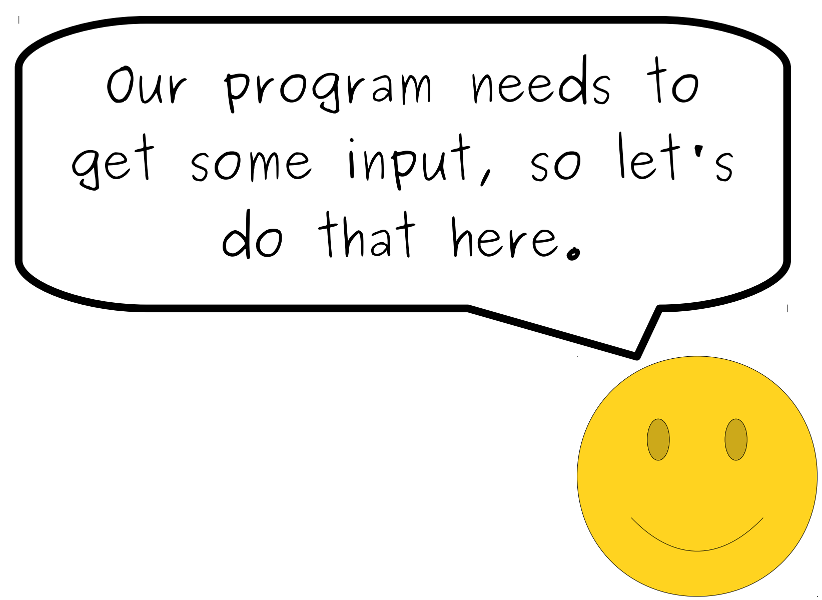



```
bool willAccept(string program, string input)
```

Program P design specification:

If P accepts its input, then
P does not accept its input.
If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();


}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

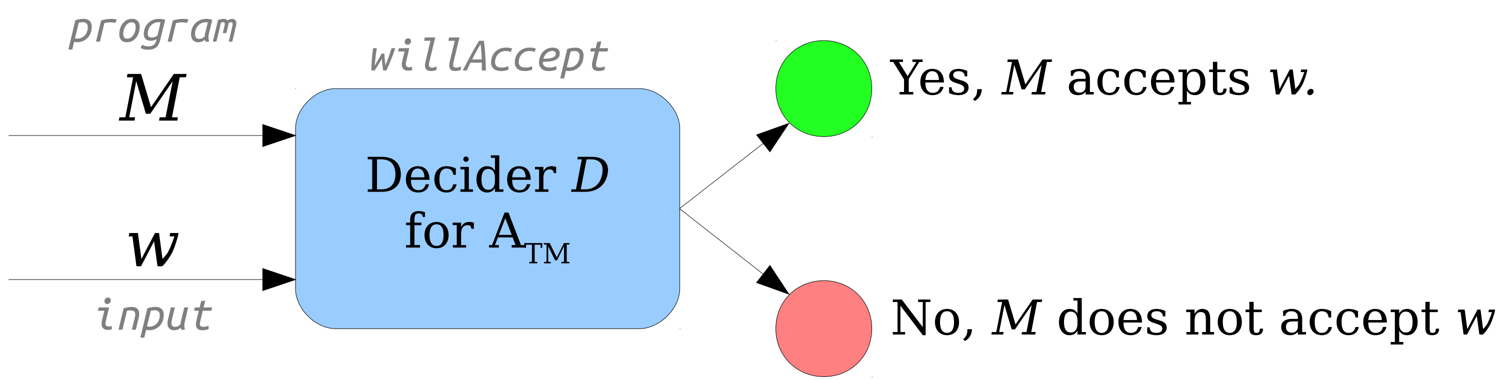
↓

We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

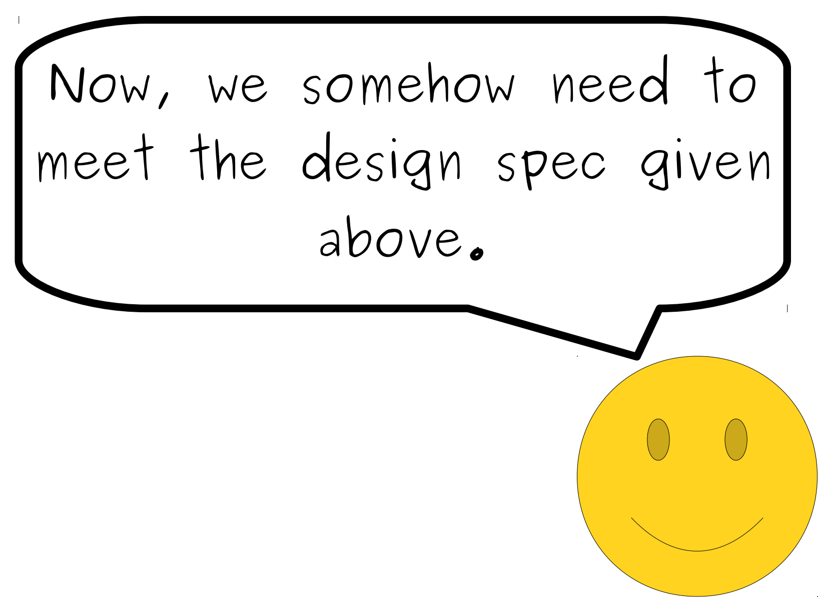



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();




}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

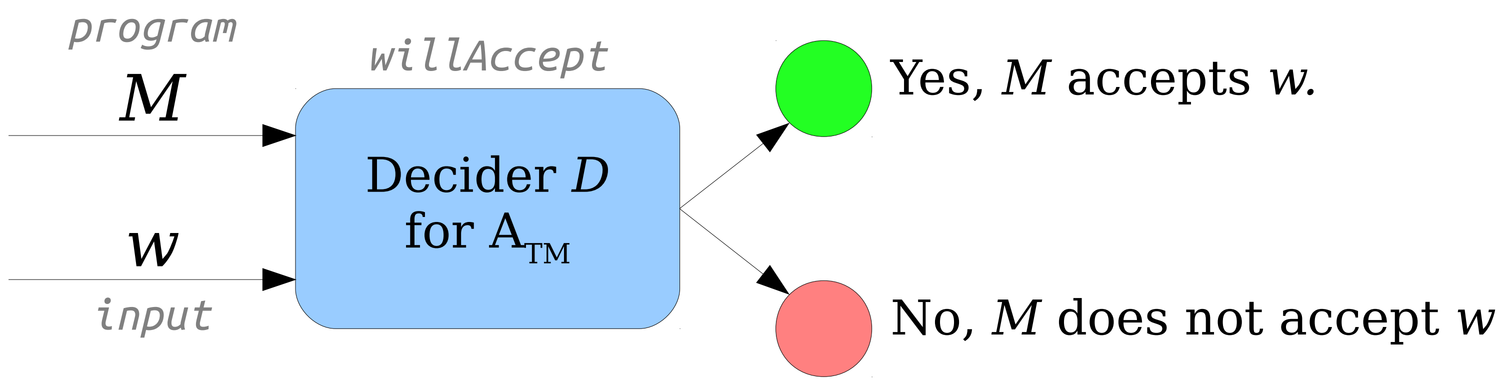
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!

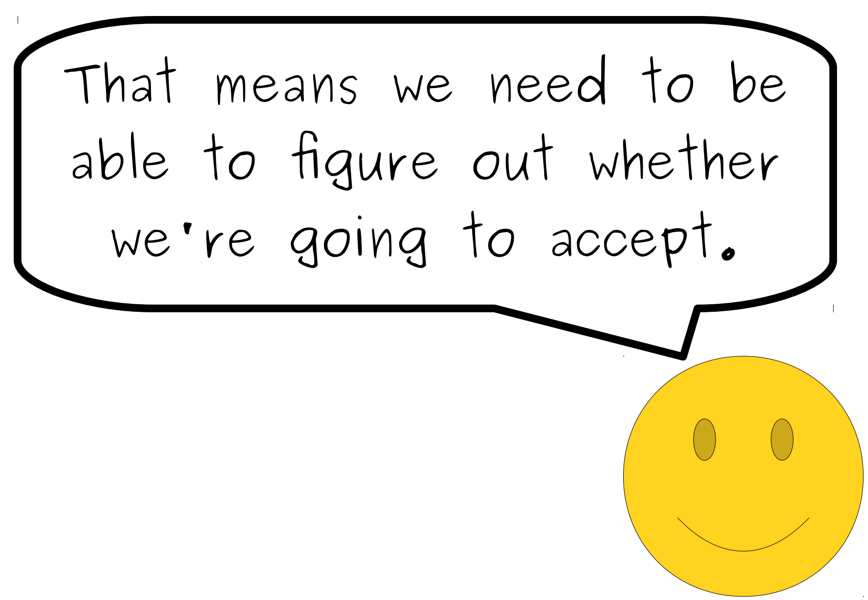



```
bool willAccept(string program, string input)
```

Program P design specification:

If P accepts its input, then
  P does not accept its input.
If P does not accept its input, then
  P accepts its input.

```
// Program P

int main() {
  string input = getInput();

}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

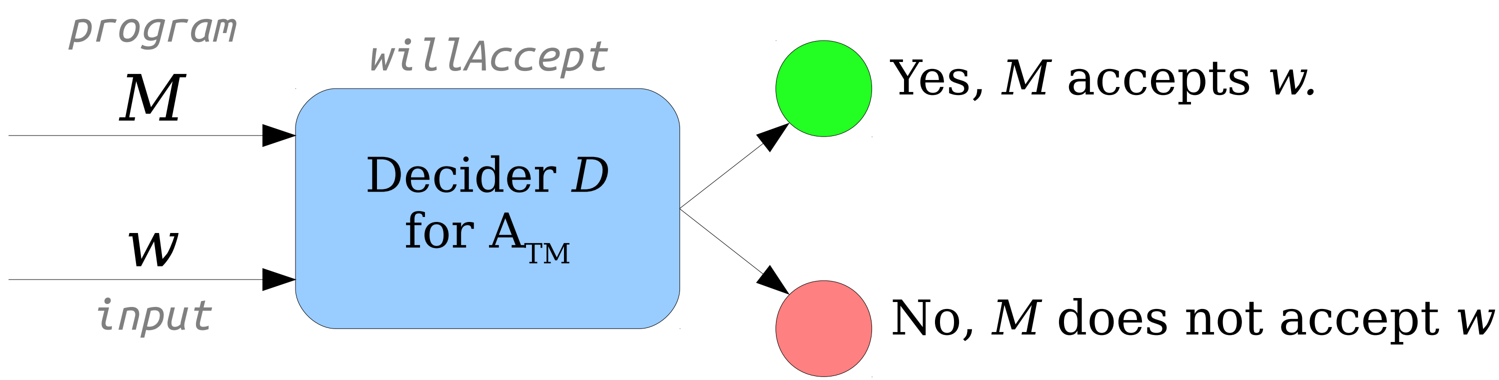
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!

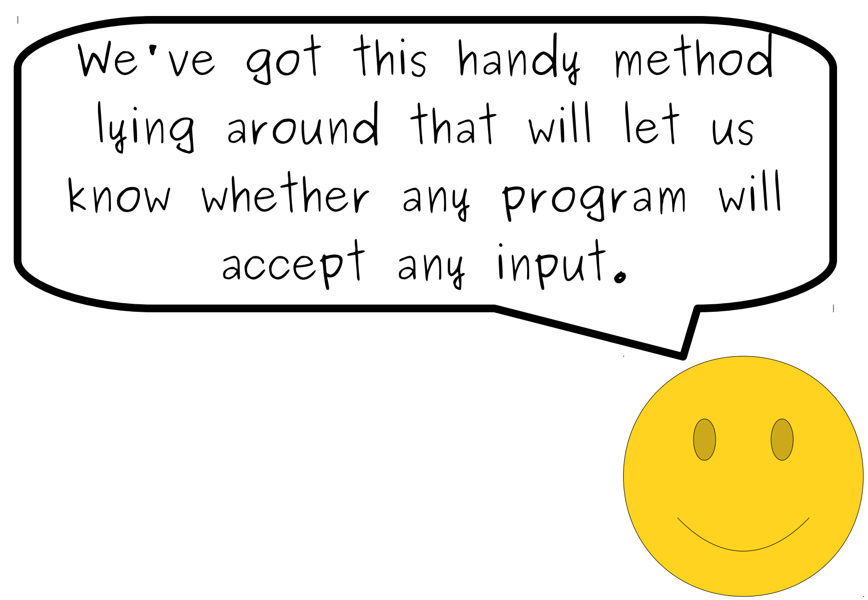



```
bool willAccept(string program, string input)
```

Program P design specification:

If P accepts its input, then
    P does not accept its input.
If P does not accept its input, then
    P accepts its input.

```
// Program P

int main() {
  string input = getInput();

}
```



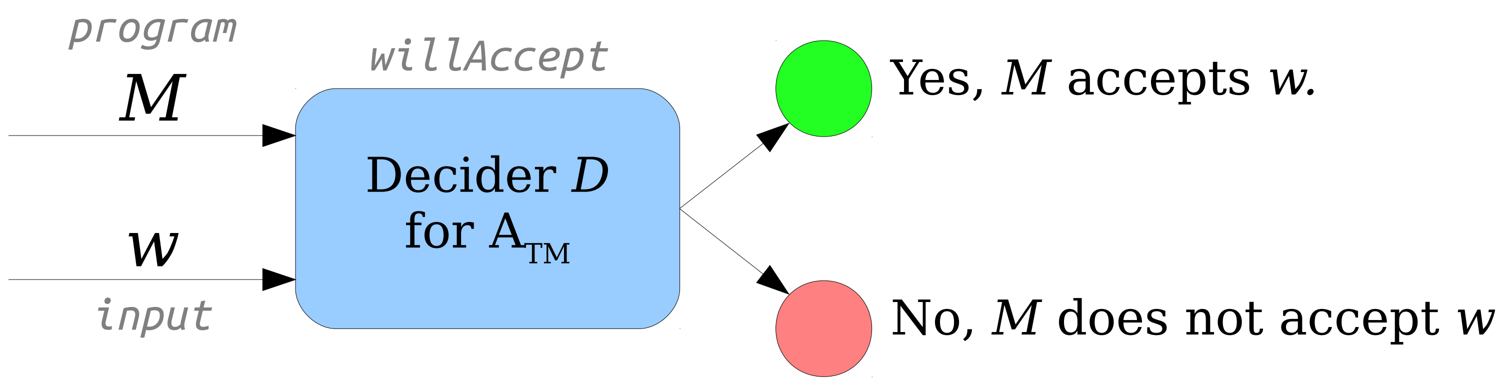
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

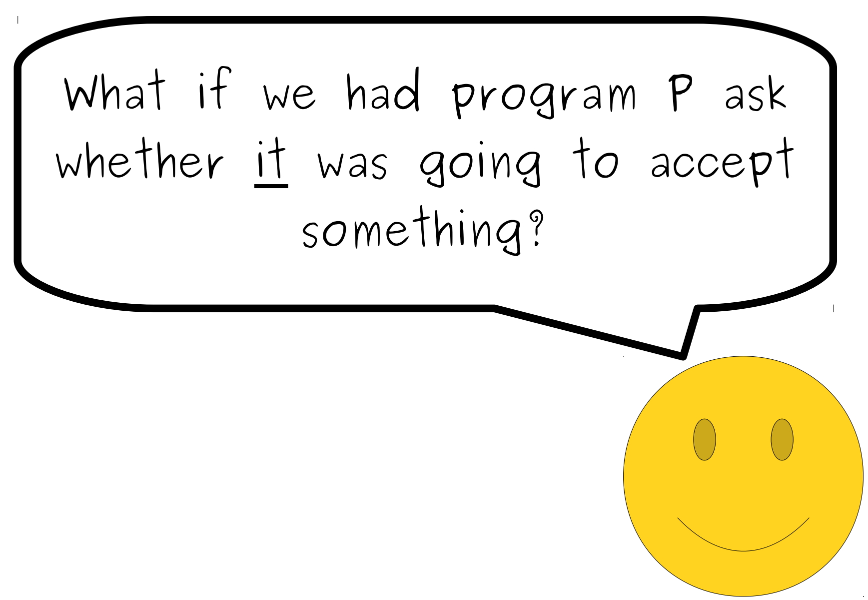



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();

}
```



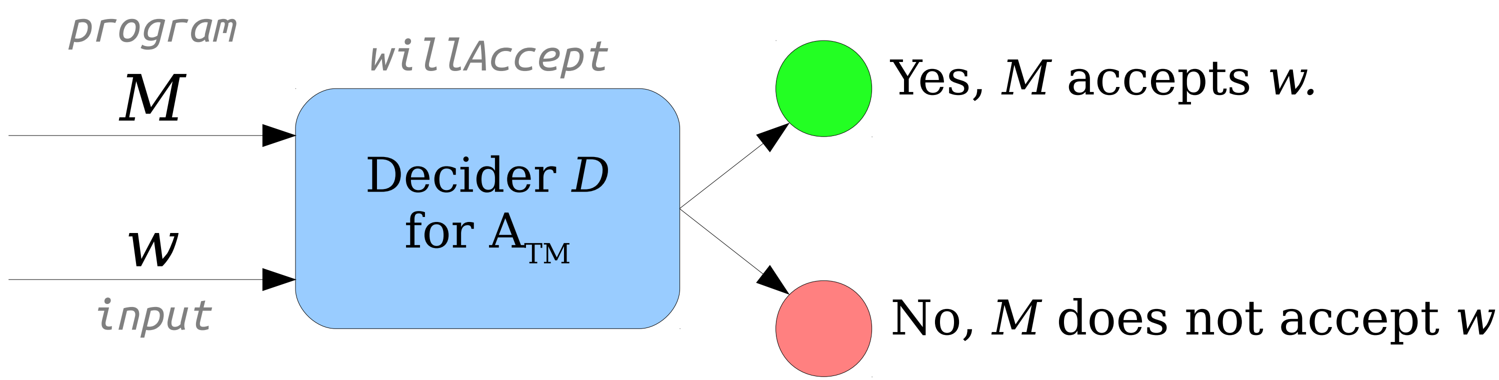
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

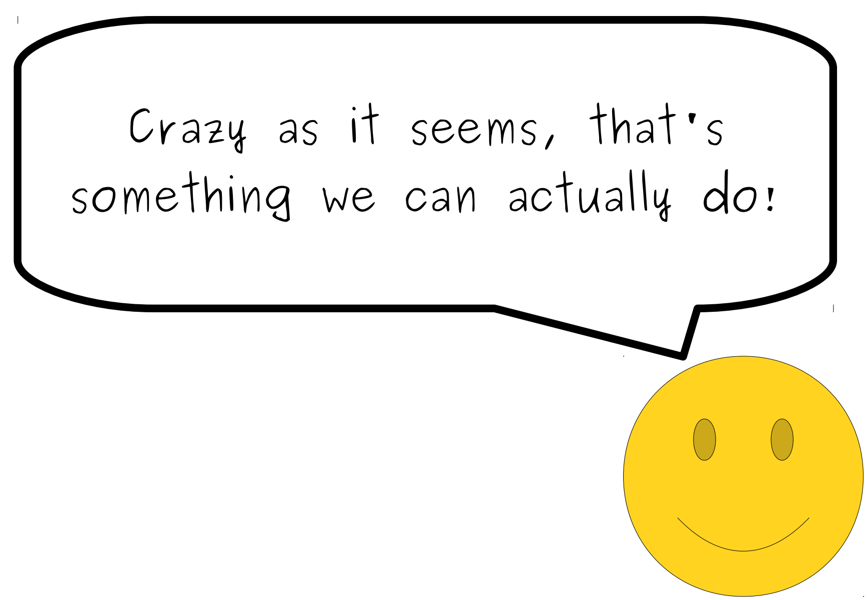



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();



}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

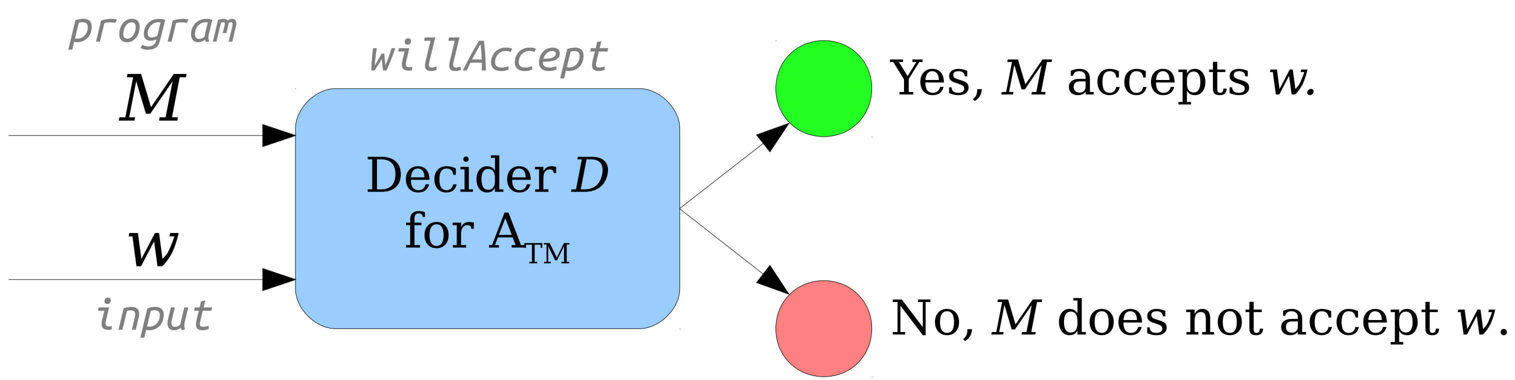
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!

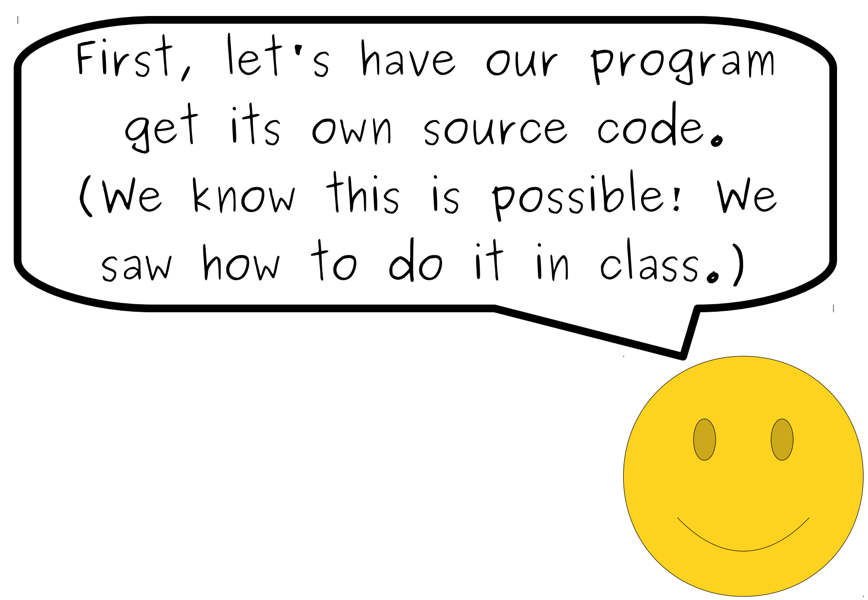



```
bool willAccept(string program, string input)
```

Program P design specification:

If P accepts its input, then
P does not accept its input.
If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();



}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

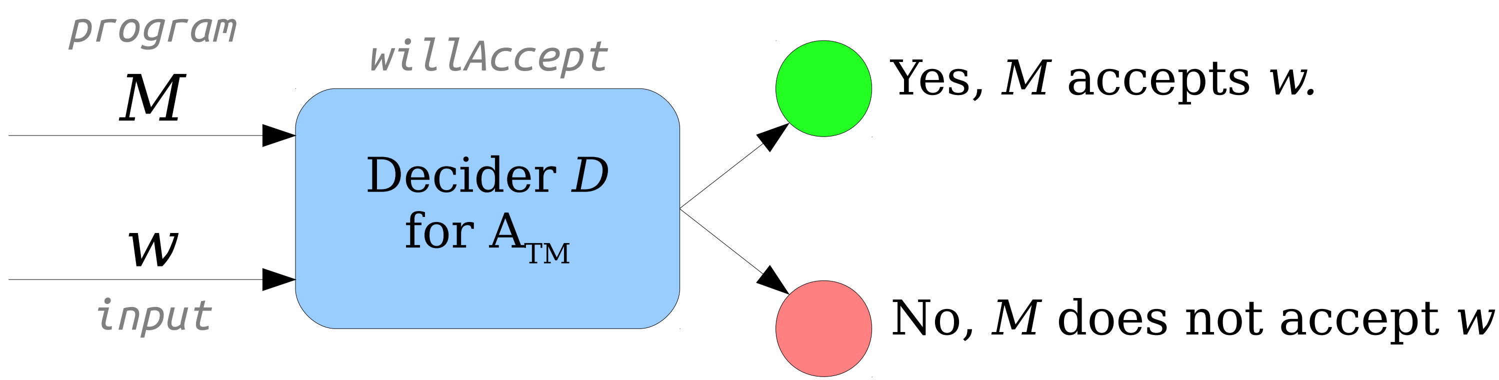
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!

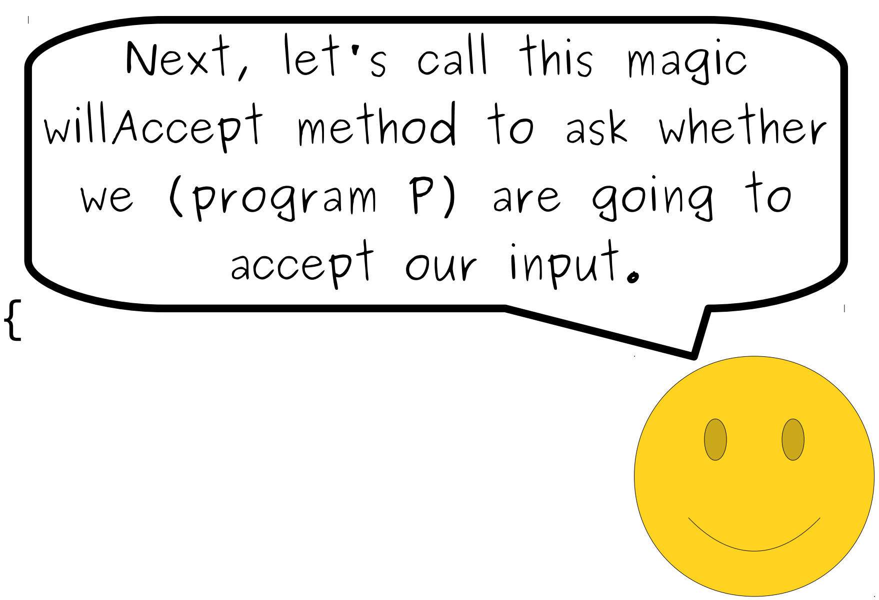



```
bool willAccept(string program, string input)
```

Program P design specification:

If P accepts its input, then
P does not accept its input.
If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {

  } else {

  }
}
```



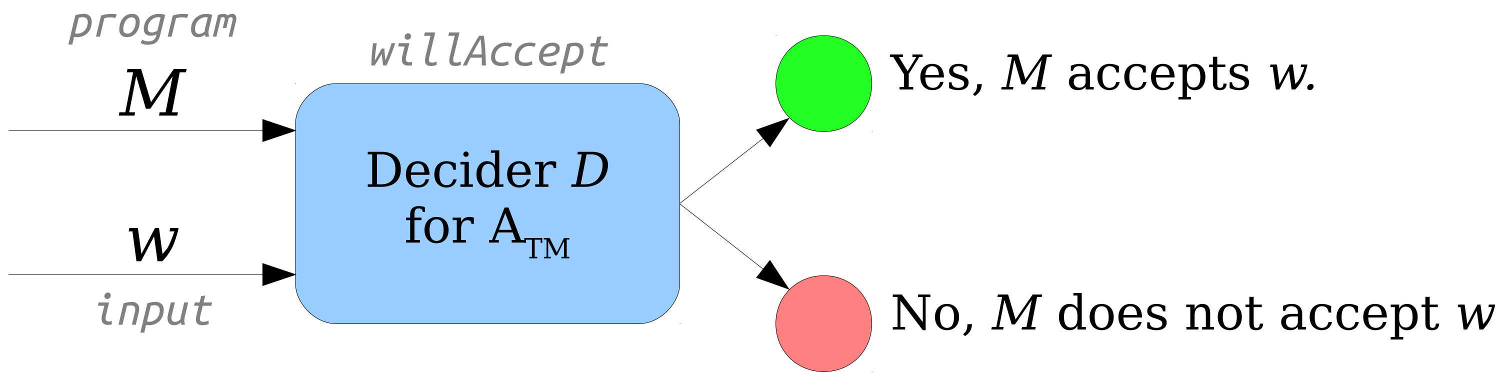
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!

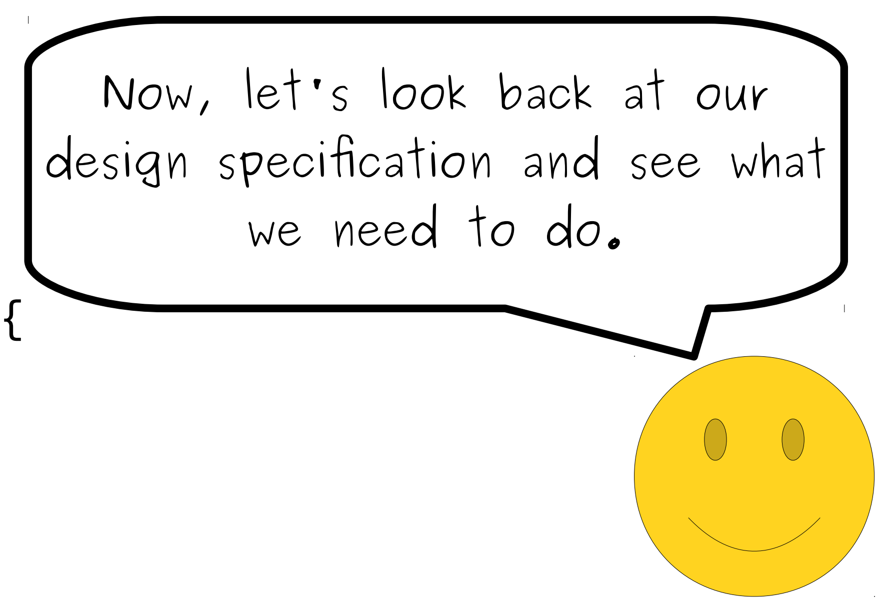



```
bool willAccept(string program, string input)
```

Program P design specification:

If P accepts its input, then
P does not accept its input.
If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {

  } else {

  }
}
```



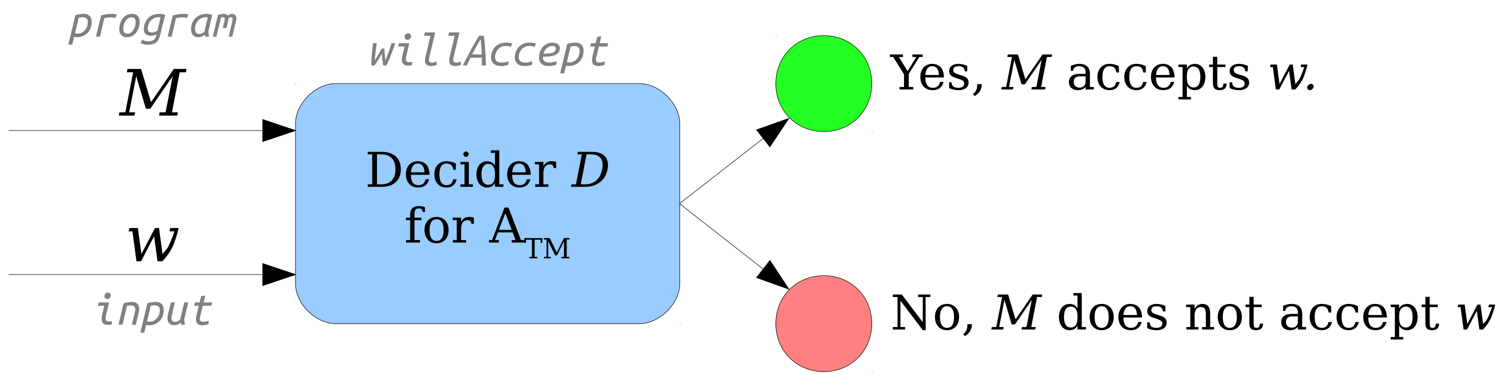
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

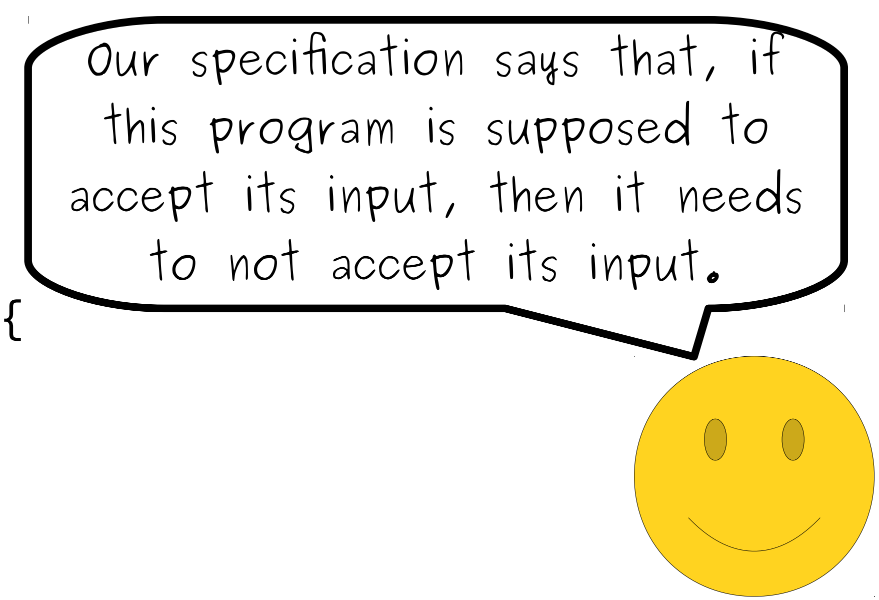



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {

  } else {

  }
}
```



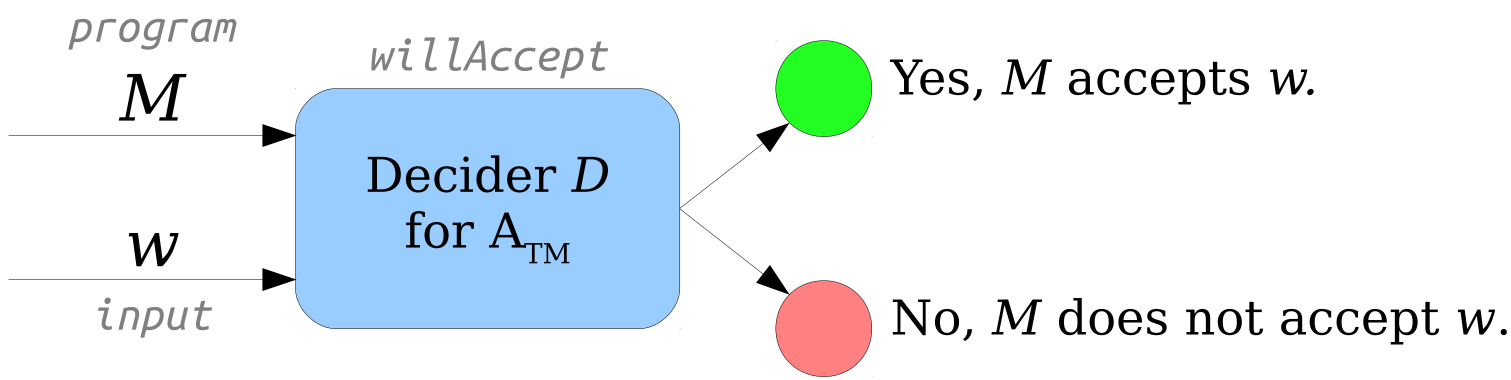
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

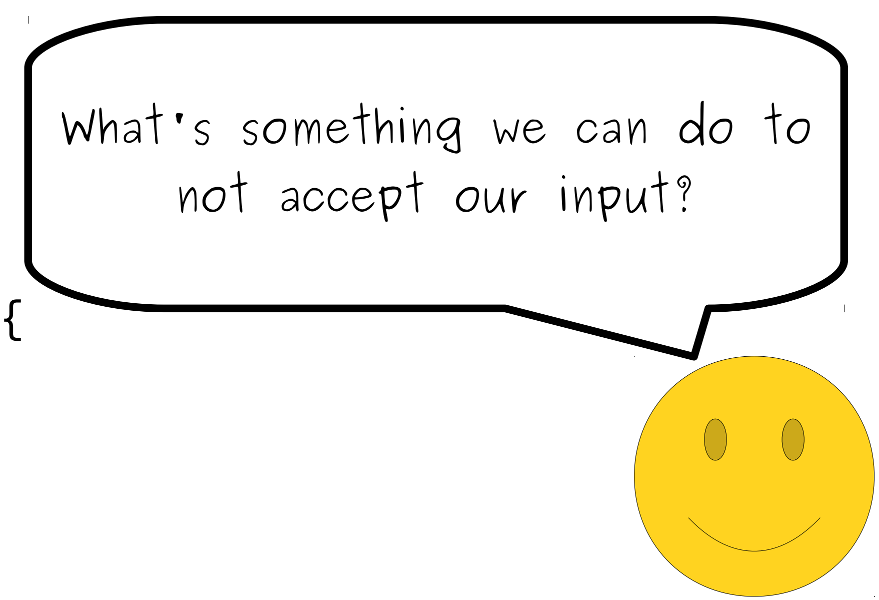



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {

  } else {

  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

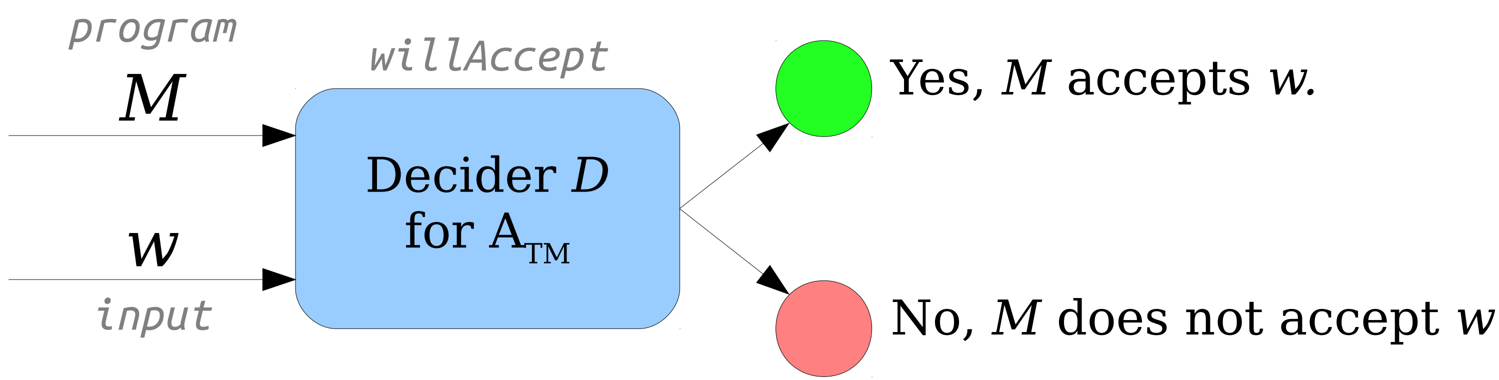
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!

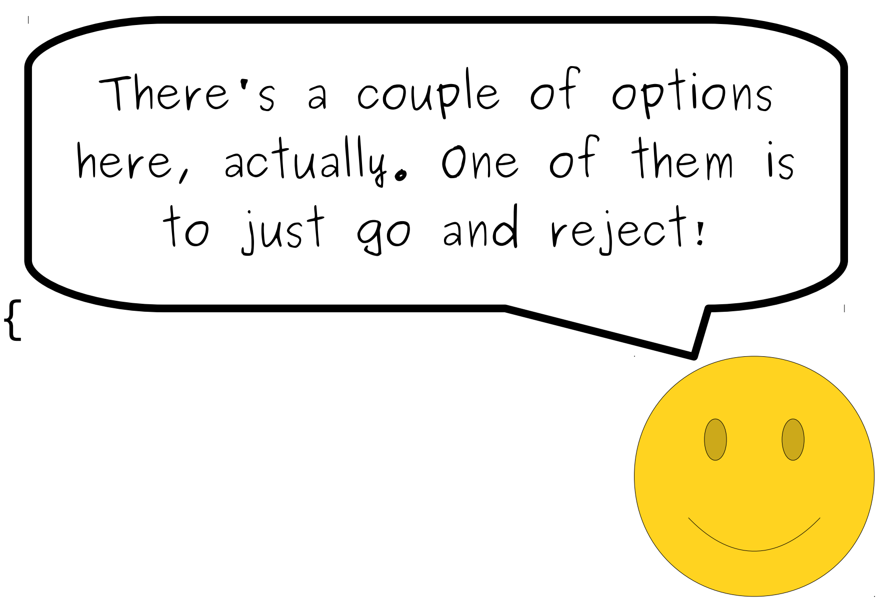



```
bool willAccept(string program, string input)
```

Program *P* design specification:

If *P* accepts its input, then
P does not accept its input.
If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {

  }
}
```



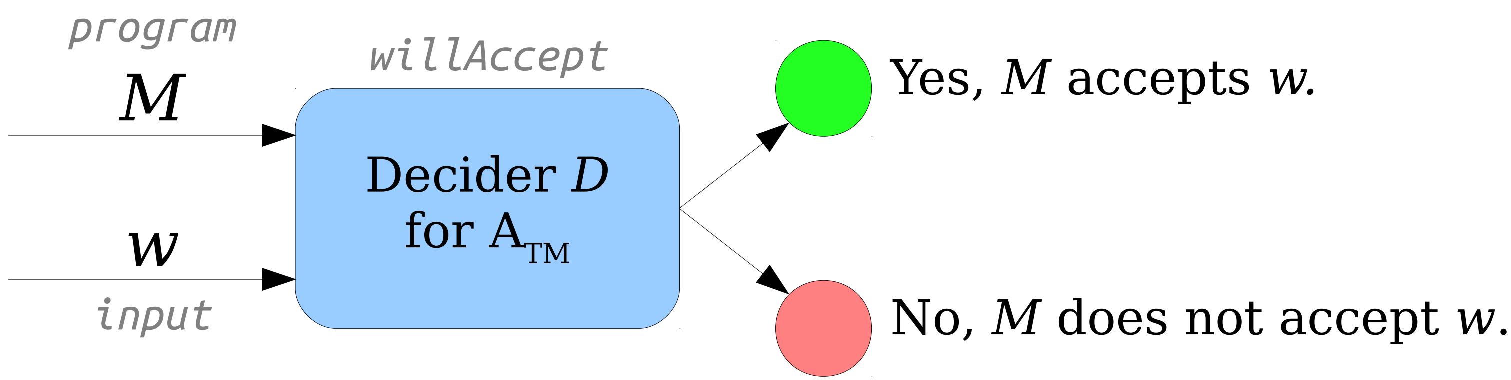
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!



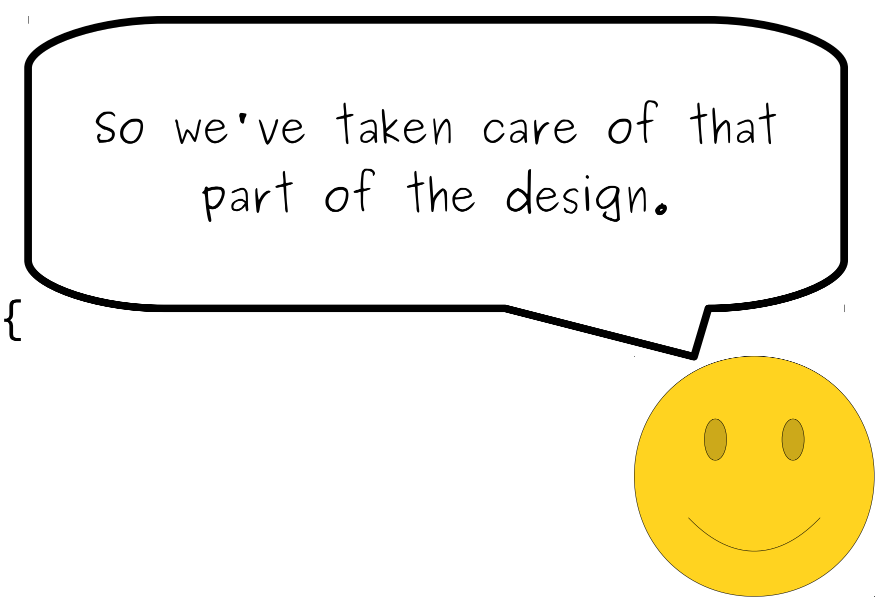

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {

  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

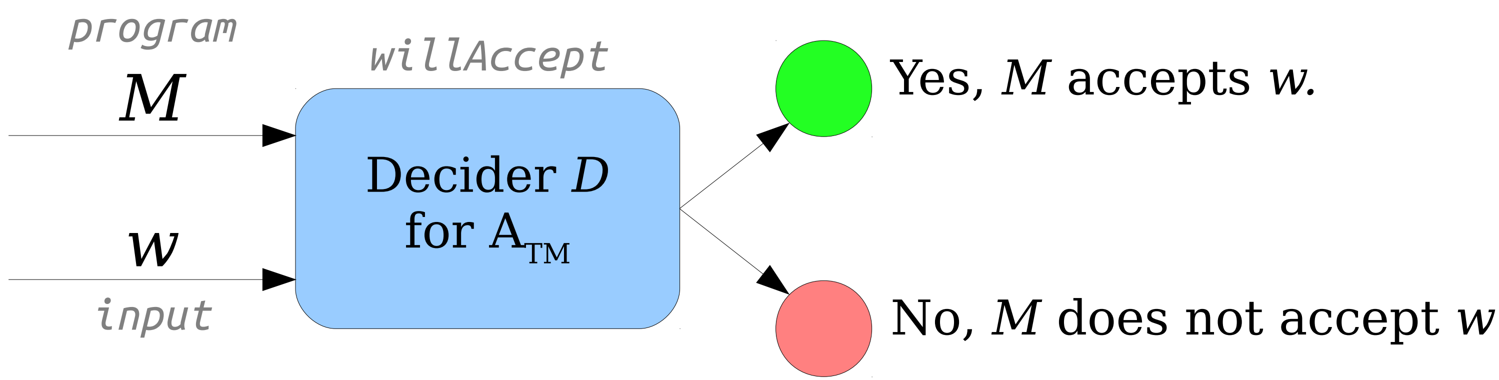
↓

We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!



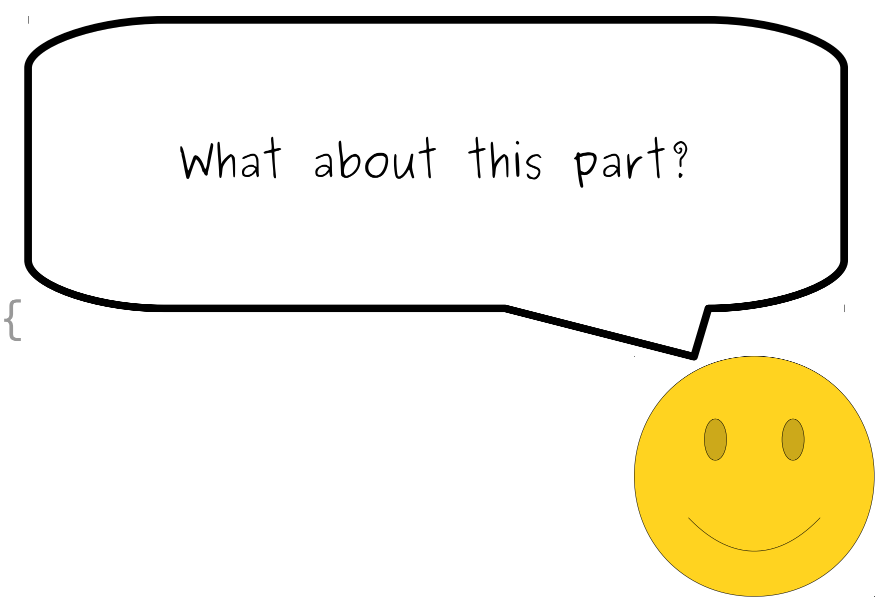

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {

  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

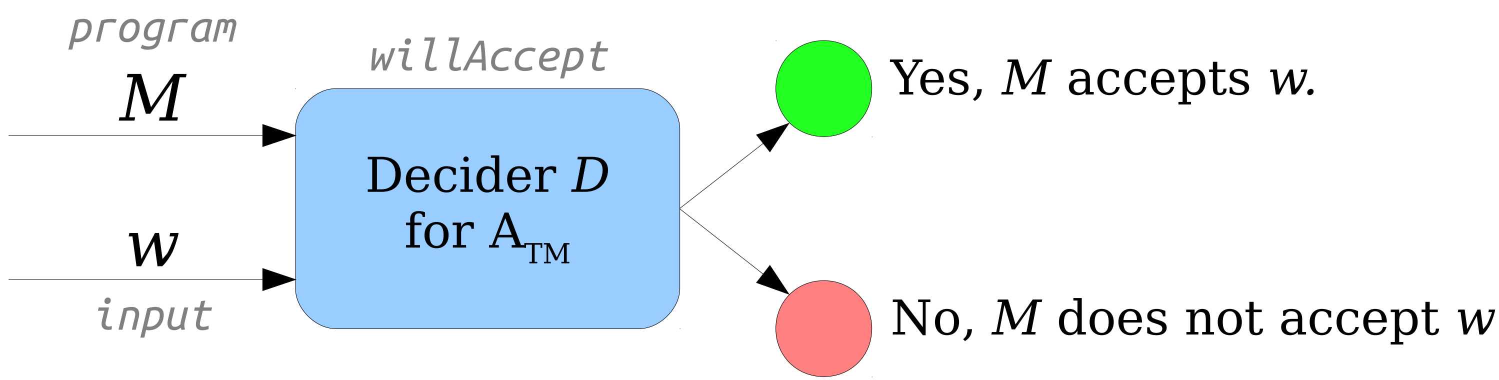
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!



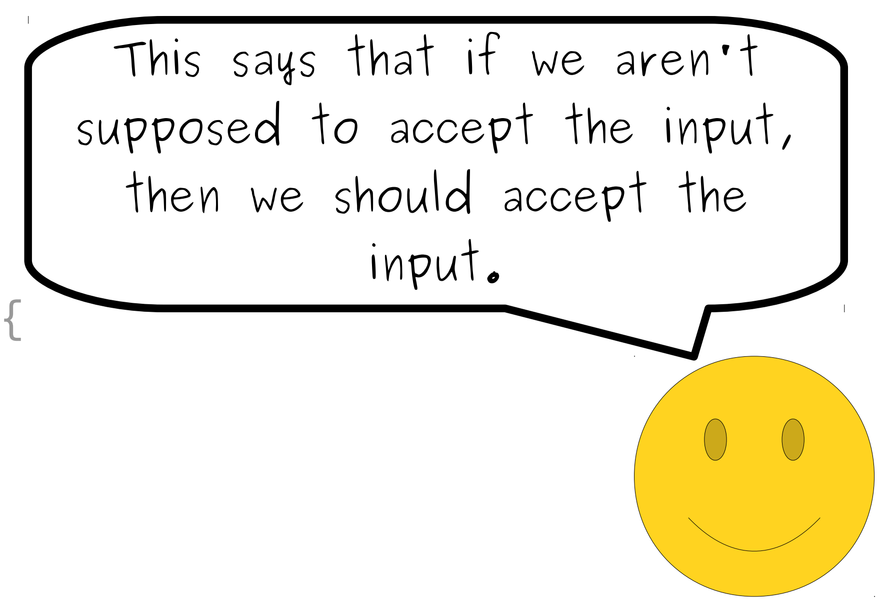

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {

  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

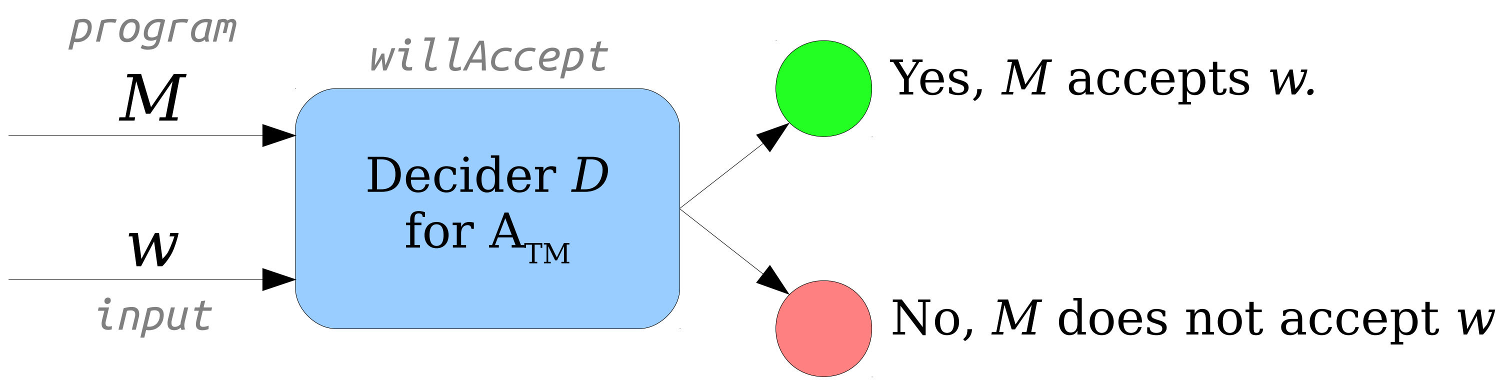
↓

We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

Contradiction!



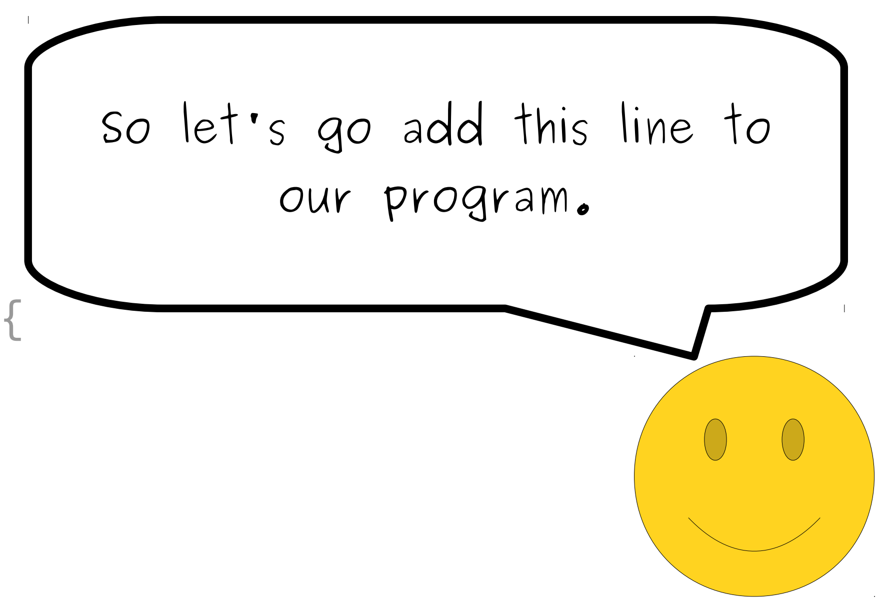

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



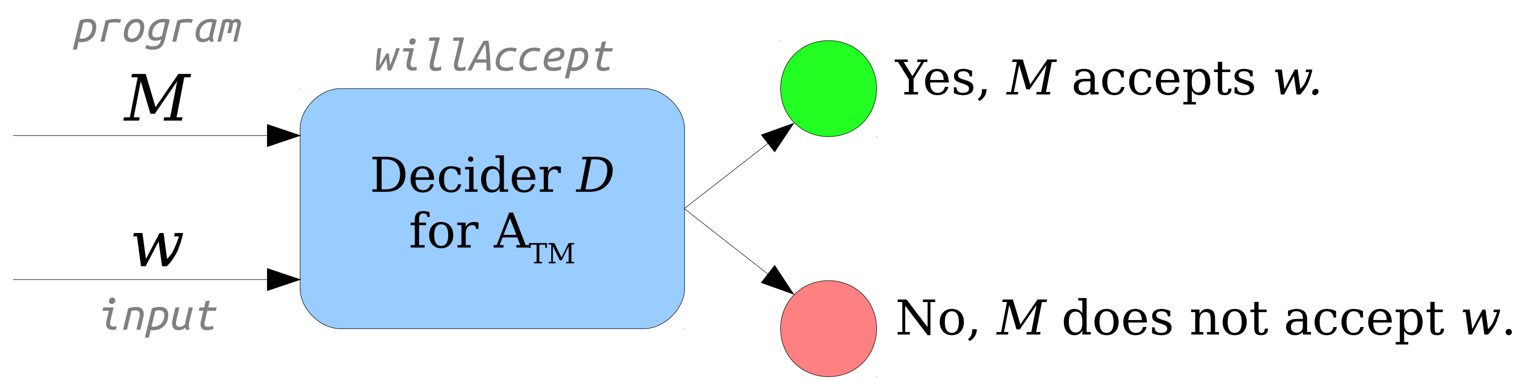
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!



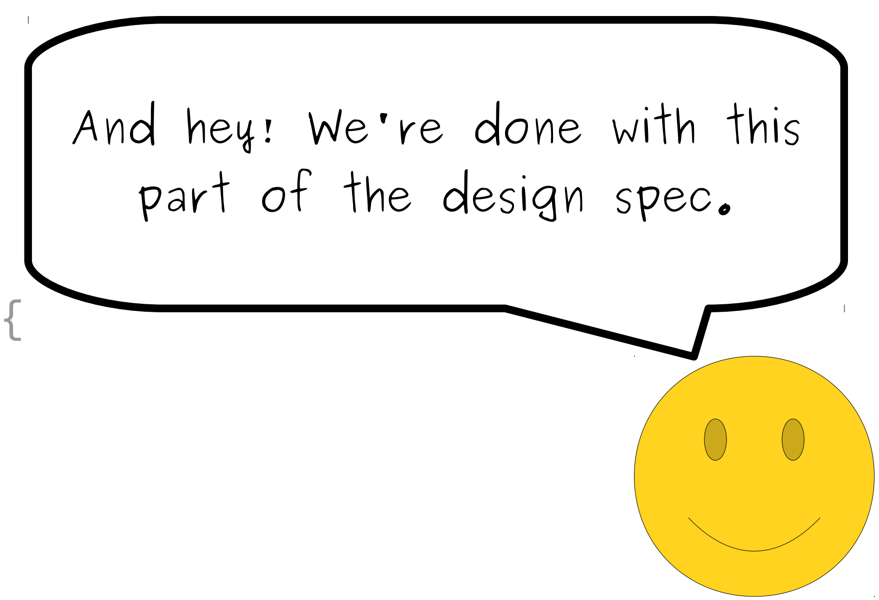

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



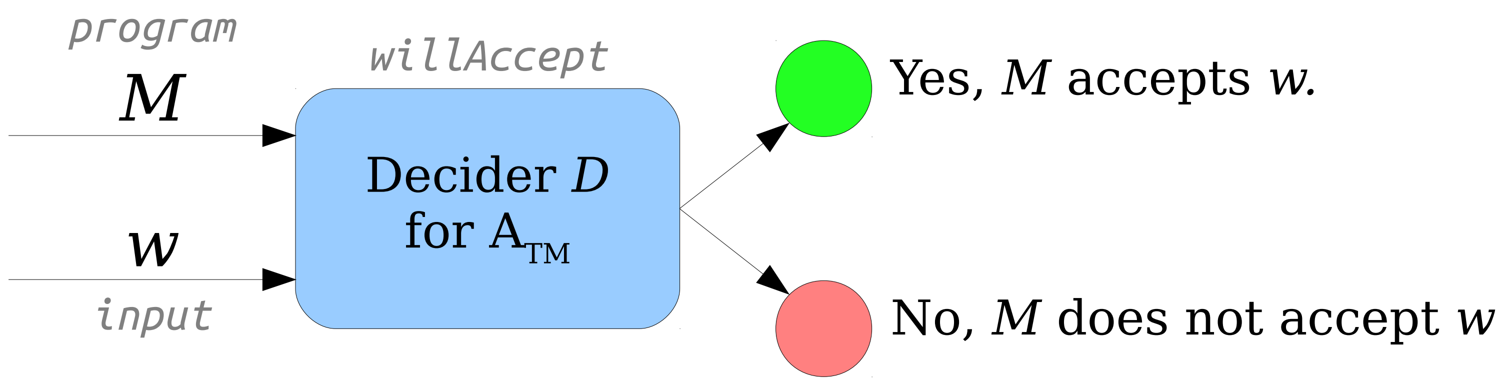
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!

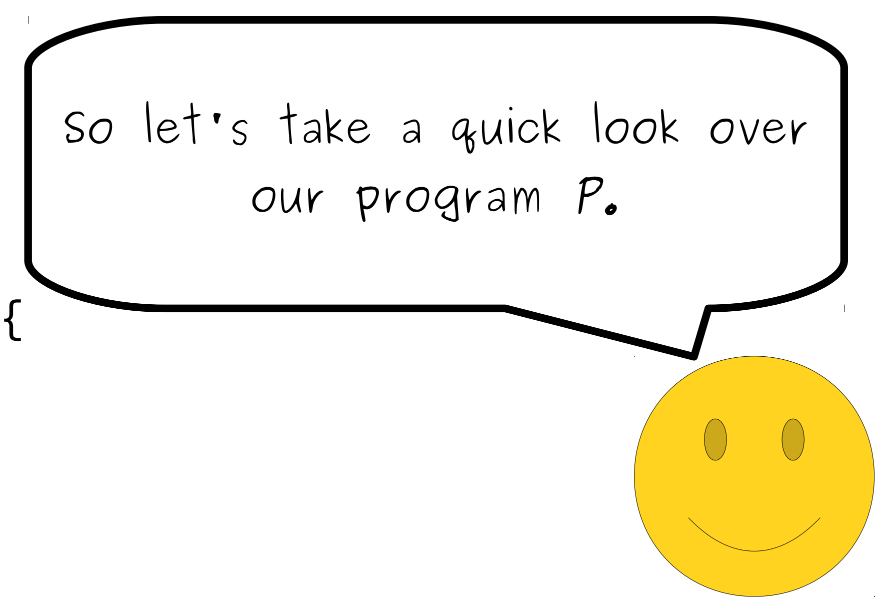



```
bool willAccept(string program, string input)
```

Program *P* design specification:

- ✓ If *P* accepts its input, then *P* does not accept its input.
- ✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

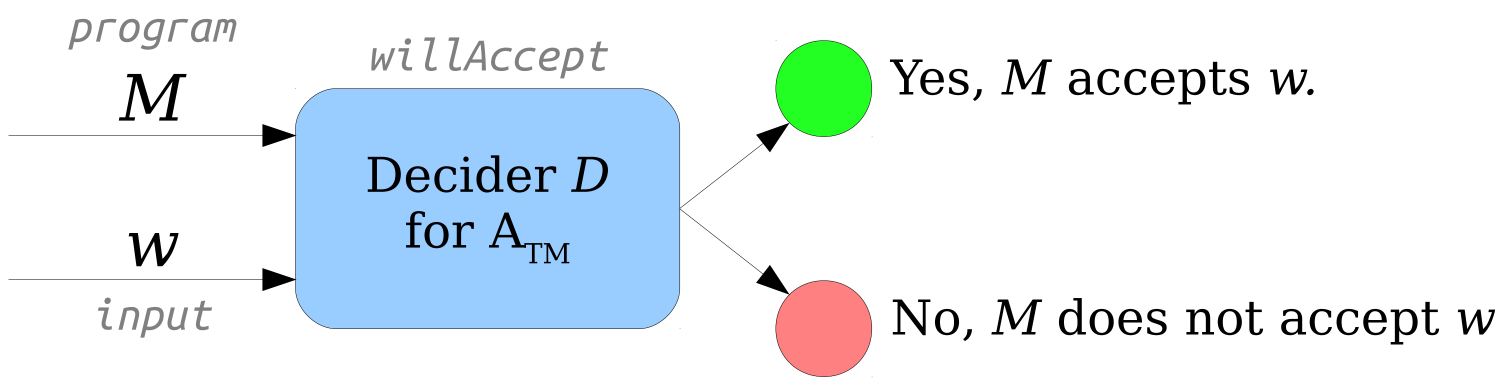
↓

We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!



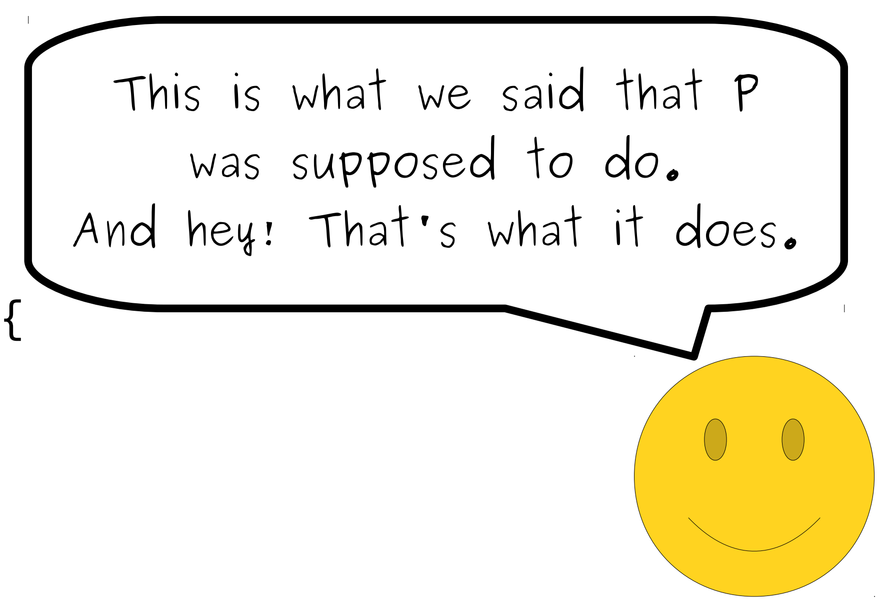

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



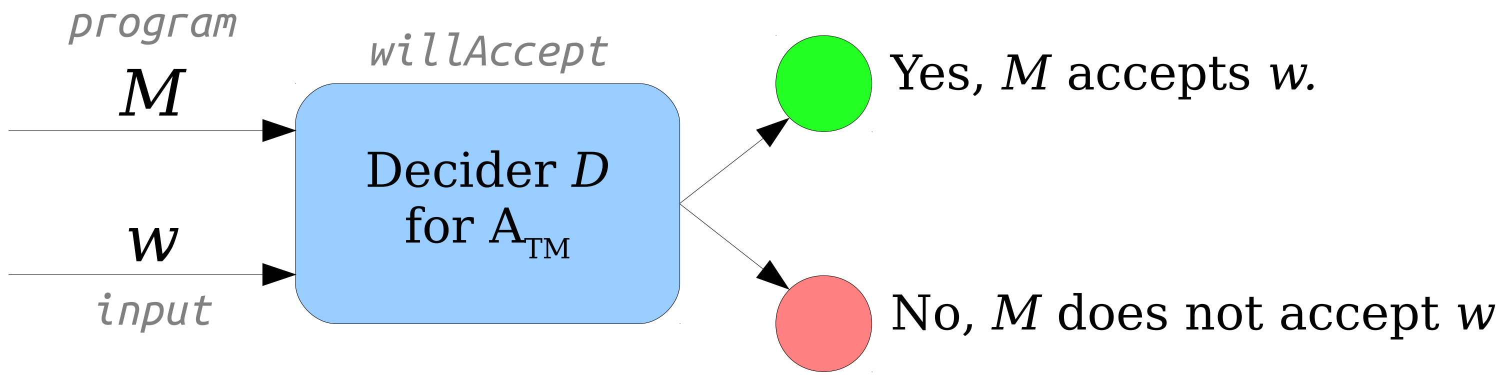
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!



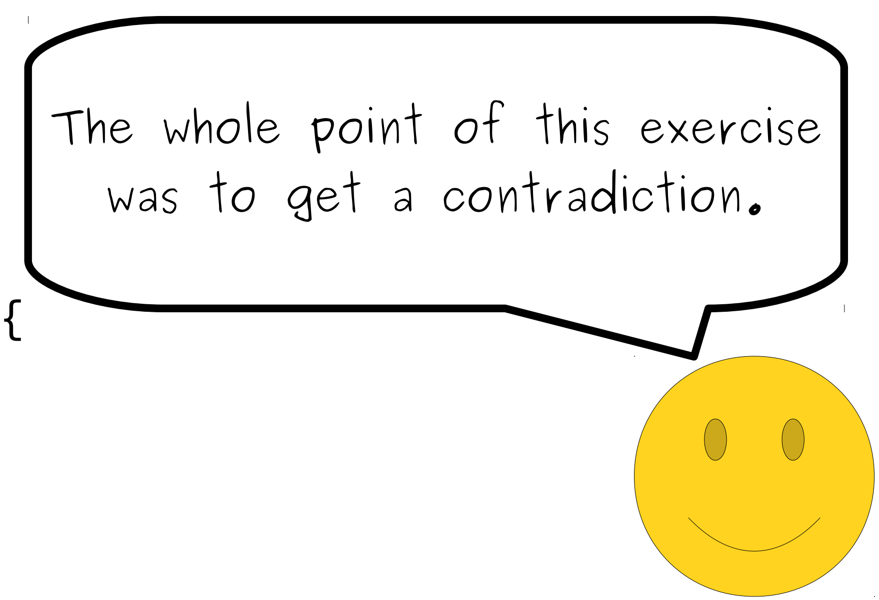

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

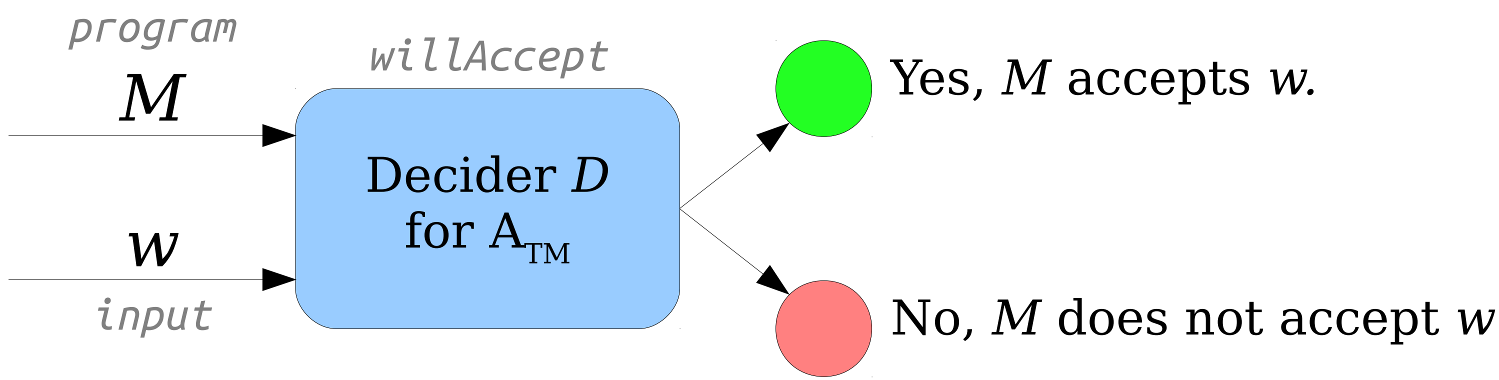
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



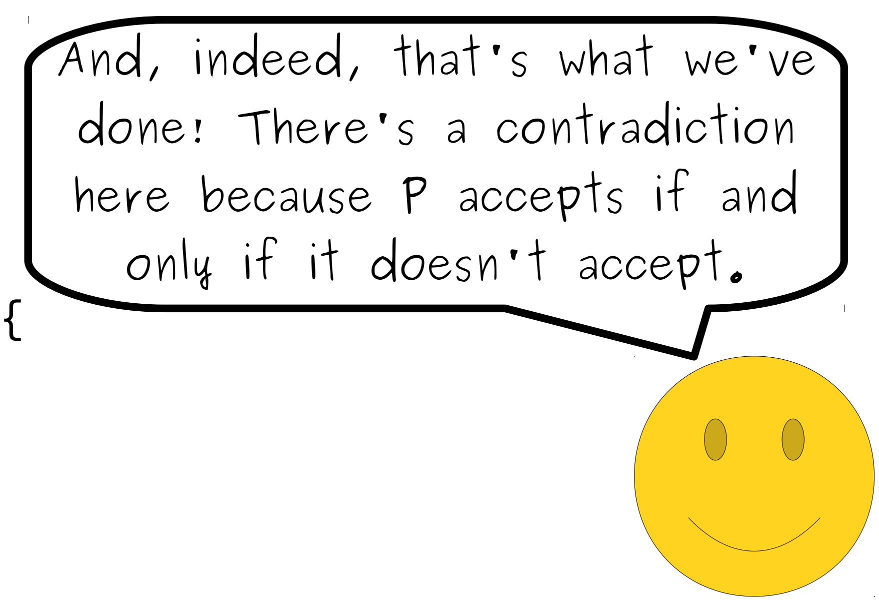

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



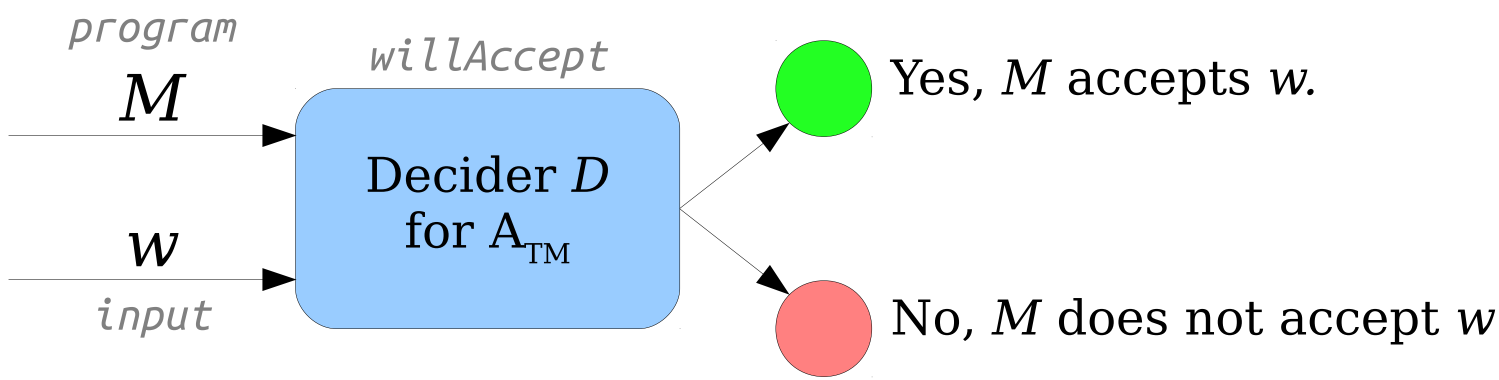
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!



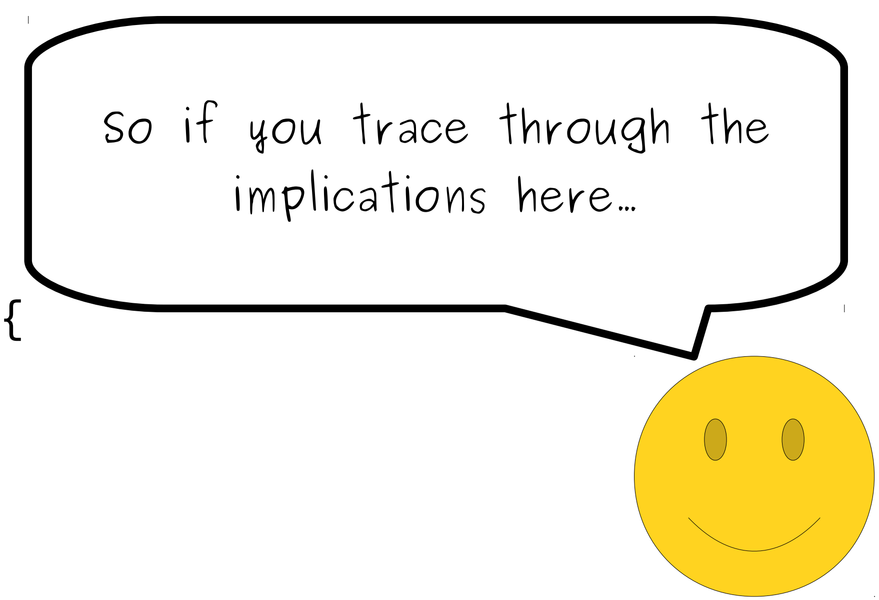

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

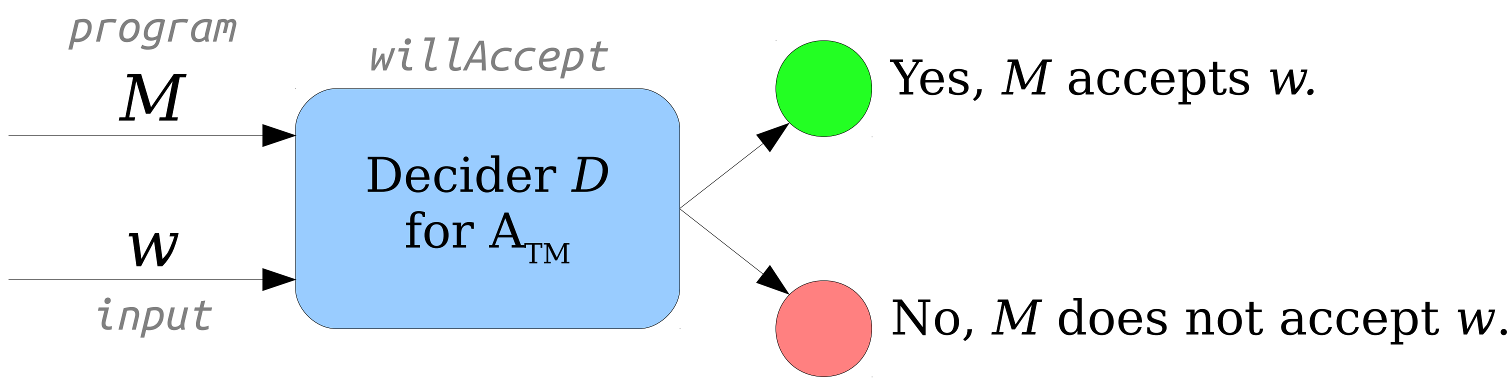
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



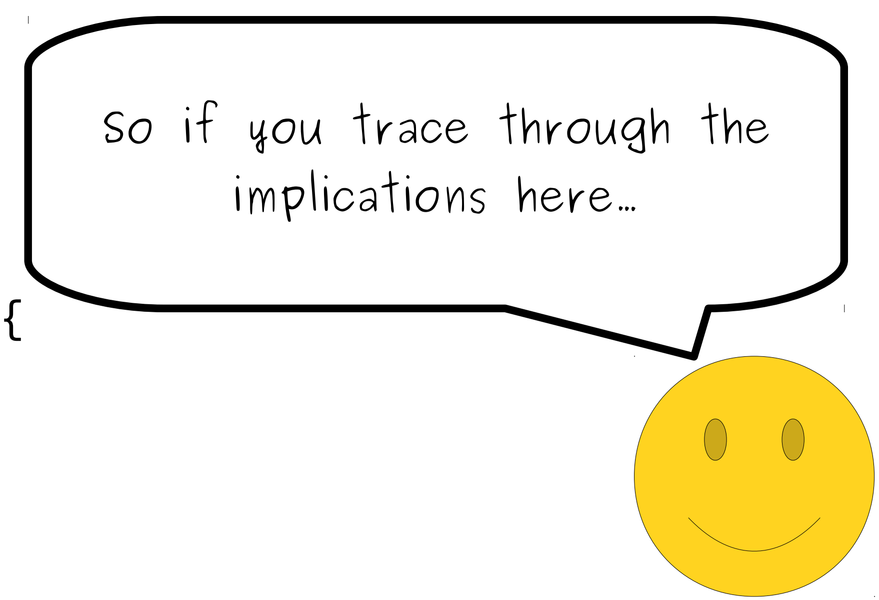

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
  P does not accept its input.

✓ If P does not accept its input, then
  P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

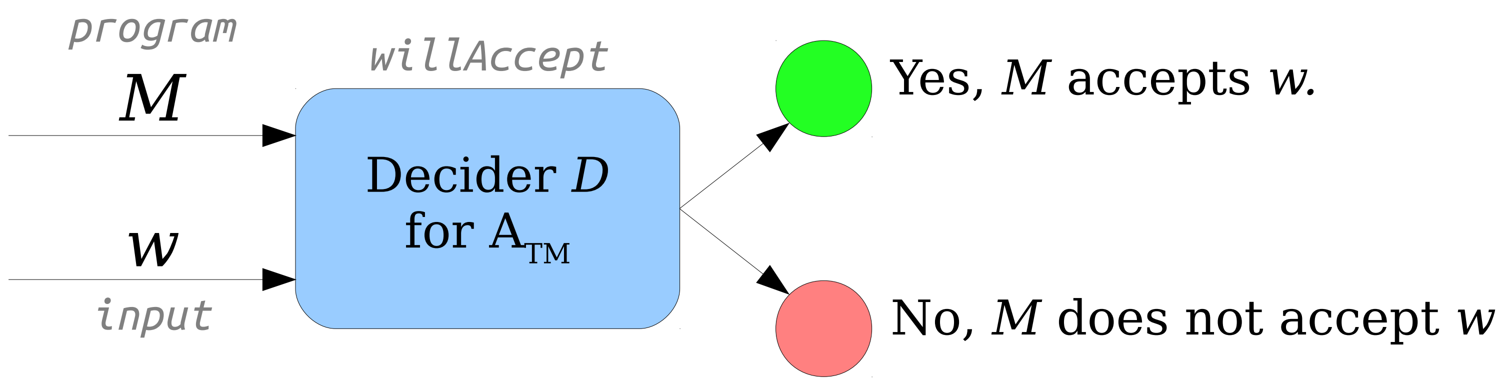
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



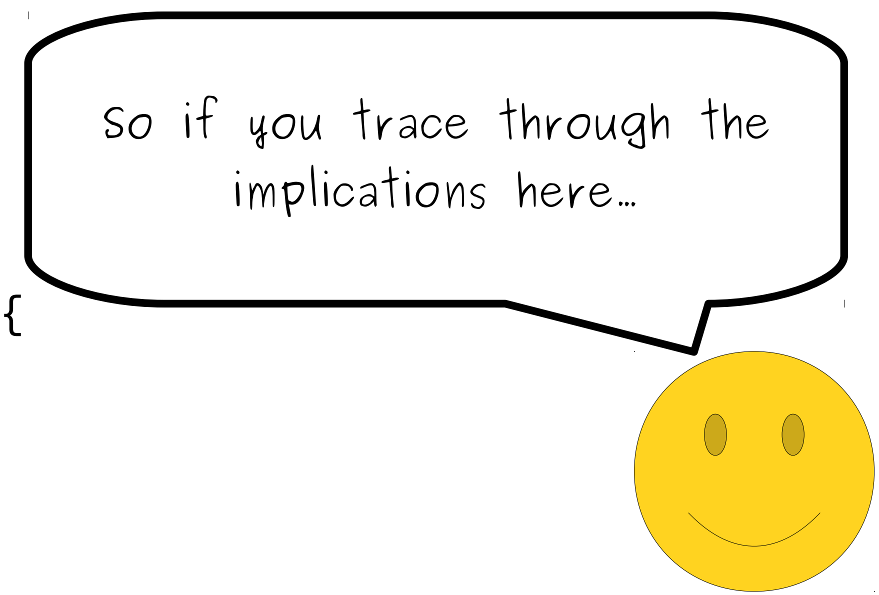

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

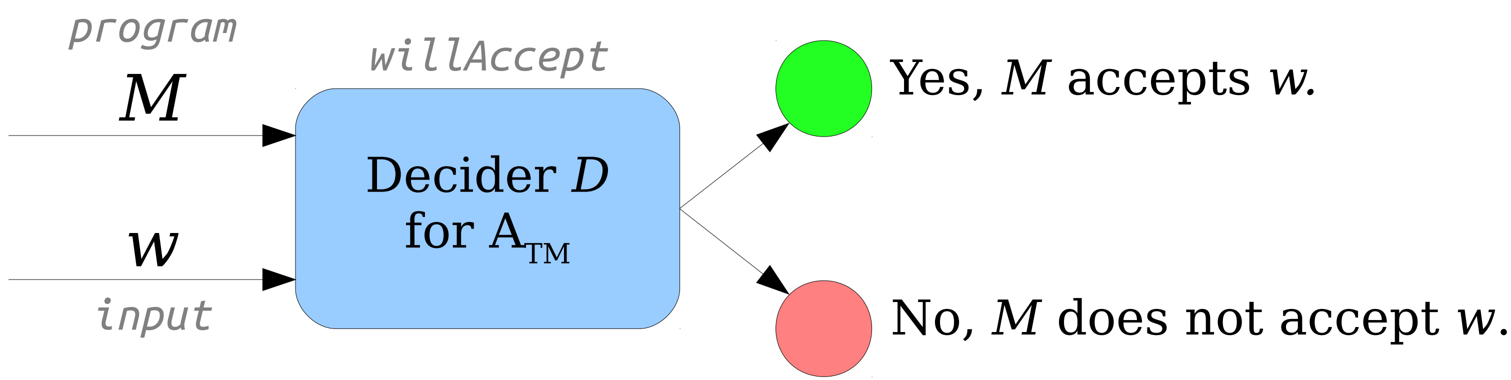
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



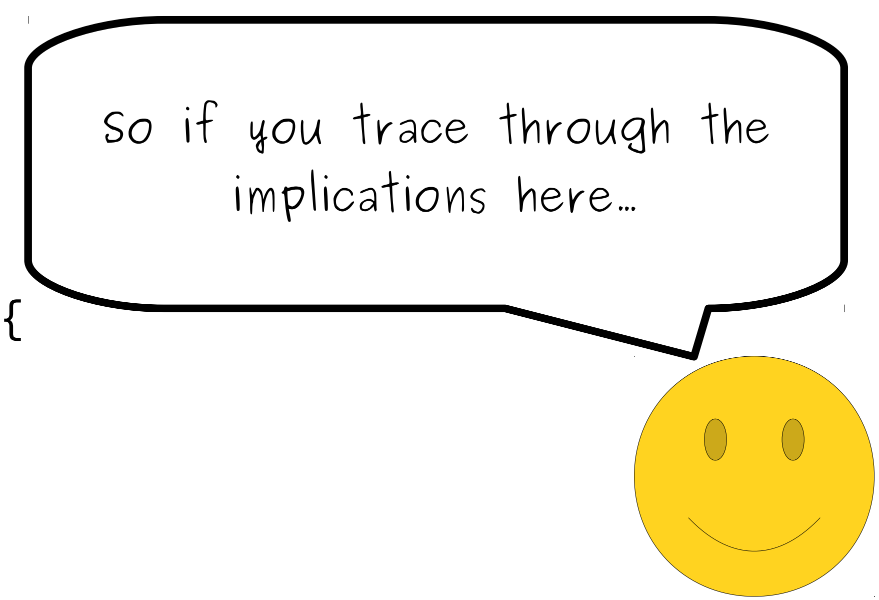

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



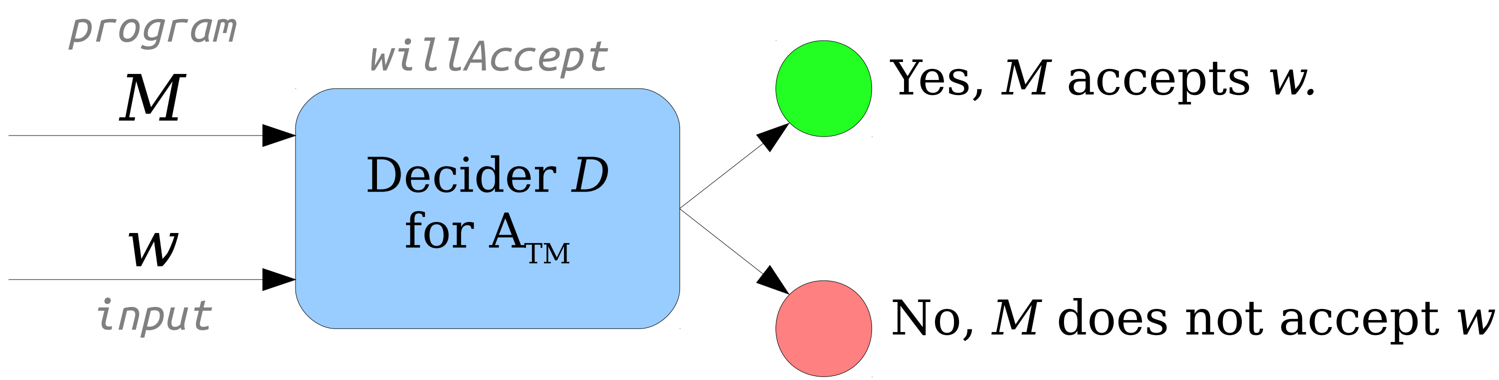
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!

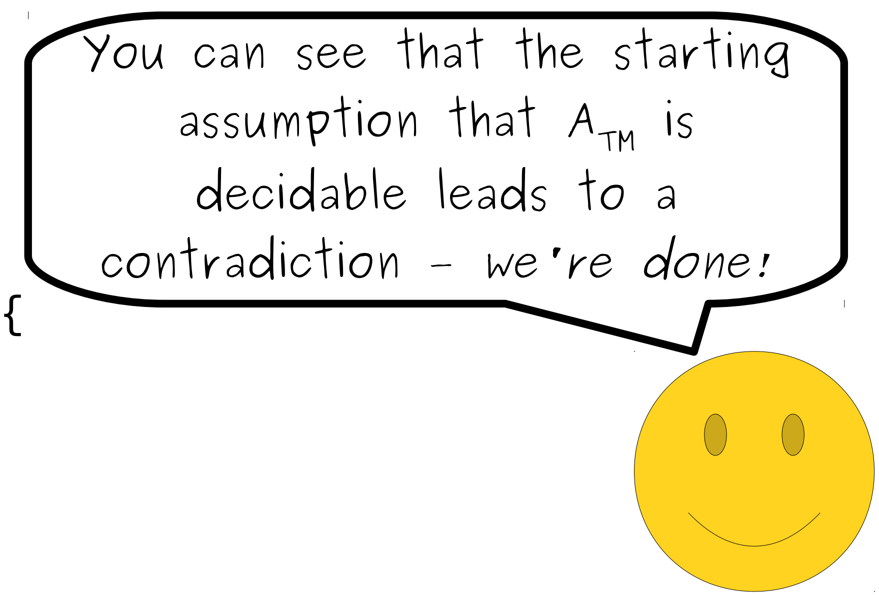



```
bool willAccept(string program, string input)
```

Program P design specification:

- ✓ If P accepts its input, then
  P does not accept its input.
- ✓ If P does not accept its input, then
  P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



What does this program do?

```
bool willAccept(string program, string input) {
   /* … some implementation … */
}

int main() {
   string me = mySource();
   string input = getInput();

   if (willAccept(me, input)) {
      reject();
   } else {
      accept();
   }
}
```

Try running this program on any input.
What happens if

… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!

What does this program do?

```
bool willAccept(string program, string input) {
   /* … some implementation … */
}

int main() {
   string me = mySource();
   string input = getInput();

   if (willAccept(me, input)) {
      reject();
   } else {
      accept();
   }
}
```

Take a look at it more closely.

Try running this program on any input.
What happens if

… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!

What does this program do?

```
bool willAccept(string program, string input) {
    /* … some implementation … */
}

int main() {
    string me = mySource();
    string input = getInput();

    if (willAccept(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

Recognize this code? Now you know where it comes from!

Try running this program on any input. What happens if

… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!

What does this program do?

```
bool willAccept(string program, string input) {
   /* … some implementation … */
}

int main() {
   string me = mySource();
   string input = getInput();

   if (willAccept(me, input)) {
      reject();
   } else {
      accept();
   }
}
```

We created it to get these contradictions.

Try running this program on any input. What happens if

… this program accepts its input?
It rejects the input!

… this program doesn't accept its input?
It accepts the input!

$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

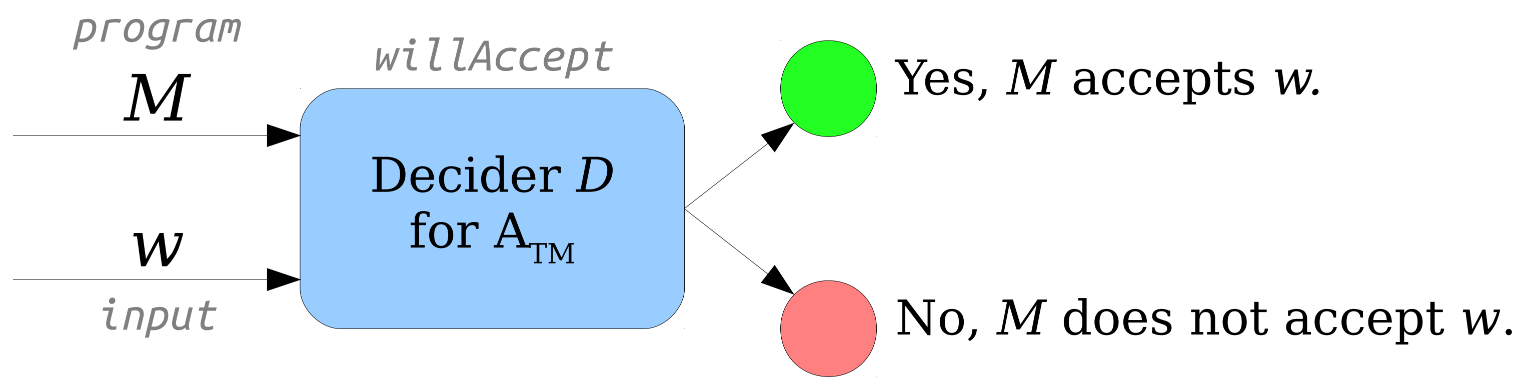
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!




```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
  P does not accept its input.

✓ If *P* does not accept its input, then
  P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```

$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

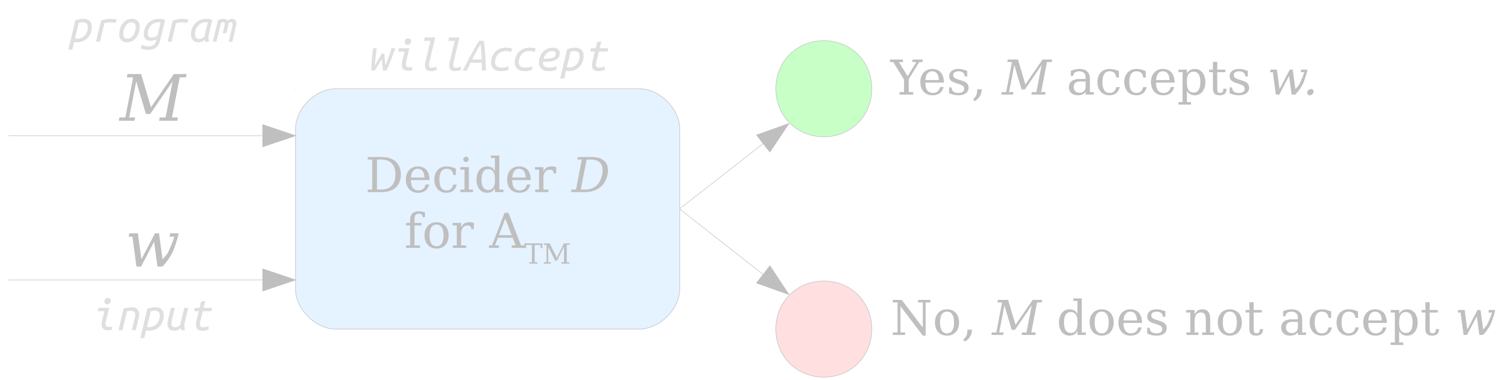
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



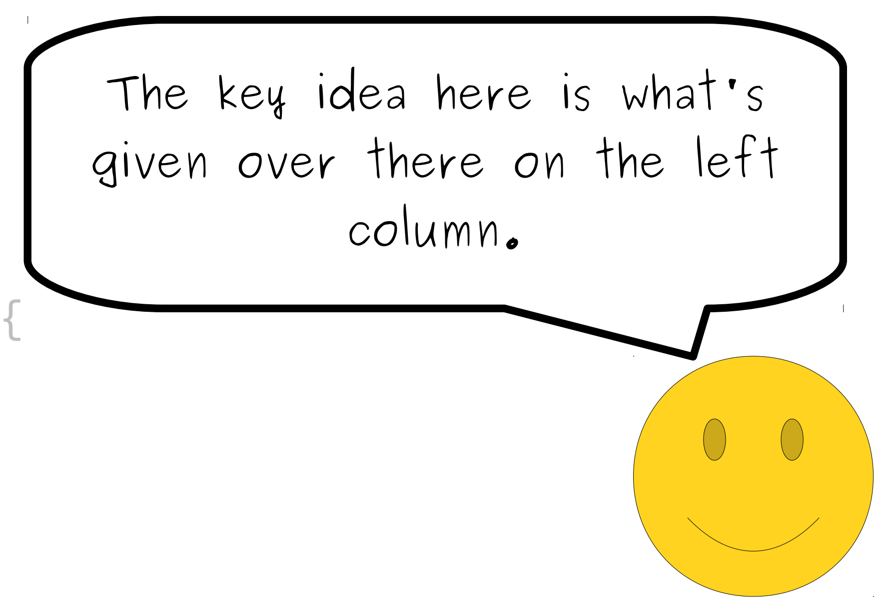

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

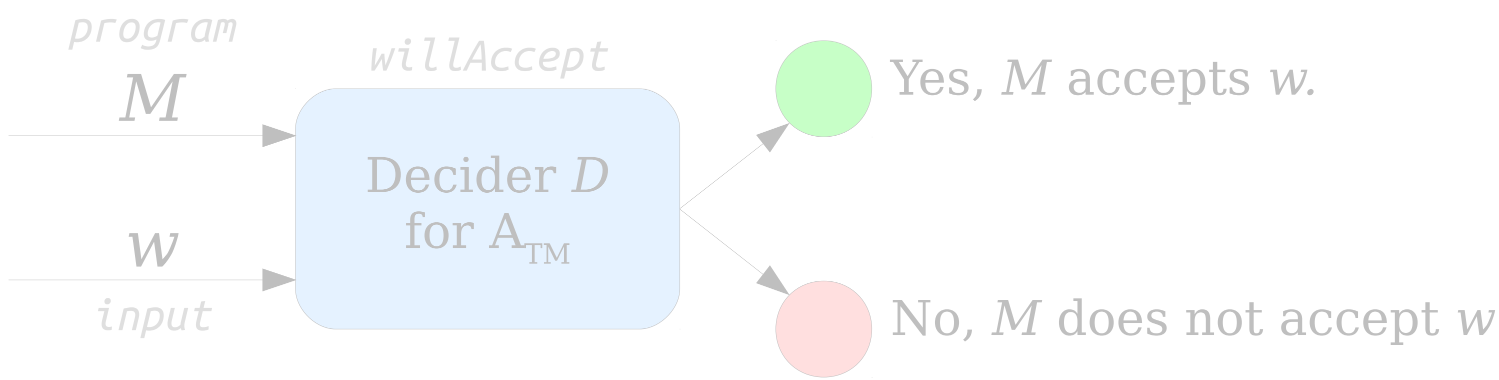
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!




```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```

$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

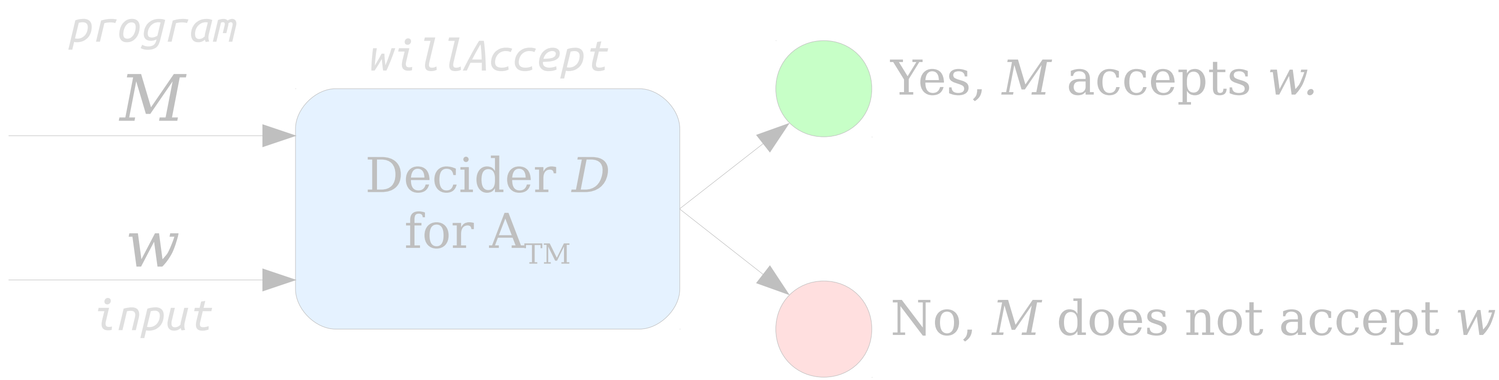
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



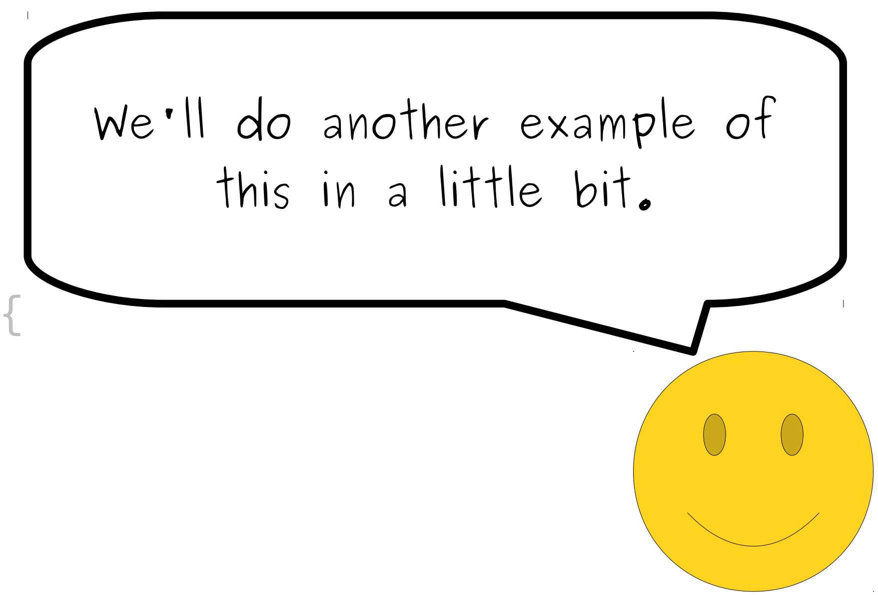

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

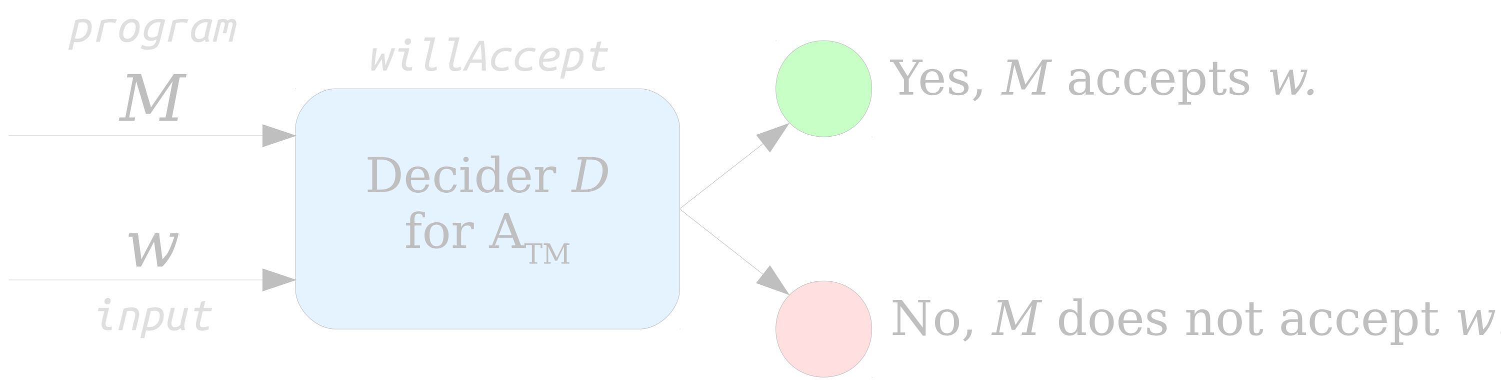
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



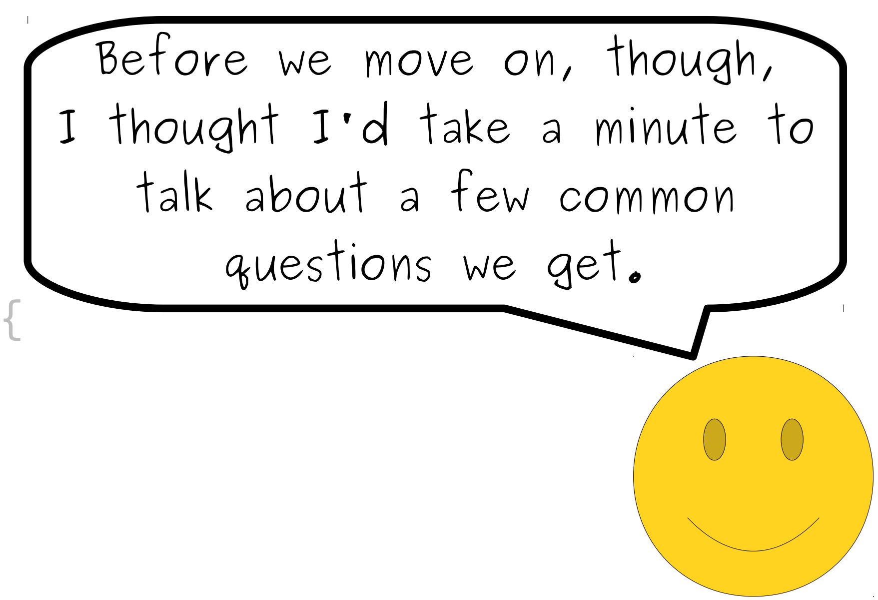

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

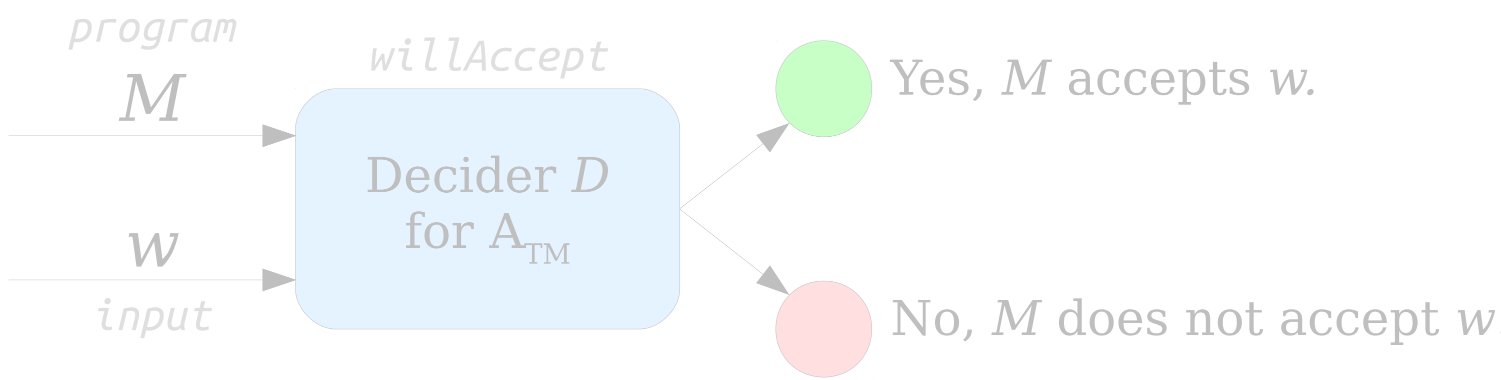
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



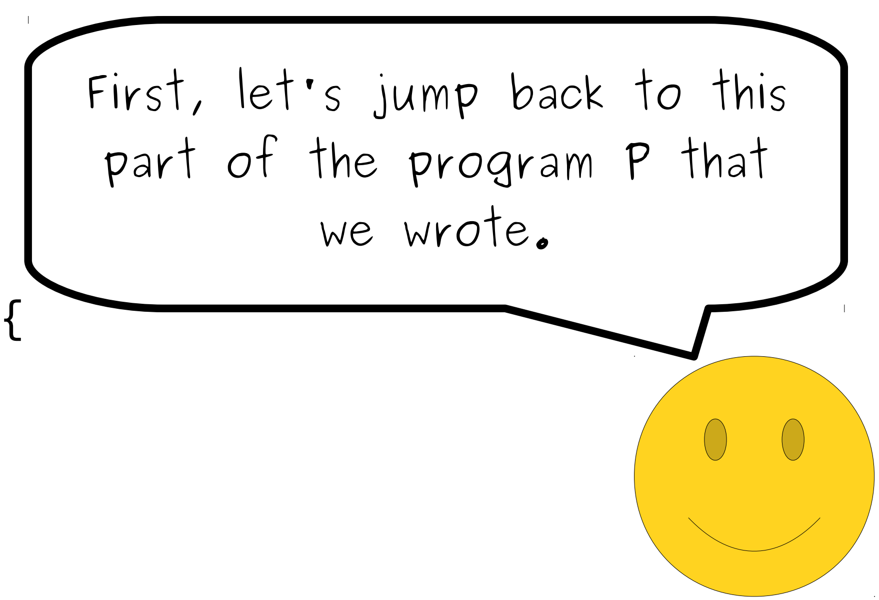

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

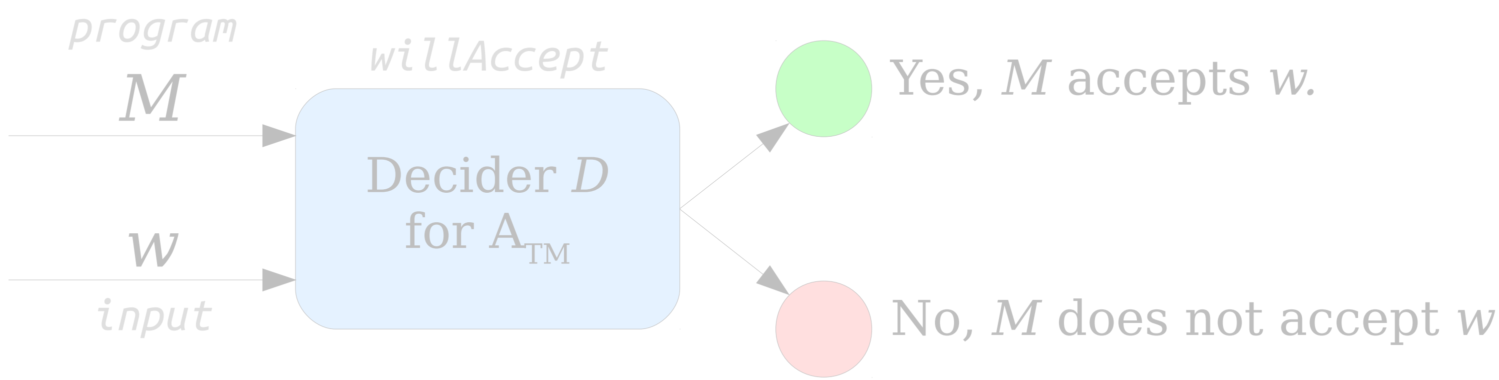
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



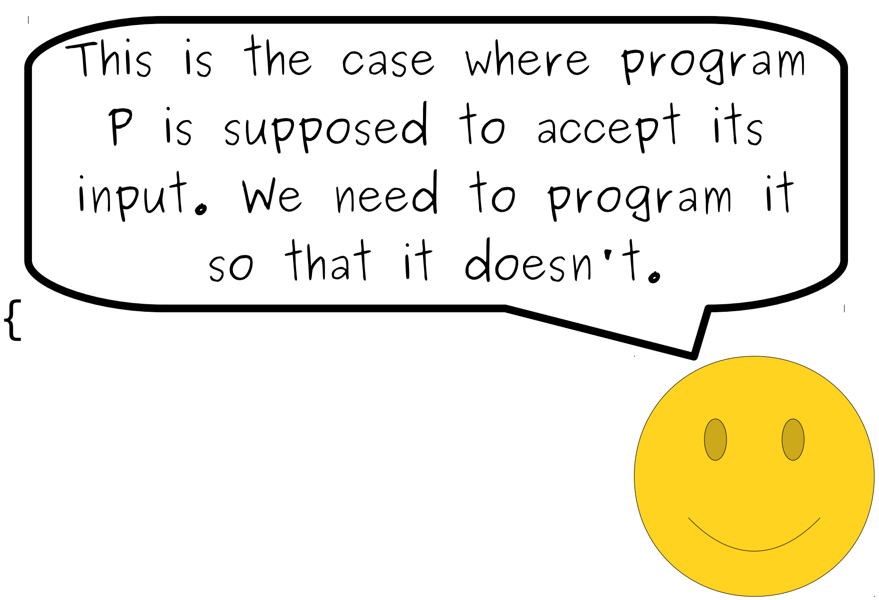

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

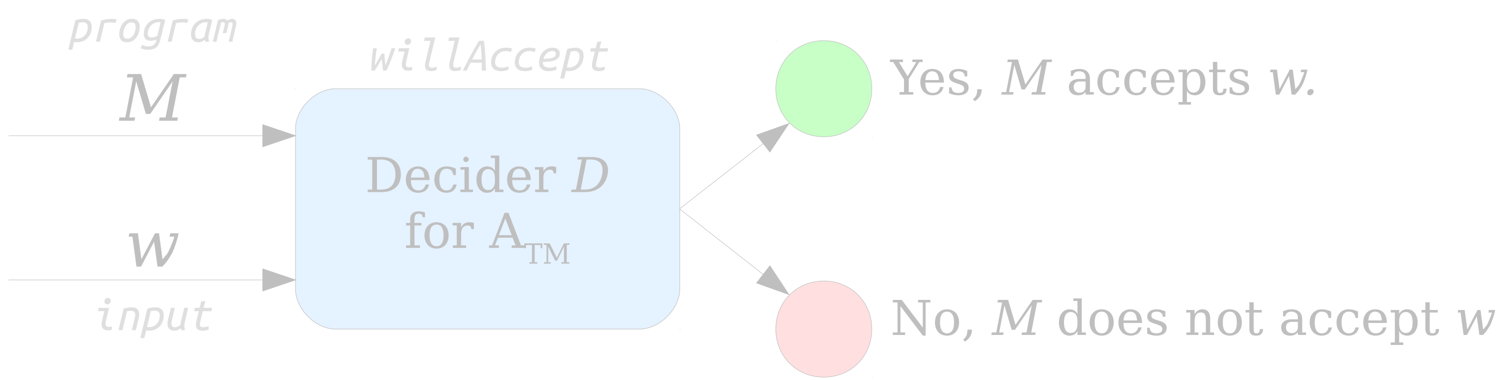
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



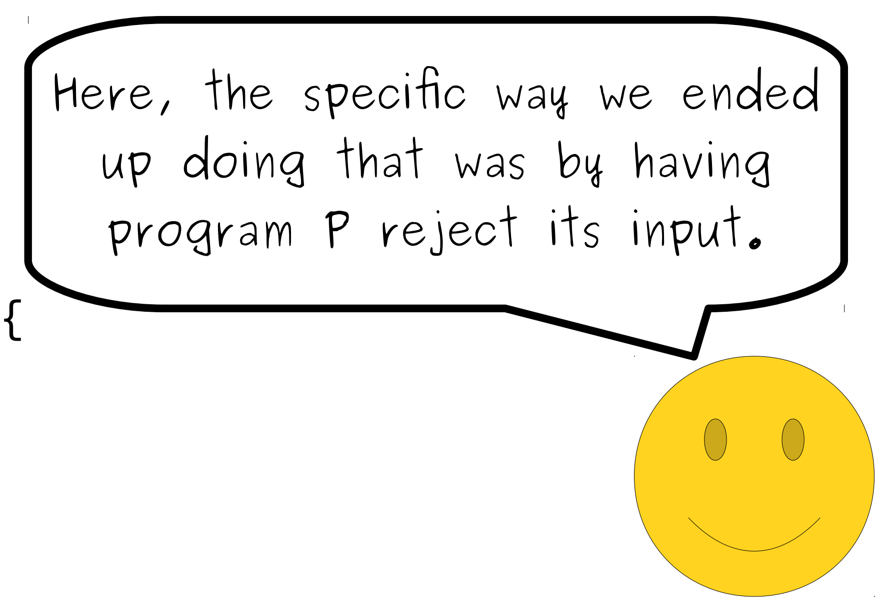

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

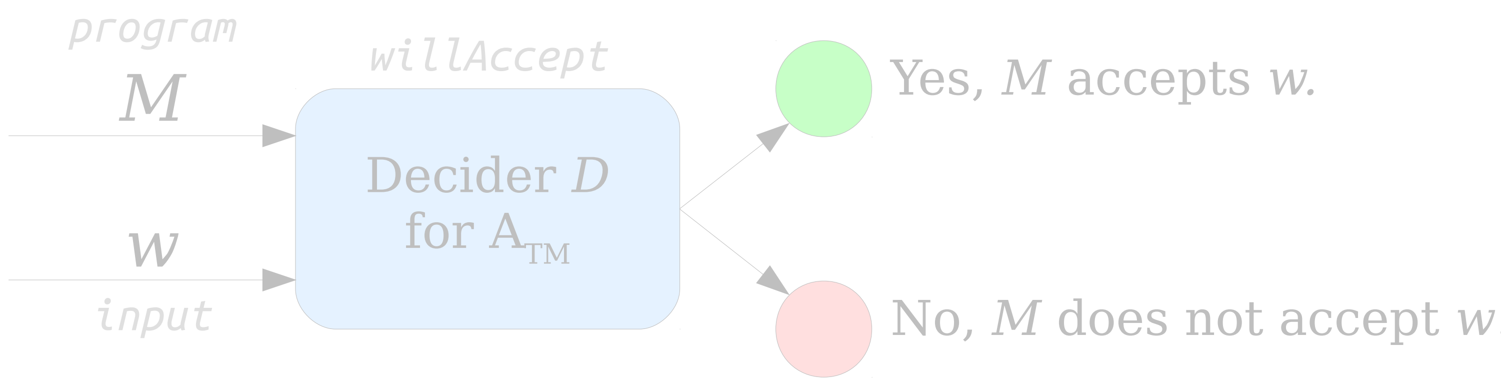
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



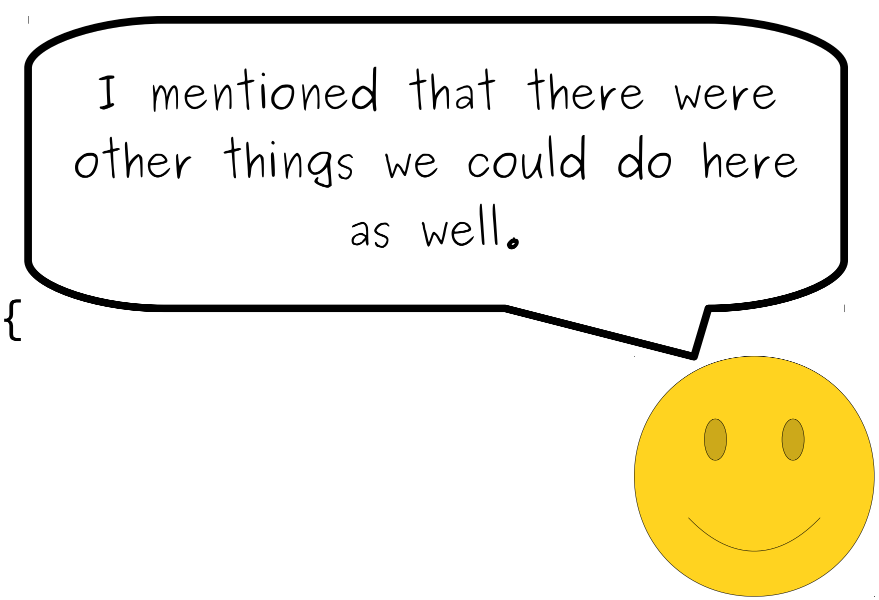

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

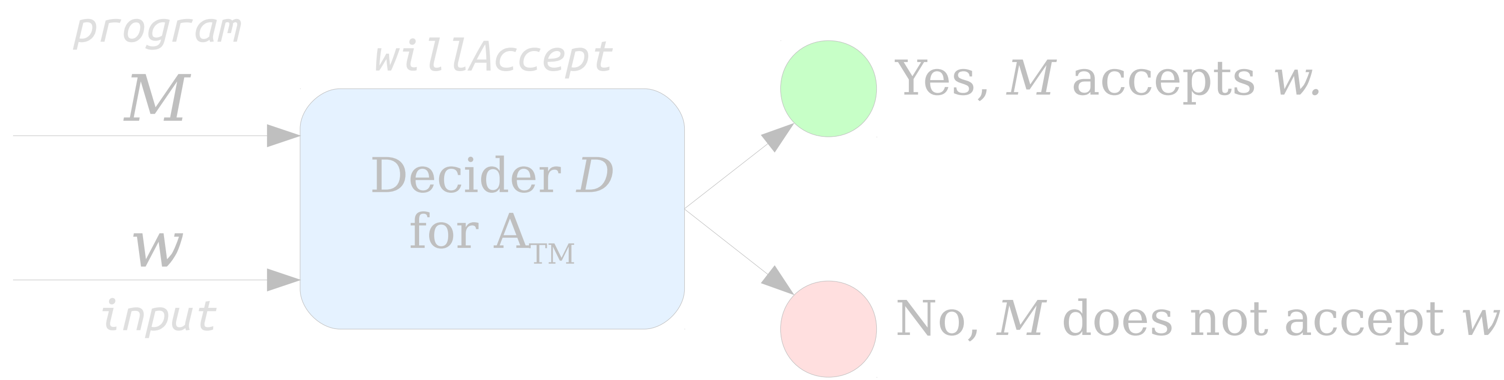
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



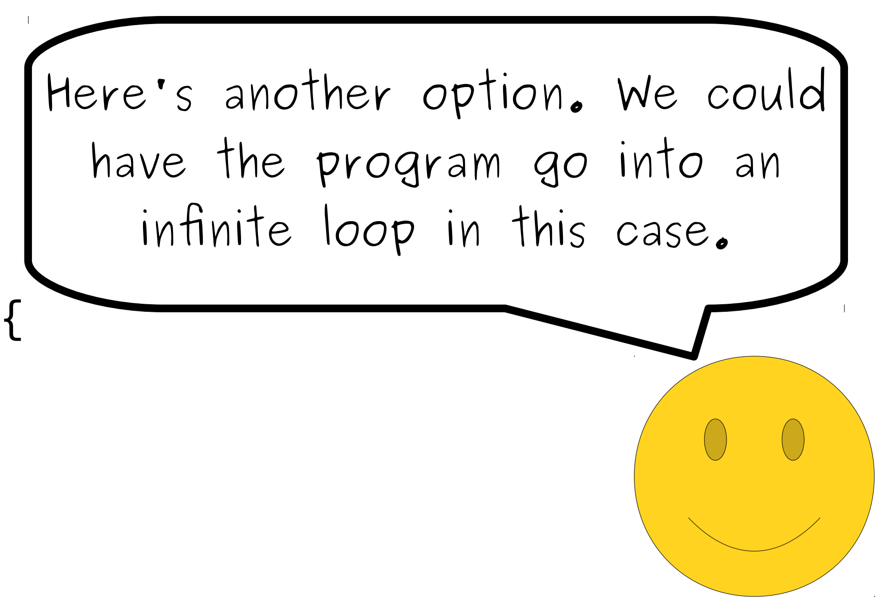

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

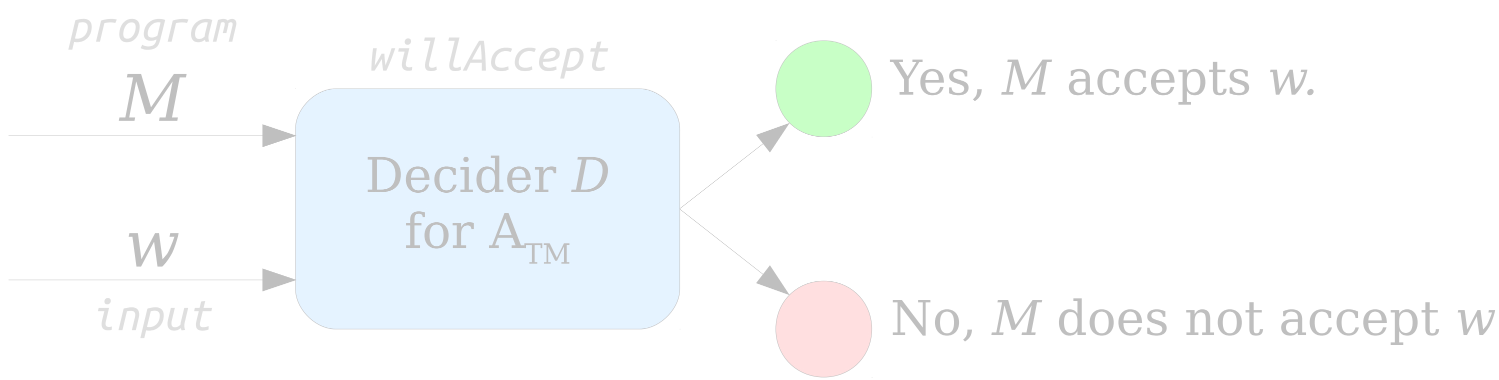
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



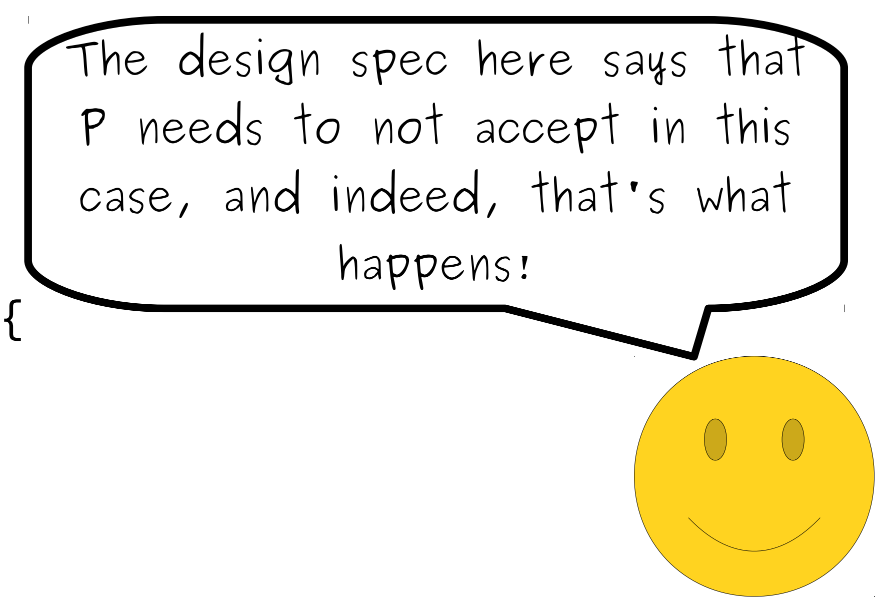

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

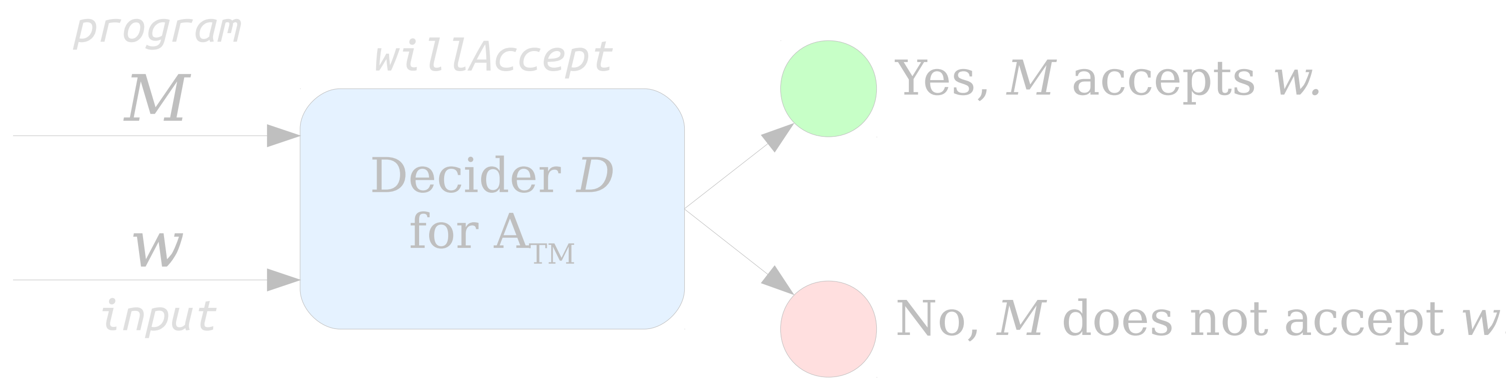
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



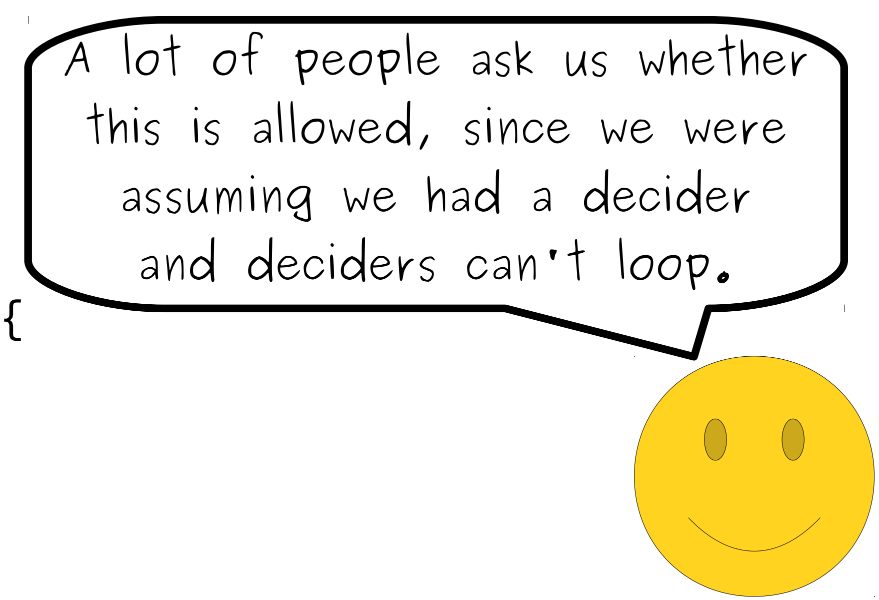

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

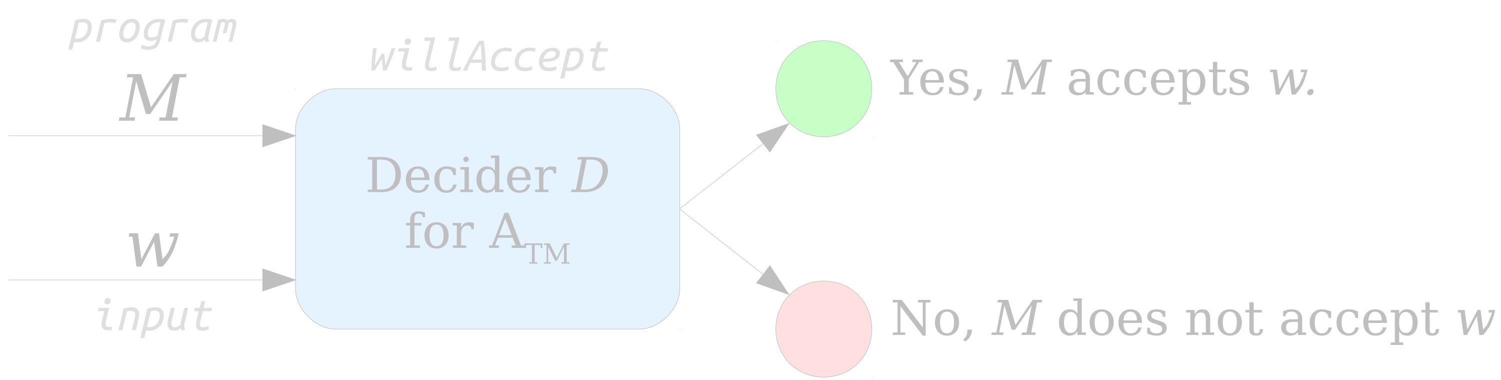
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



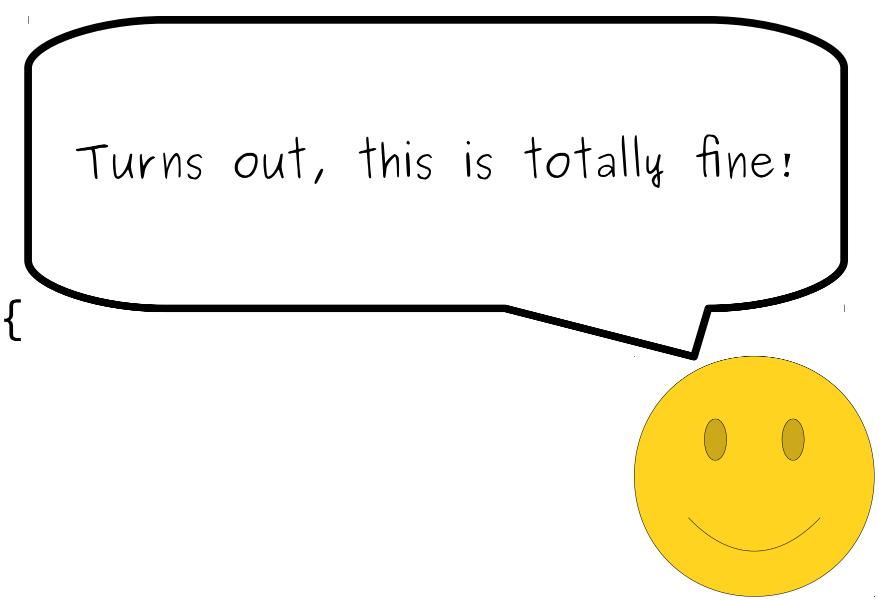

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

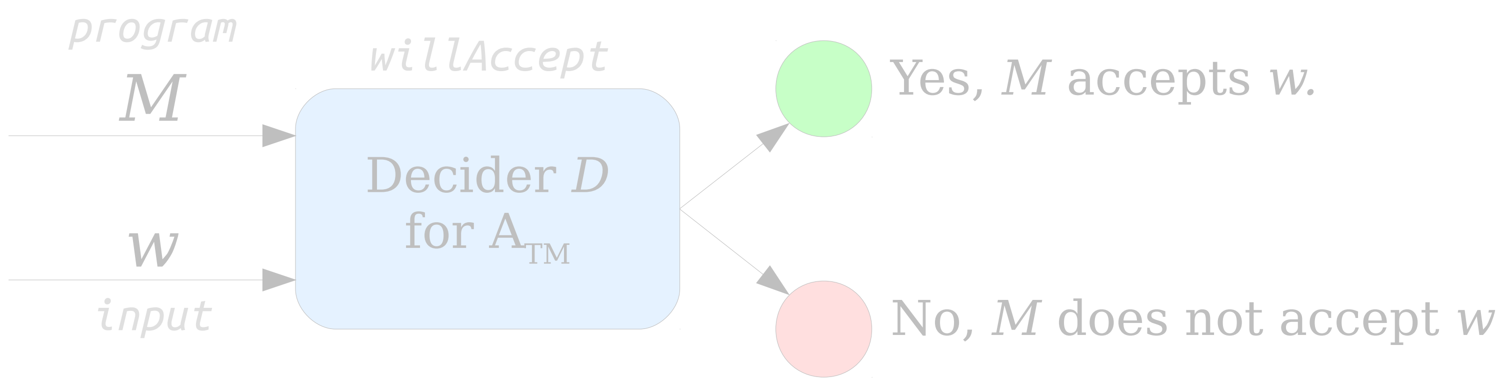
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



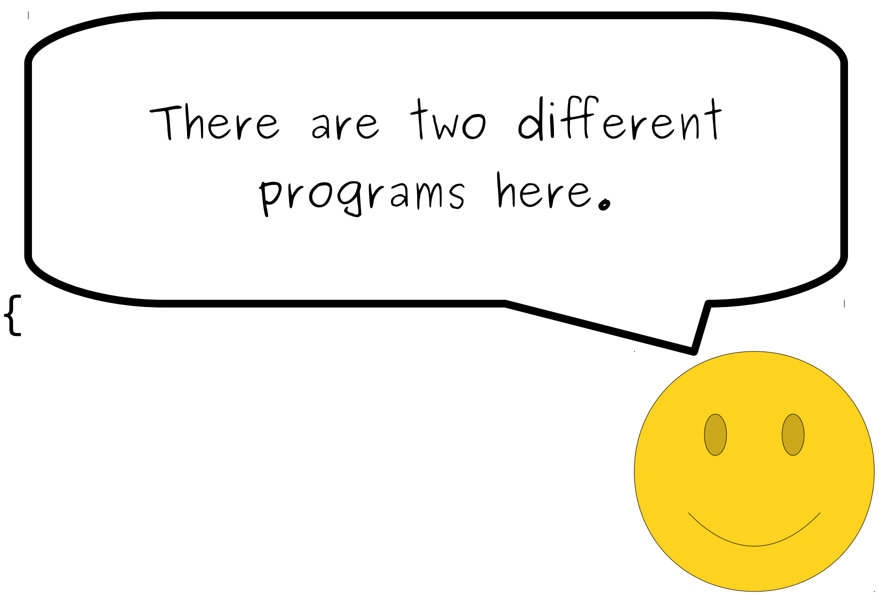

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



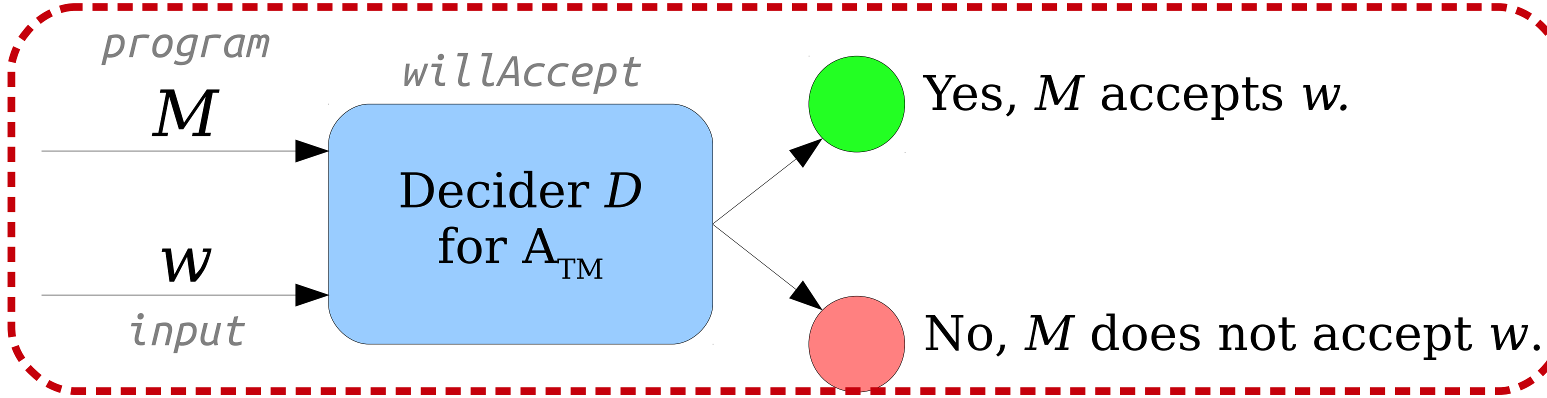
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!

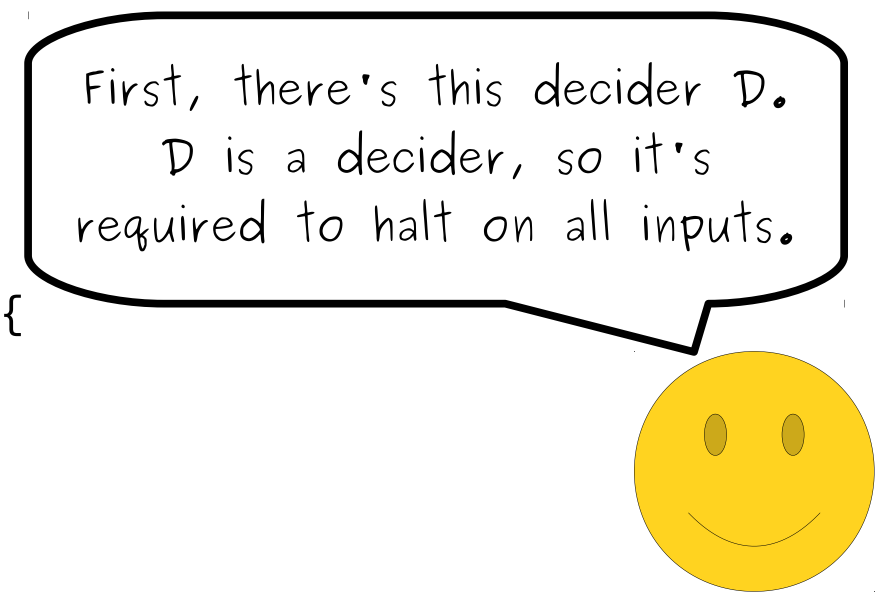



```
bool willAccept(string program, string input)
```

Program P design specification:

- ✓ If P accepts its input, then P does not accept its input.
- ✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

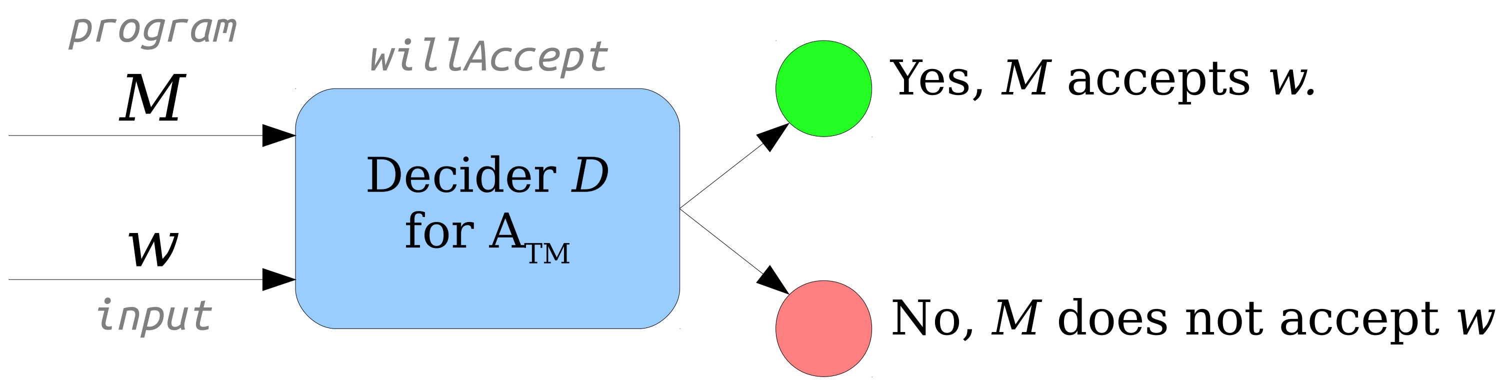
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!

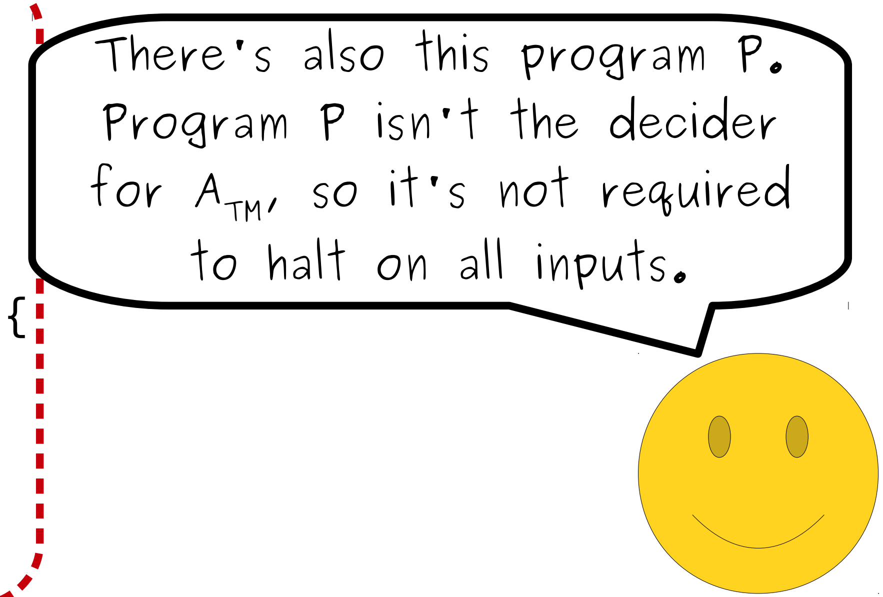



```
bool willAccept(string program, string input)
```

Program P design specification:

- ✓ If P accepts its input, then
  P does not accept its input.
- ✓ If P does not accept its input, then
  P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

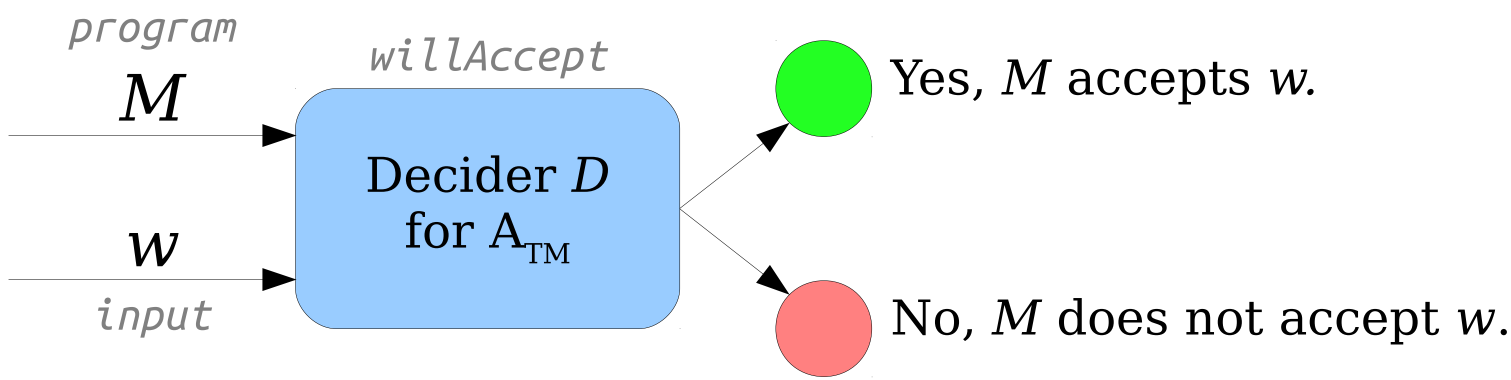
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



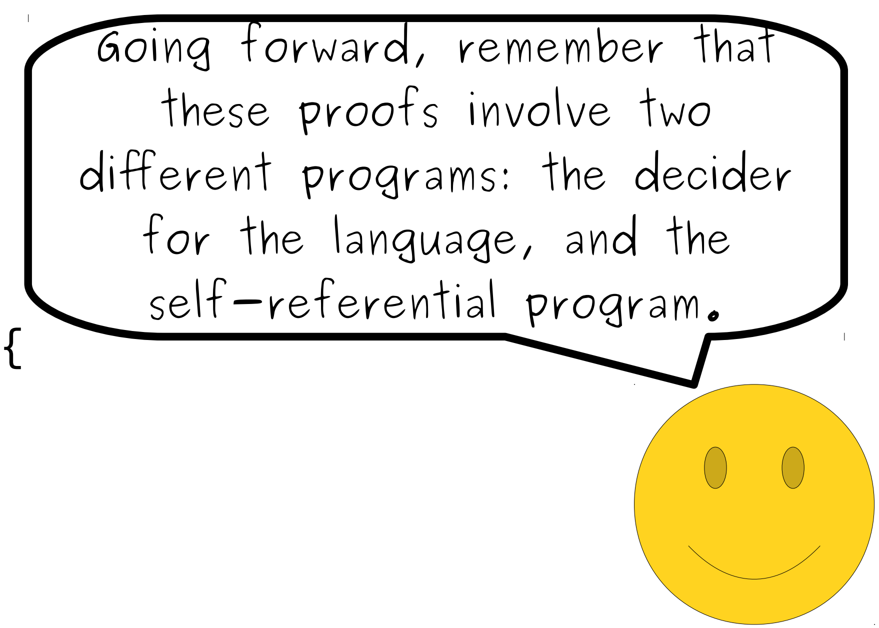

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

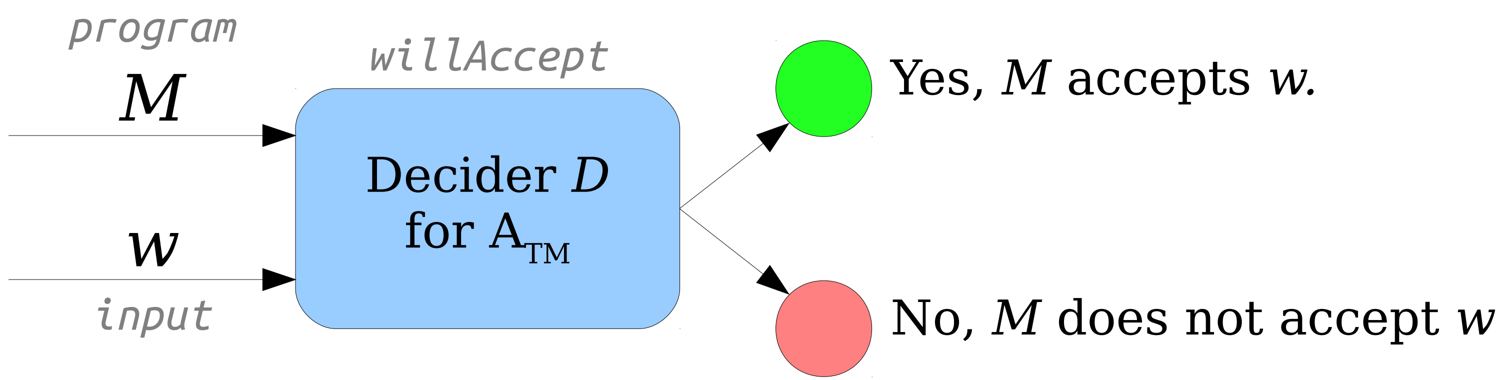
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



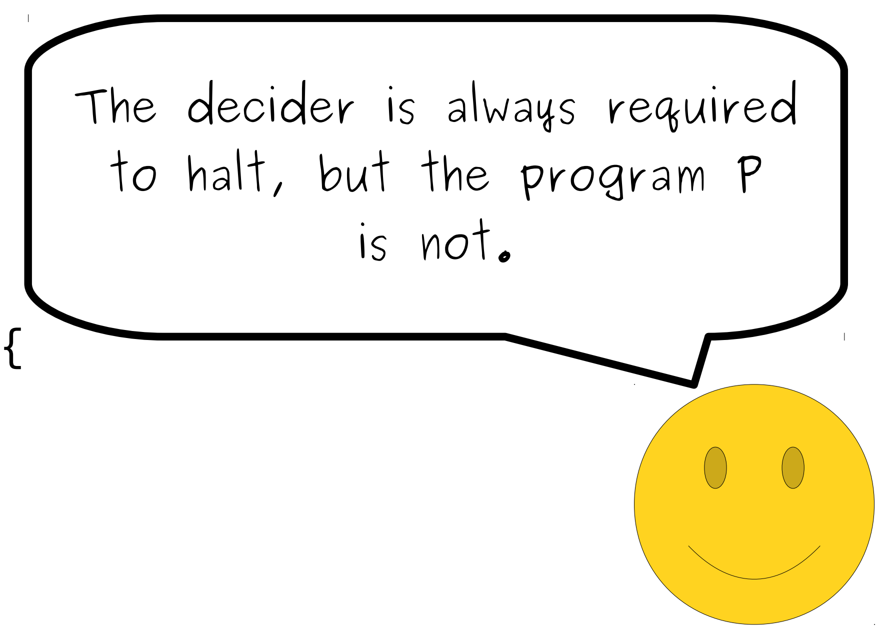

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

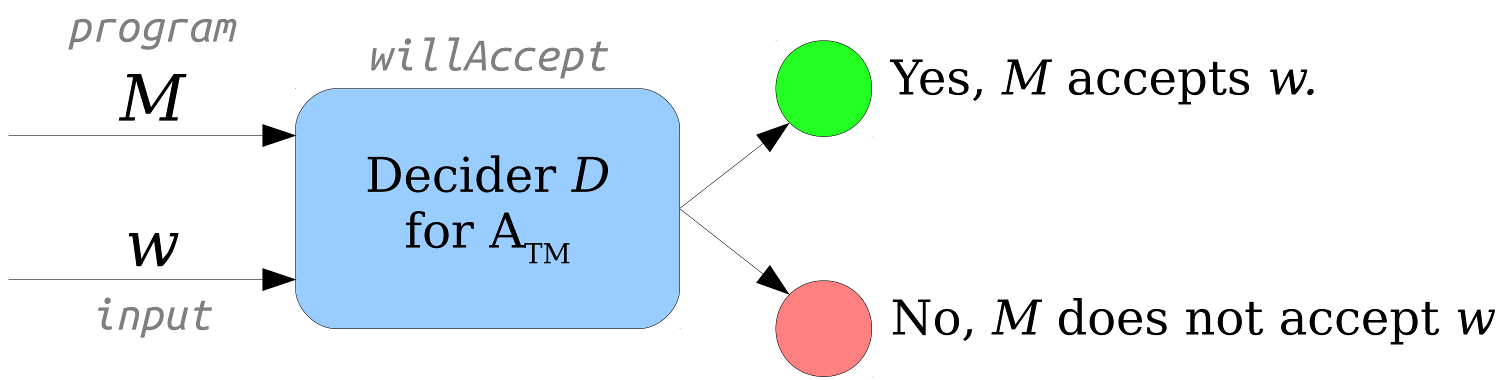
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



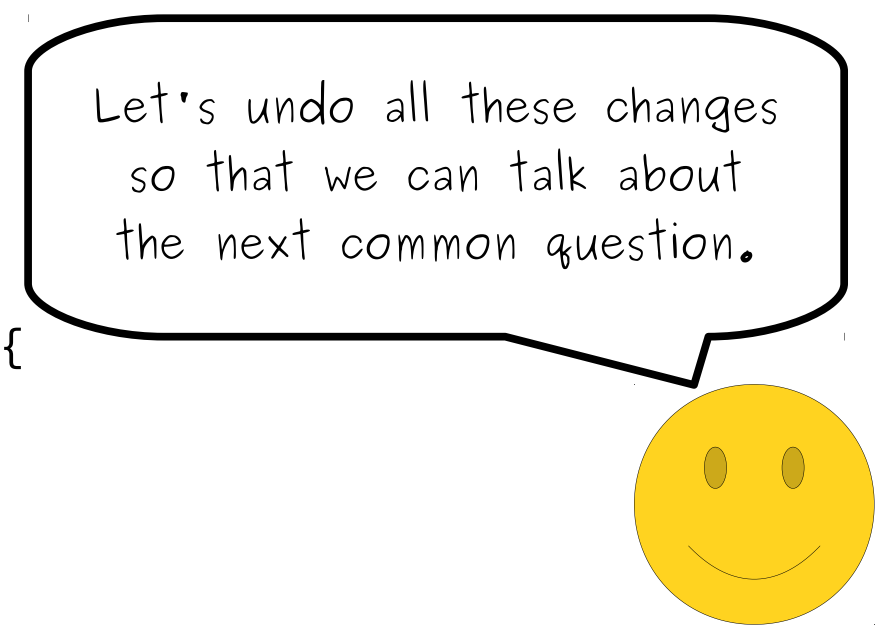

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    while (true) { }
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

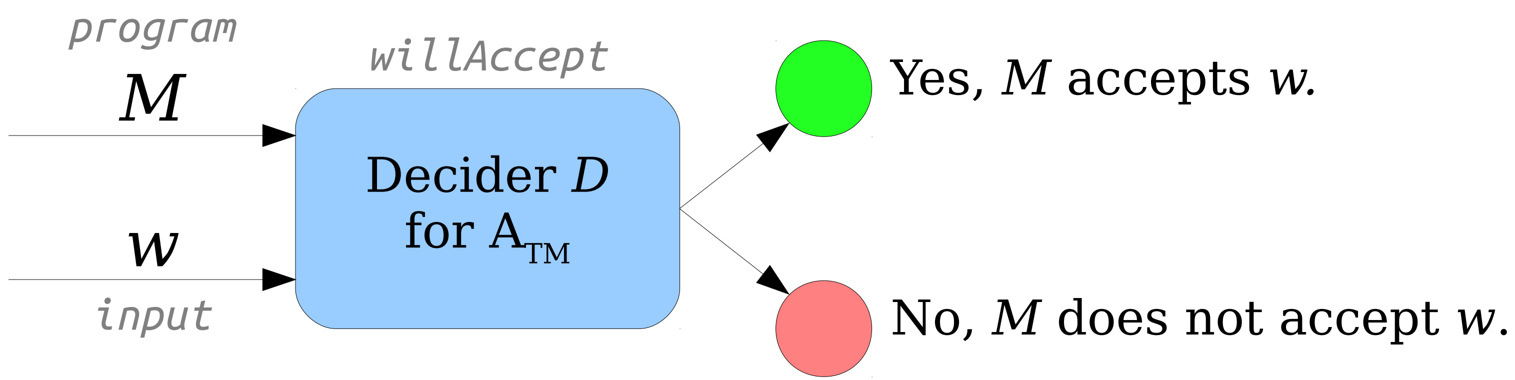
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



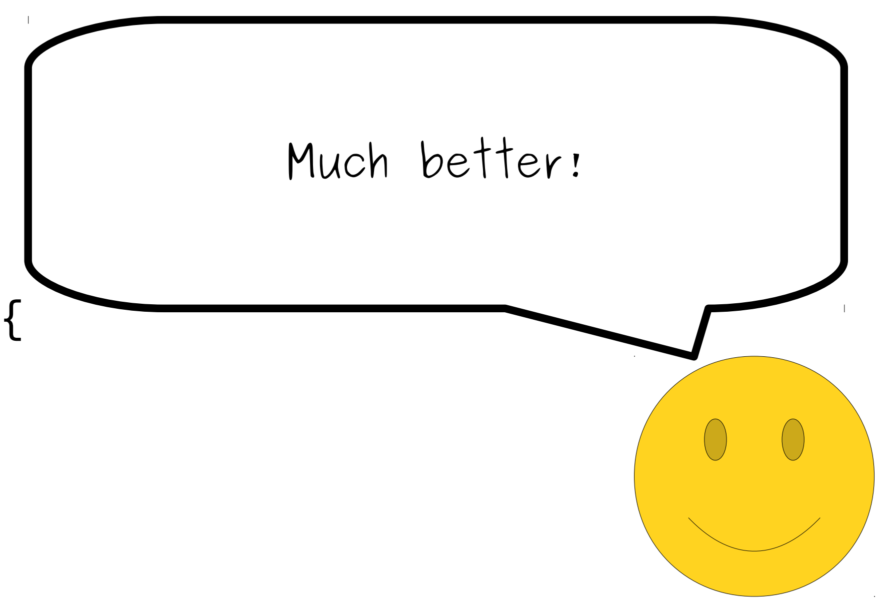

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

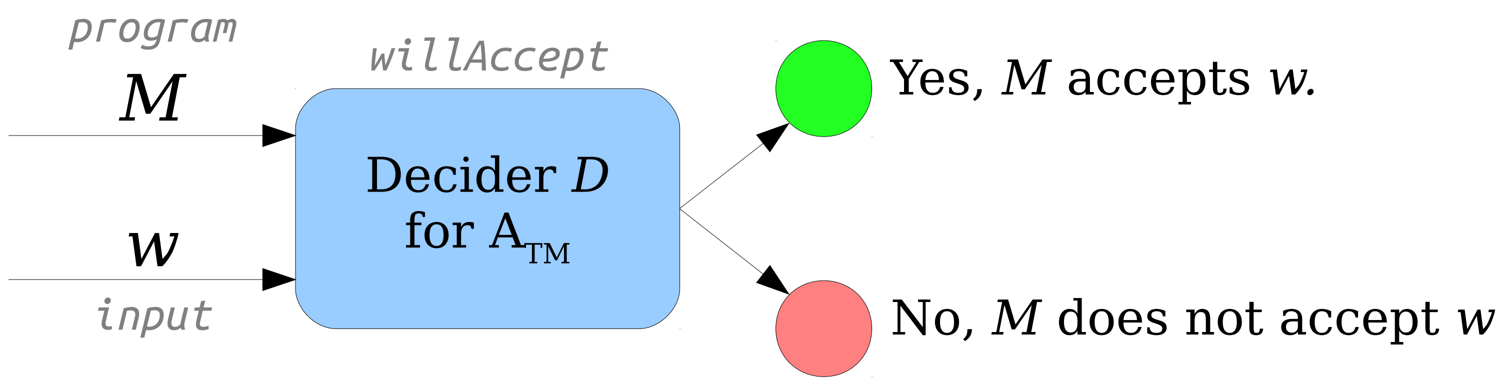
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



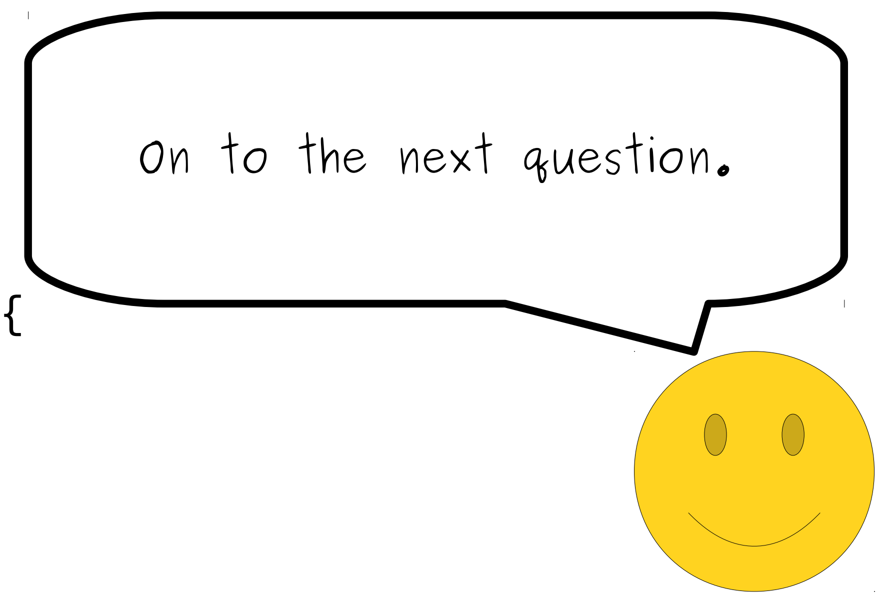

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

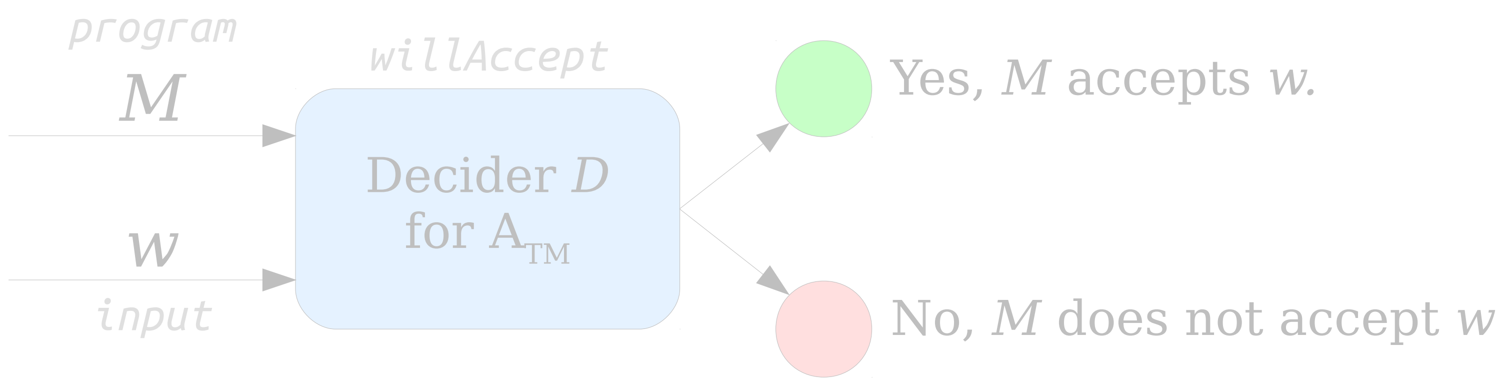
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



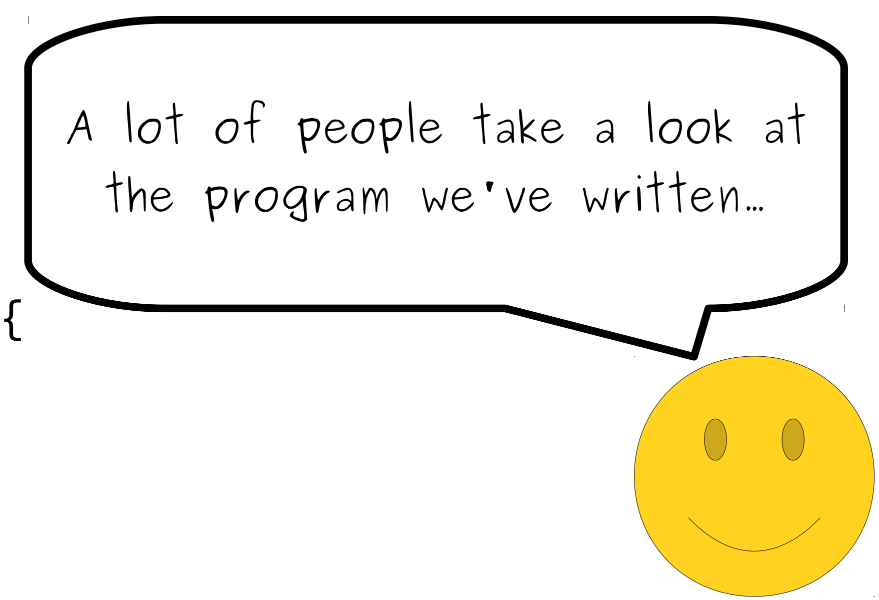

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

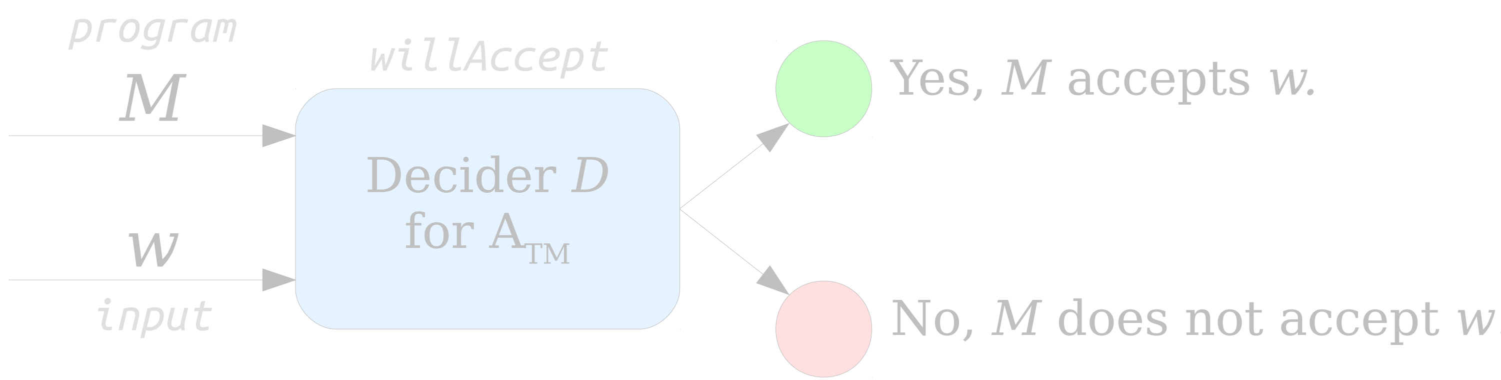
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



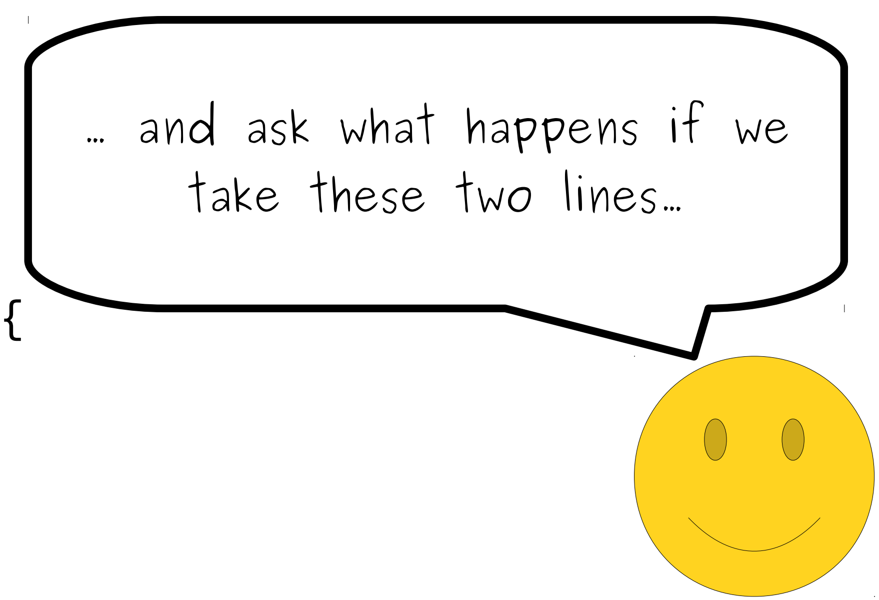

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

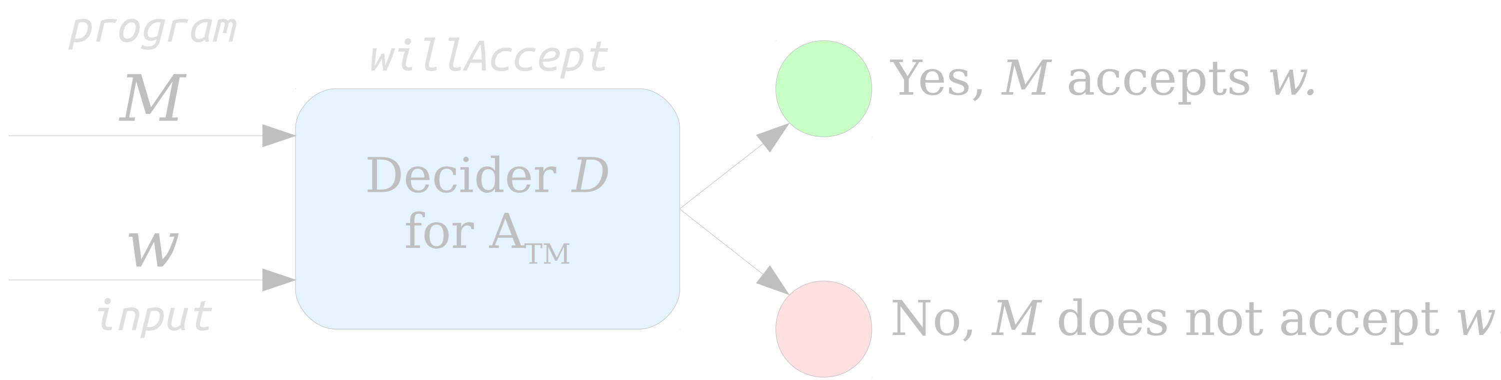
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



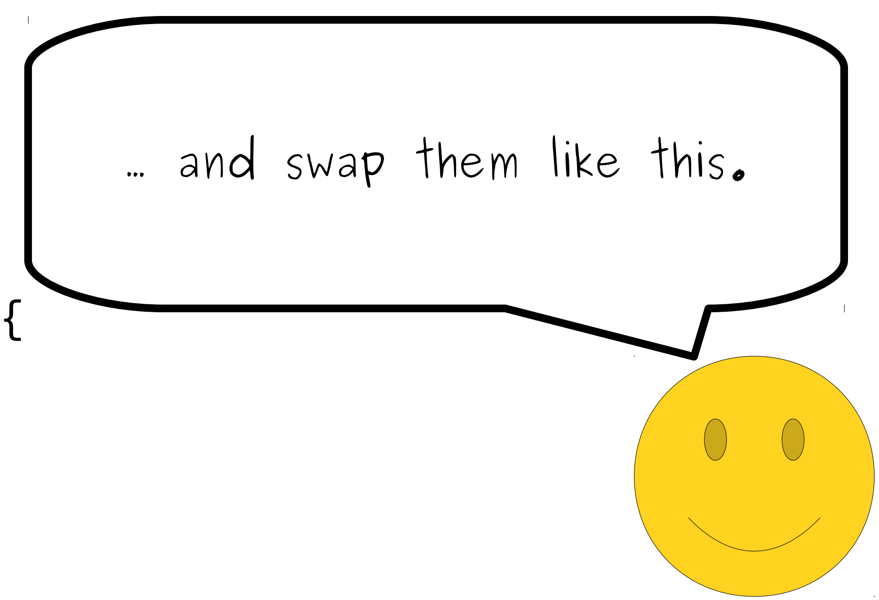

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

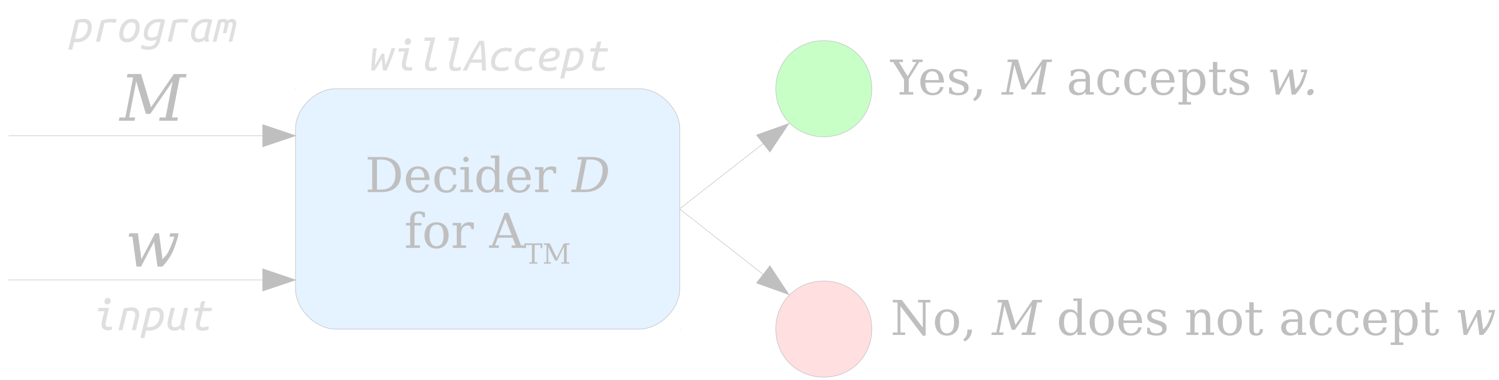
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



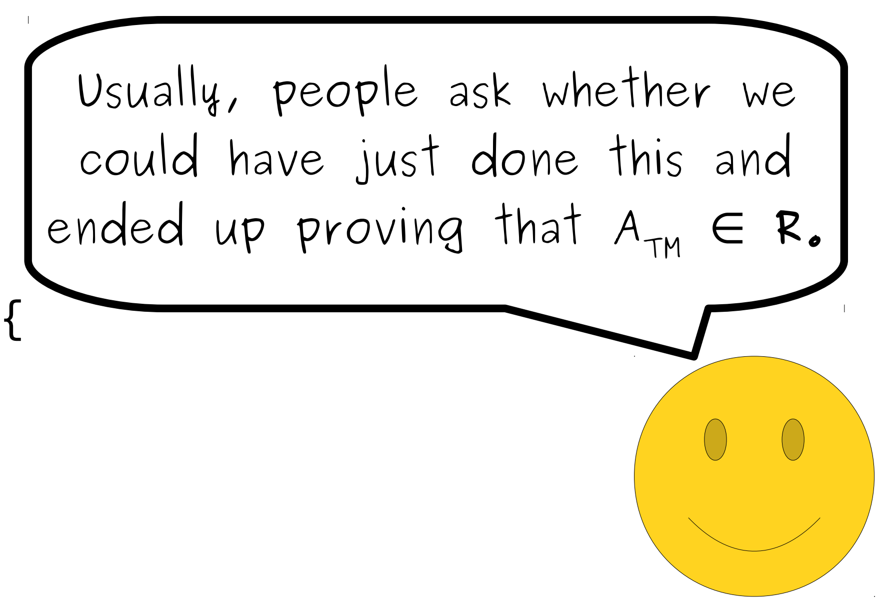

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

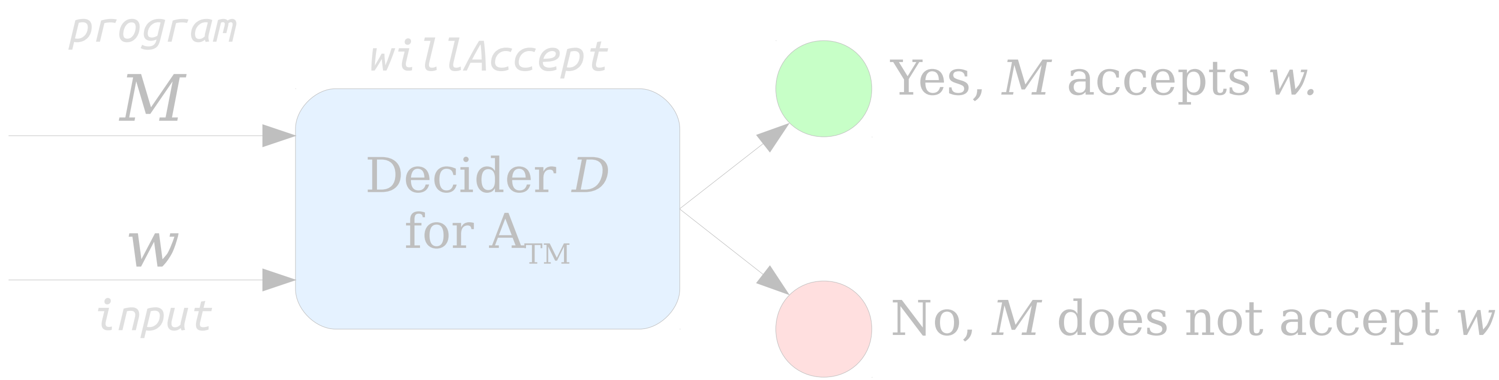
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

Contradiction!



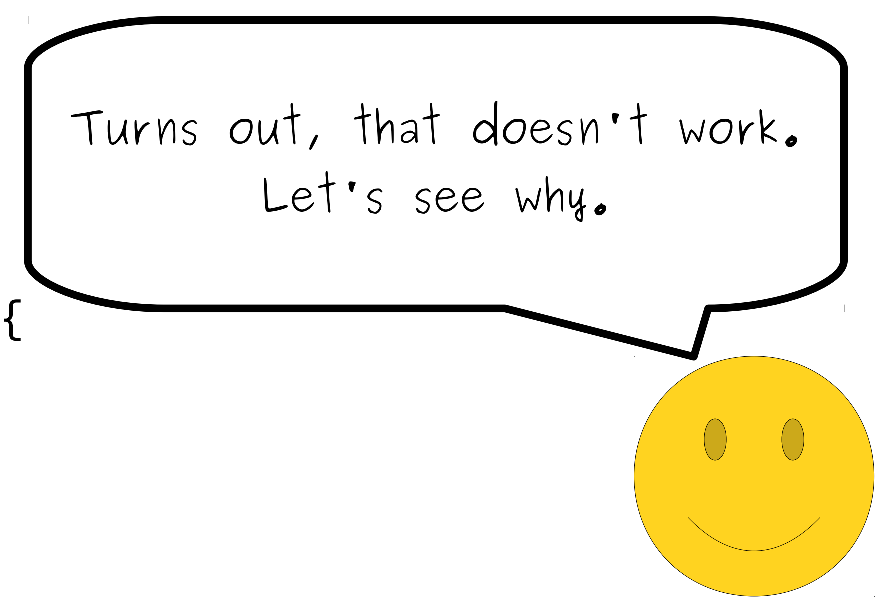

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



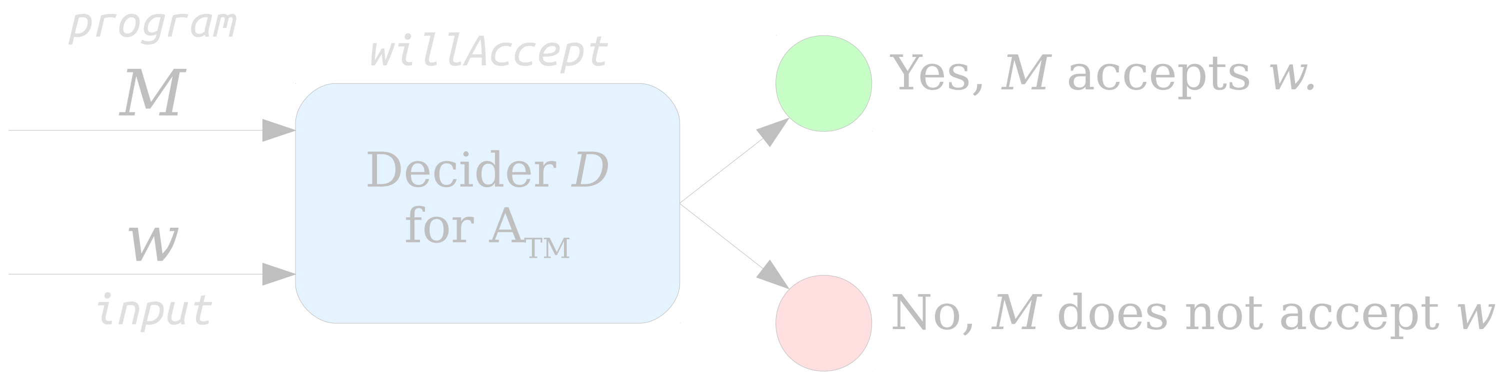
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

Contradiction!

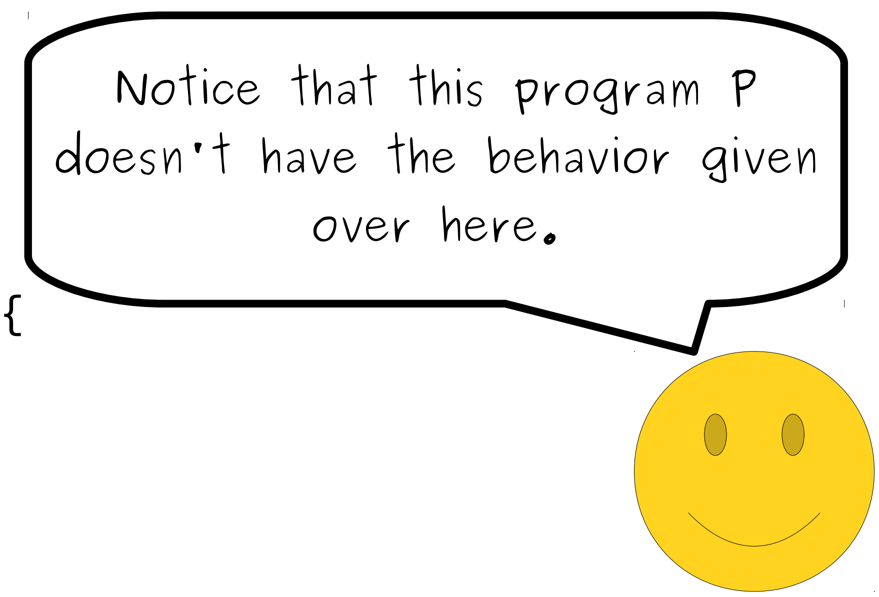



```
bool willAccept(string program, string input)
```

Program P design specification:

- ✓ If P accepts its input, then P does not accept its input.
- ✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

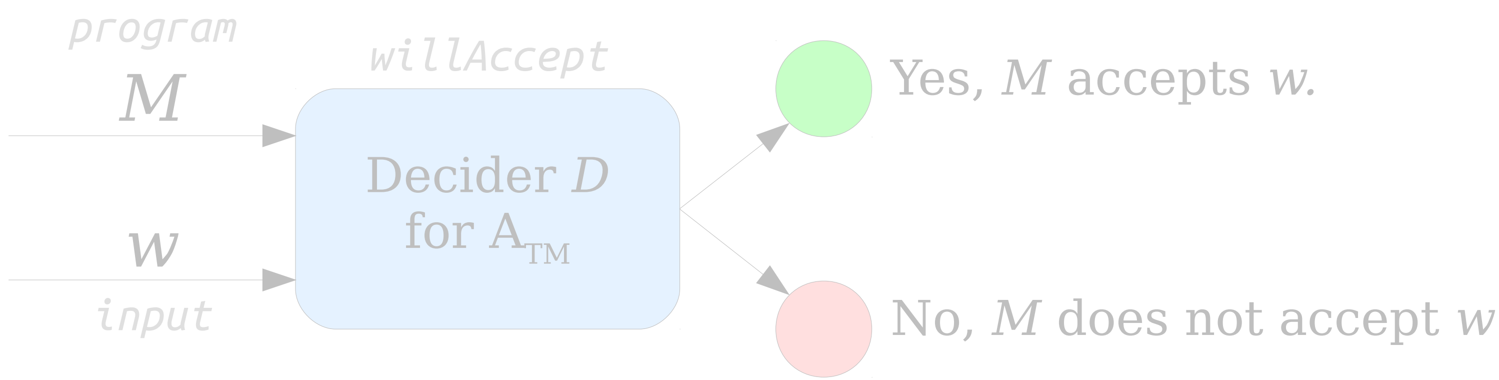
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does accept its input

↓

Contradiction!



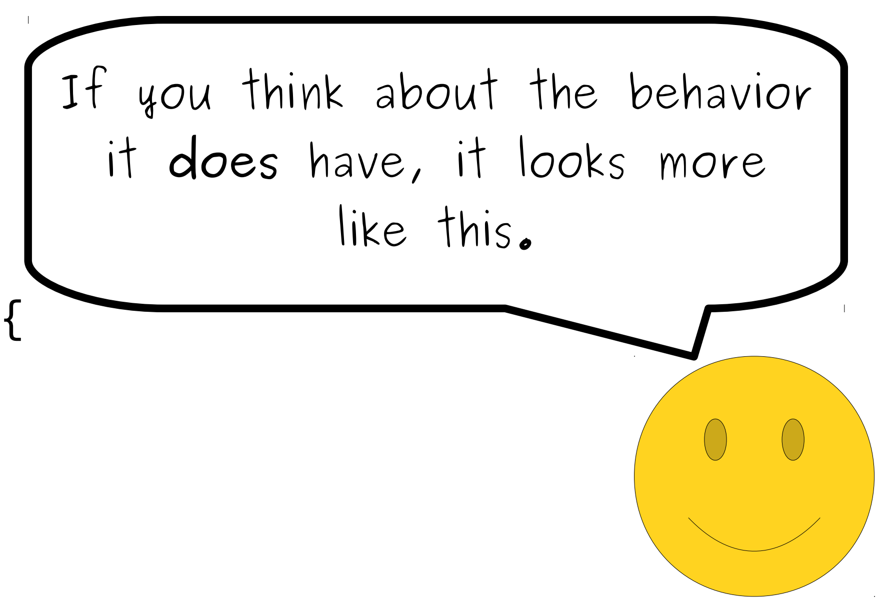

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

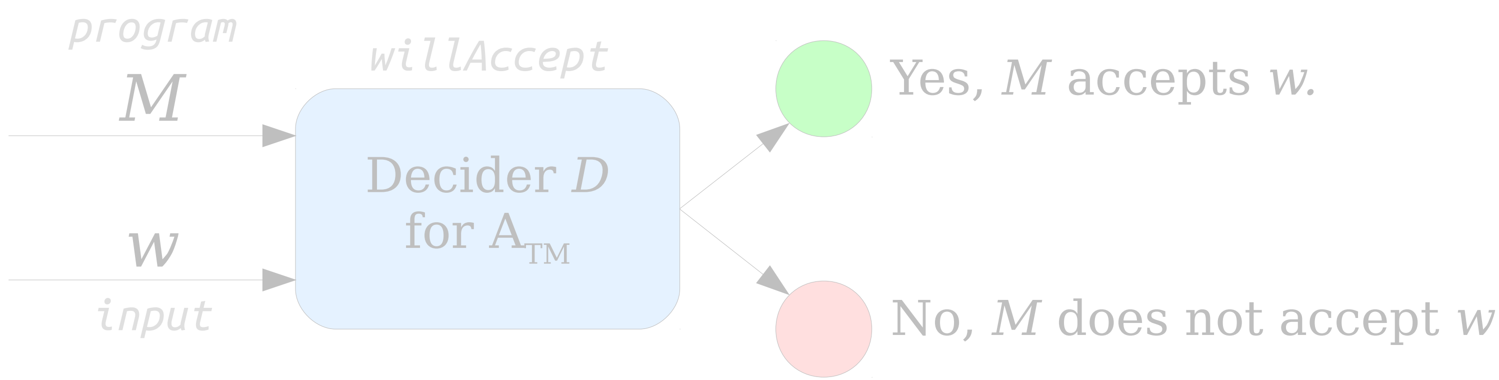
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does accept its input

↓

Contradiction!



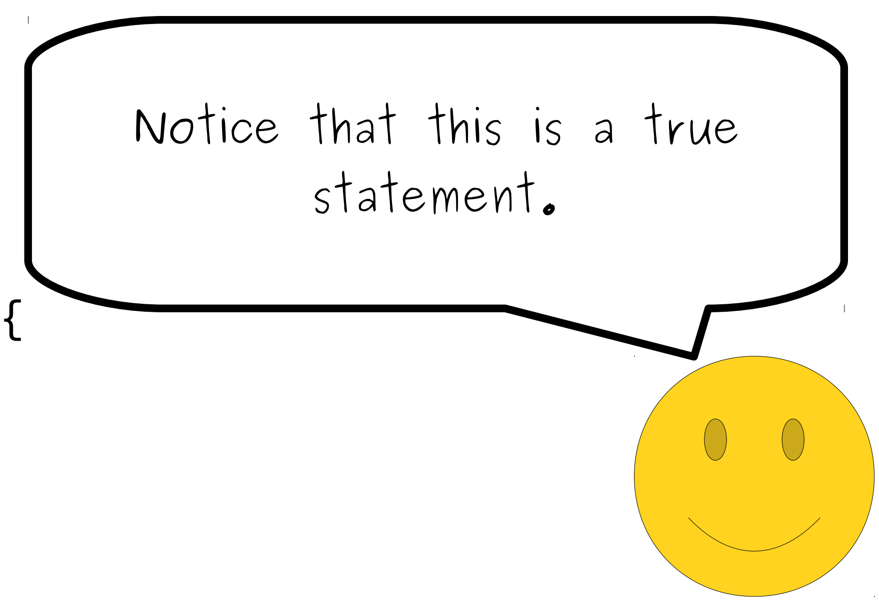

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then *P* does not accept its input.

✓ If *P* does not accept its input, then *P* accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

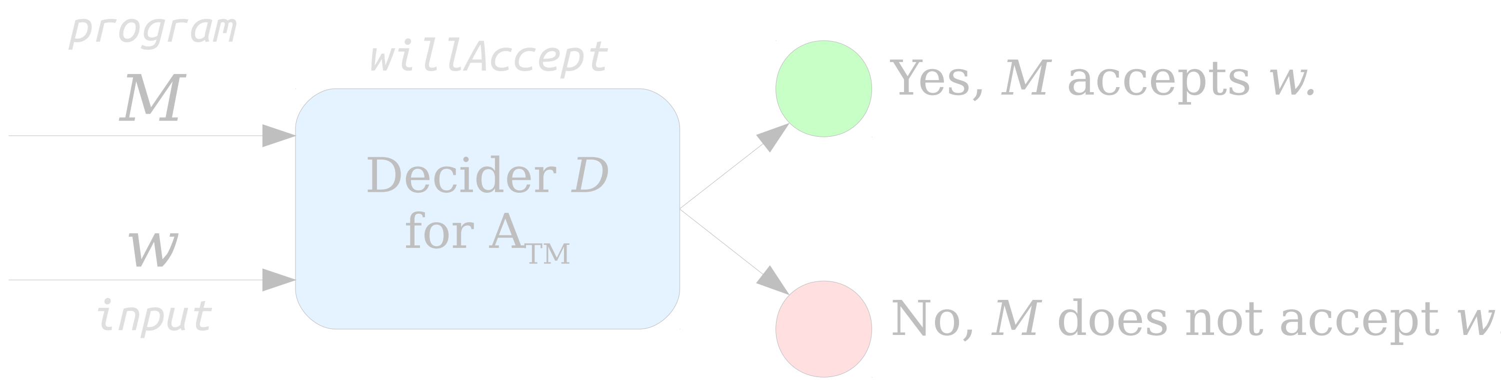
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does accept its input

↓

Contradiction!



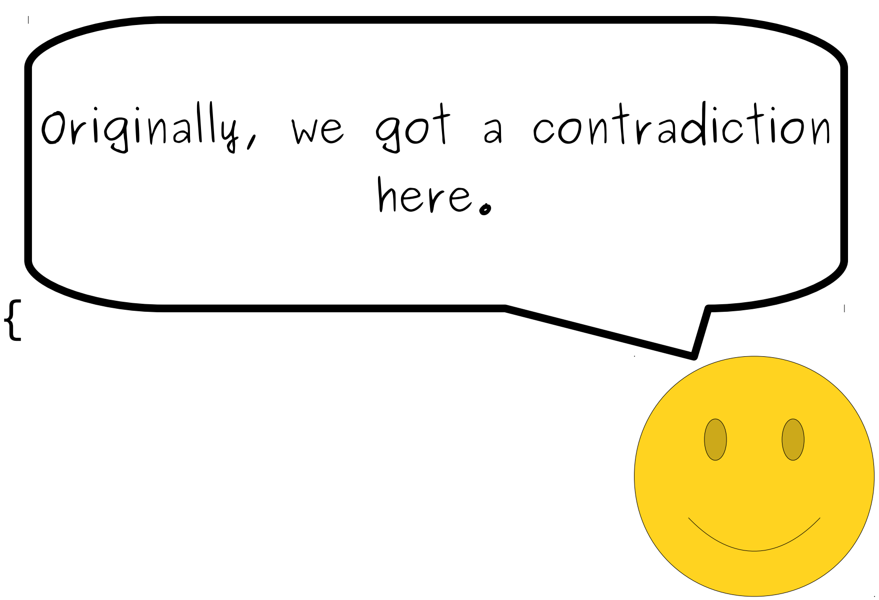

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

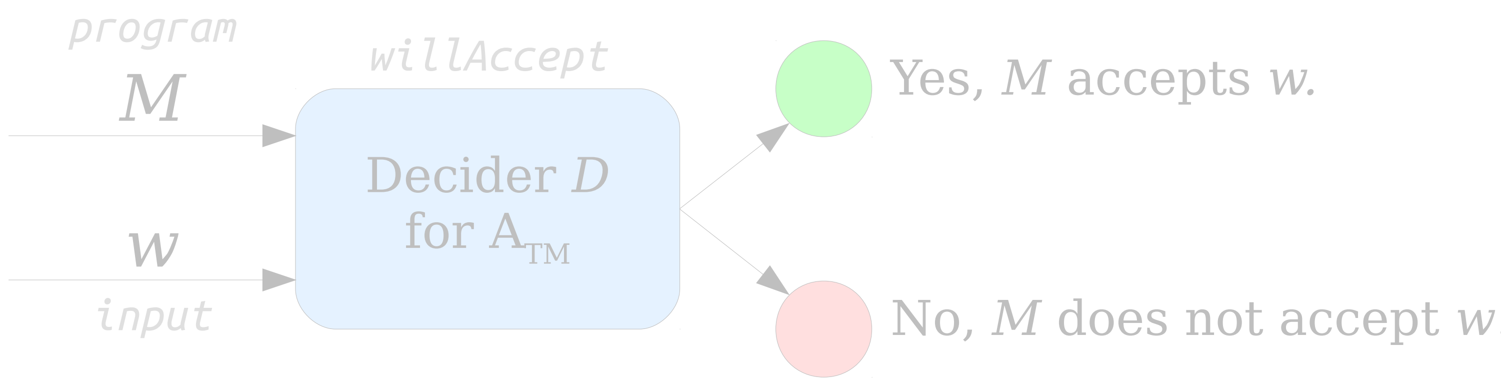
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does accept its input

↓

$\top$



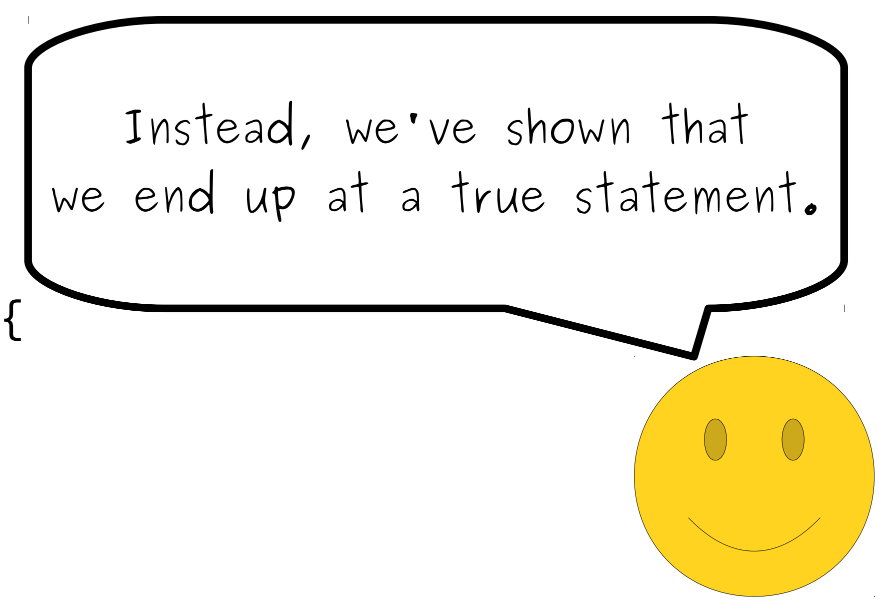

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

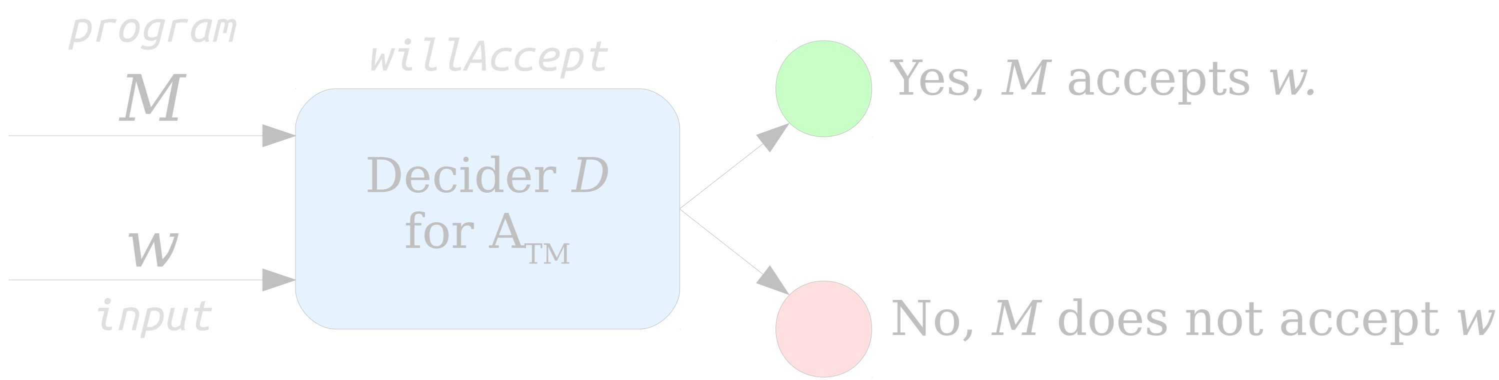
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does accept its input

↓

$\bot$



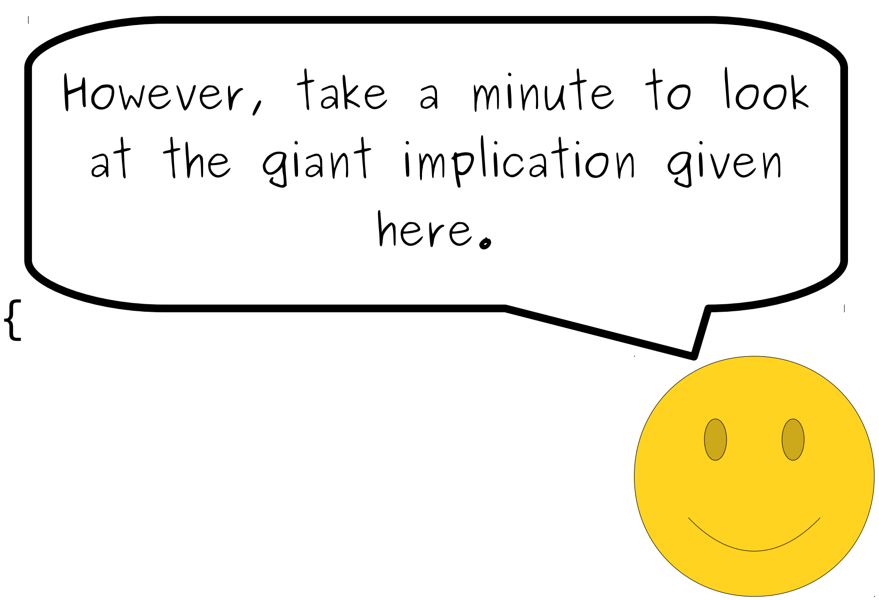

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

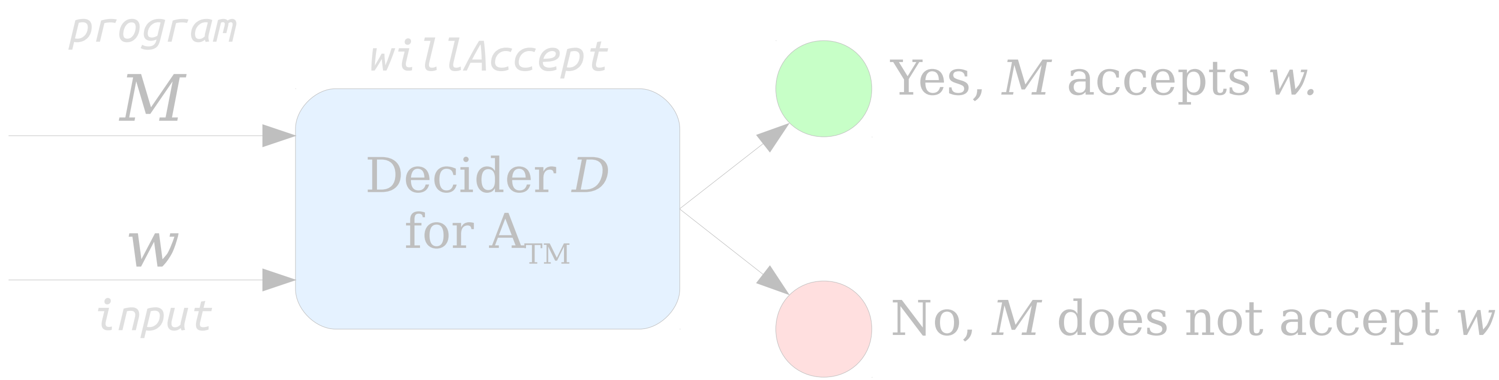
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

$\top$



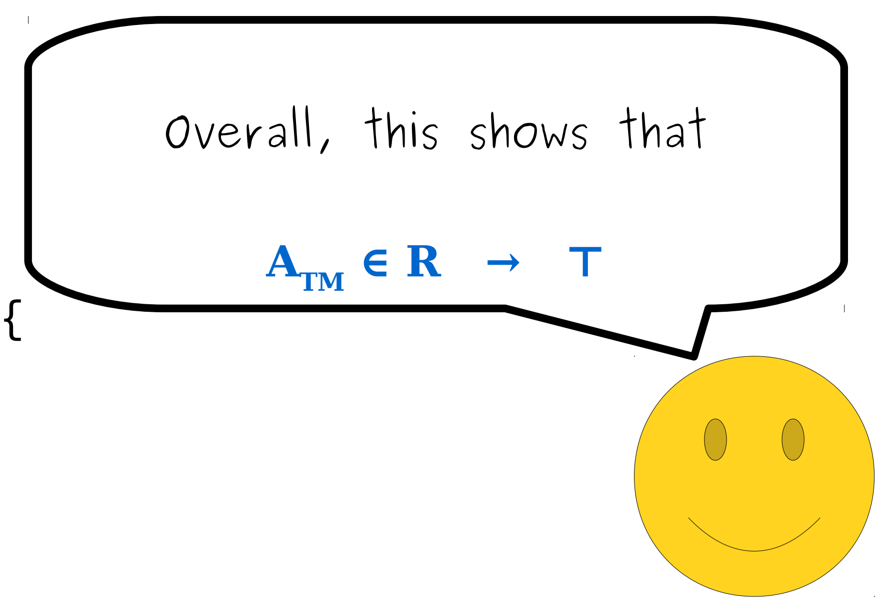

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

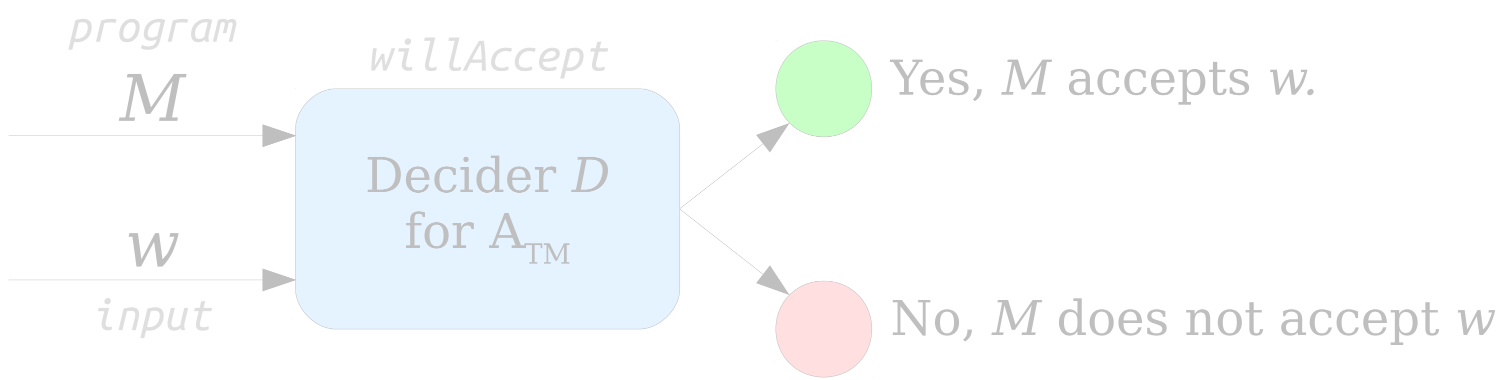
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

⊤

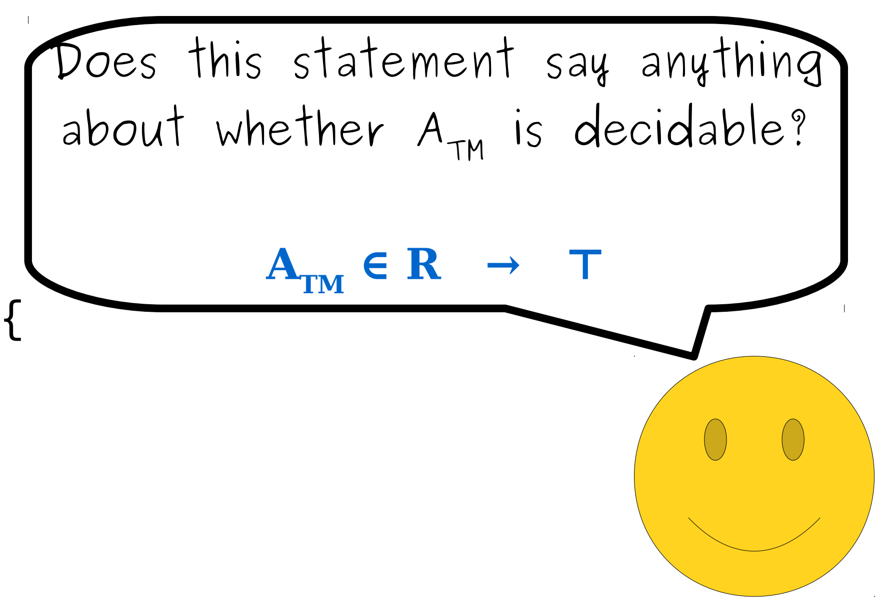



```
bool willAccept(string program, string input)
```

Program P design specification:

- ✓ If P accepts its input, then P does not accept its input.
- ✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

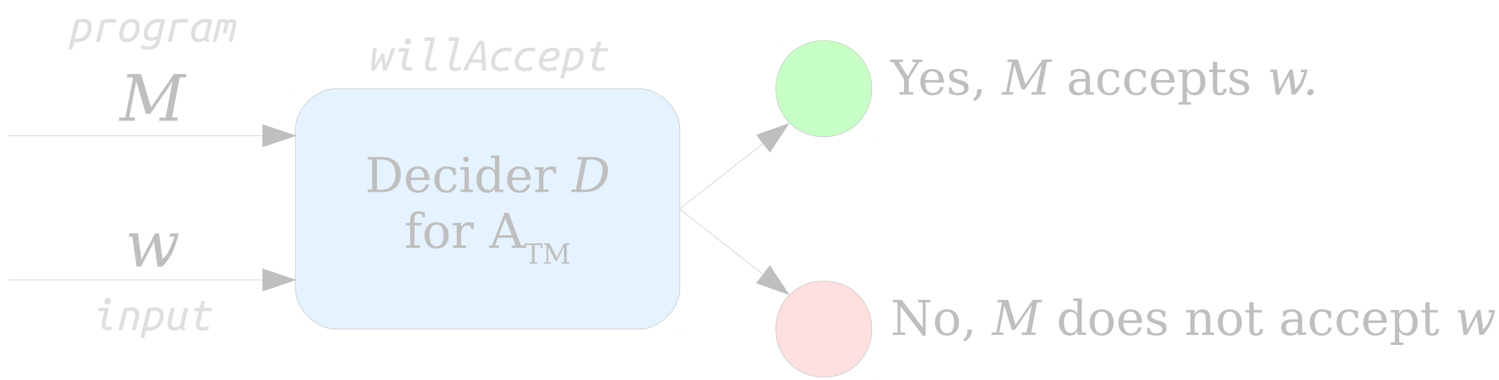
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

⊤



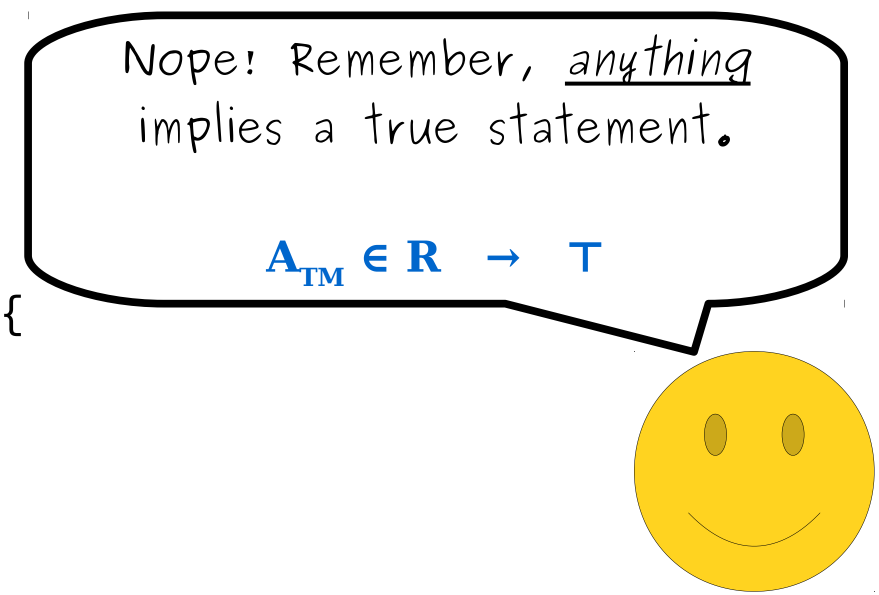

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then P does not accept its input.

✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



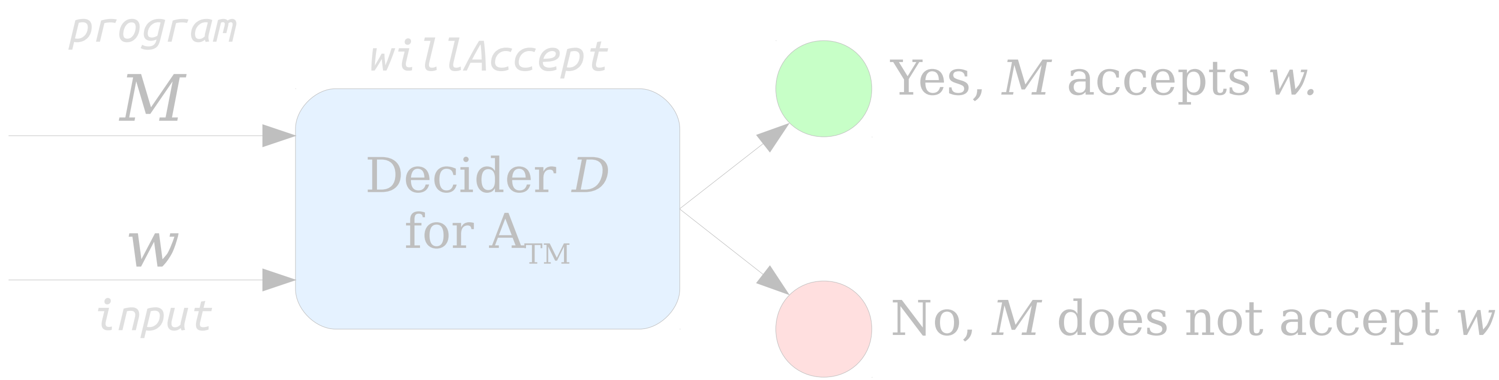
$A_{TM} \in \mathbf{R}$

There is a decider D for A_{TM}

We can write programs that use D as a helper method

Program P accepts its input if and only if program P does not accept its input

⊤

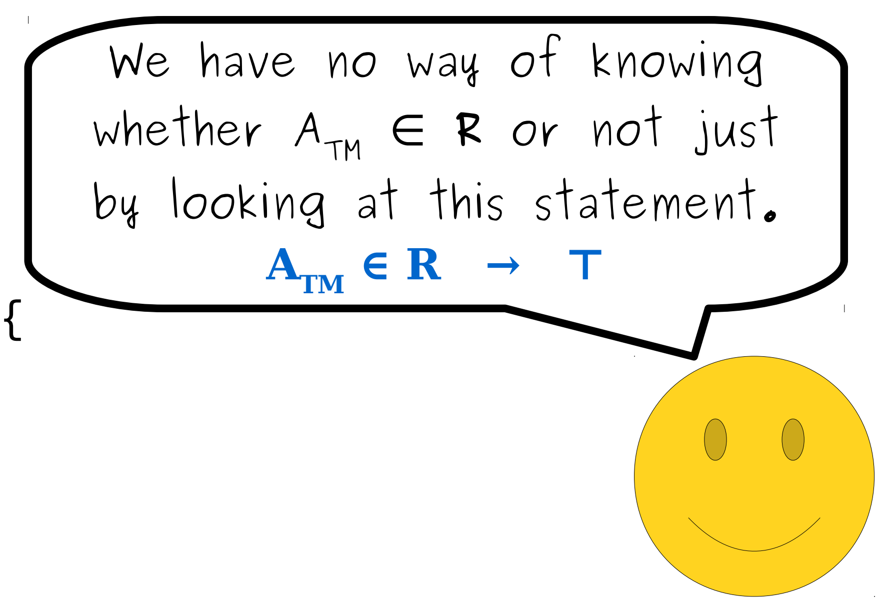



```
bool willAccept(string program, string input)
```

Program P design specification:

- ✓ If P accepts its input, then P does not accept its input.
- ✓ If P does not accept its input, then P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider *D* for A_{TM}

↓

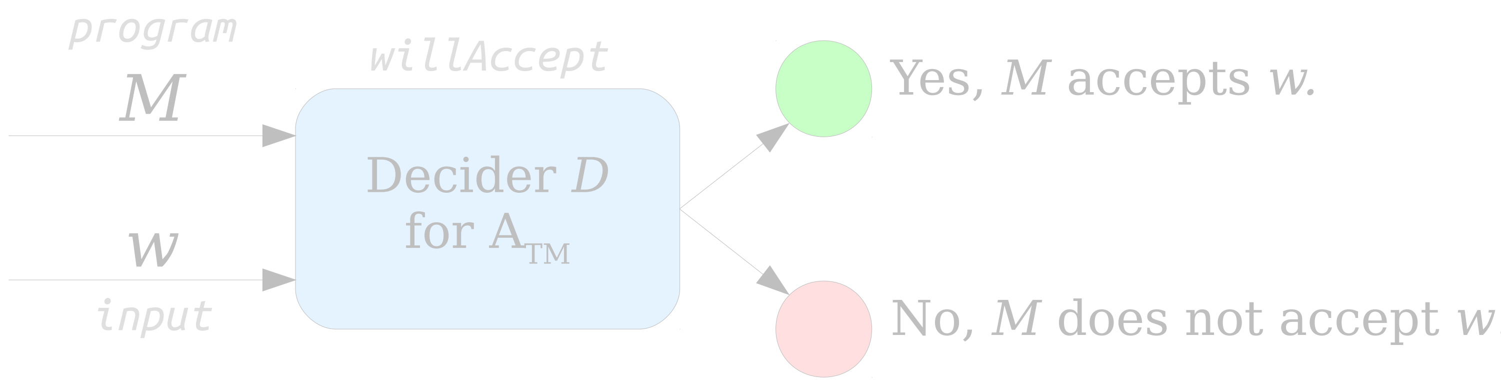
We can write programs that use *D* as a helper method

↓

Program *P* accepts its input if and only if program *P* does not accept its input

↓

$\bot$



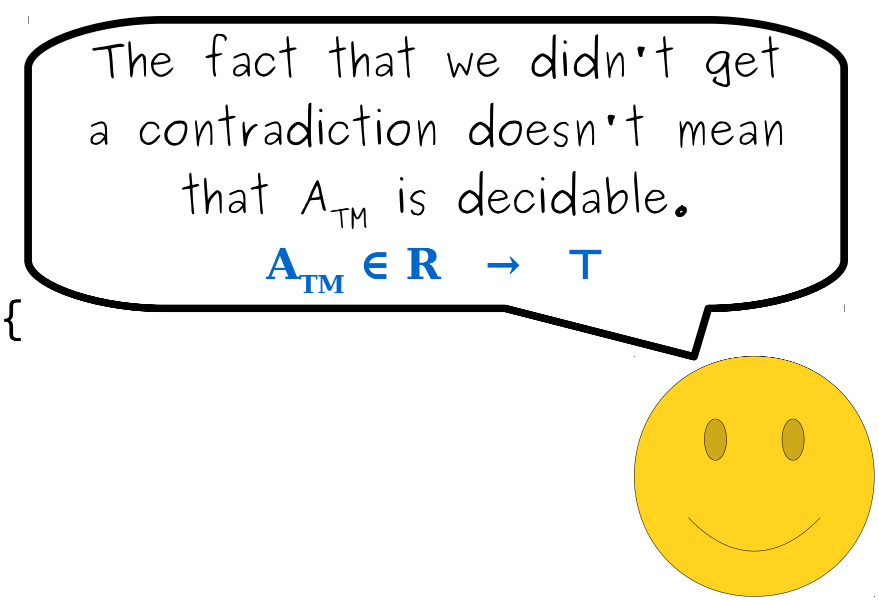

```
bool willAccept(string program, string input)
```

Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

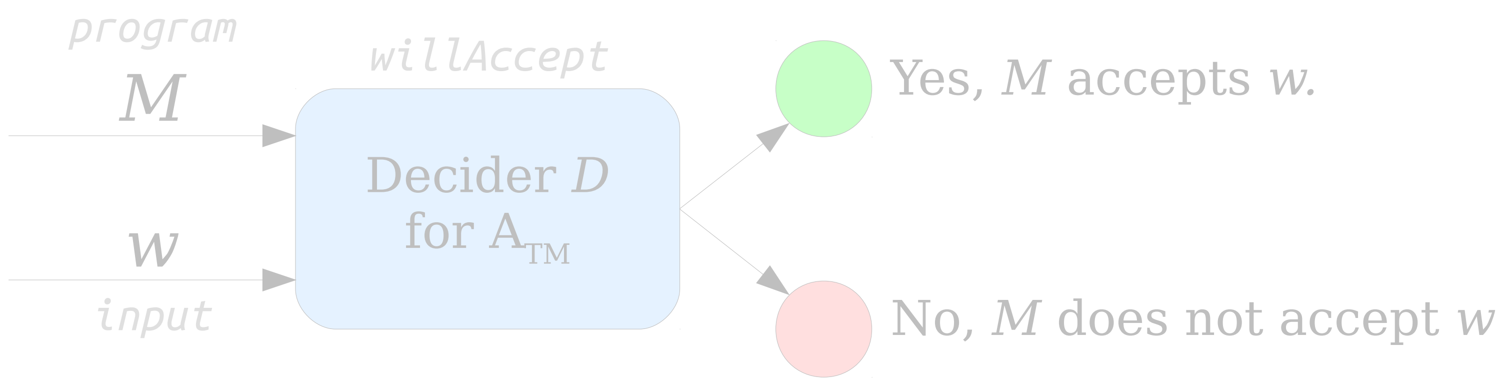
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does accept its input

↓

$\top$



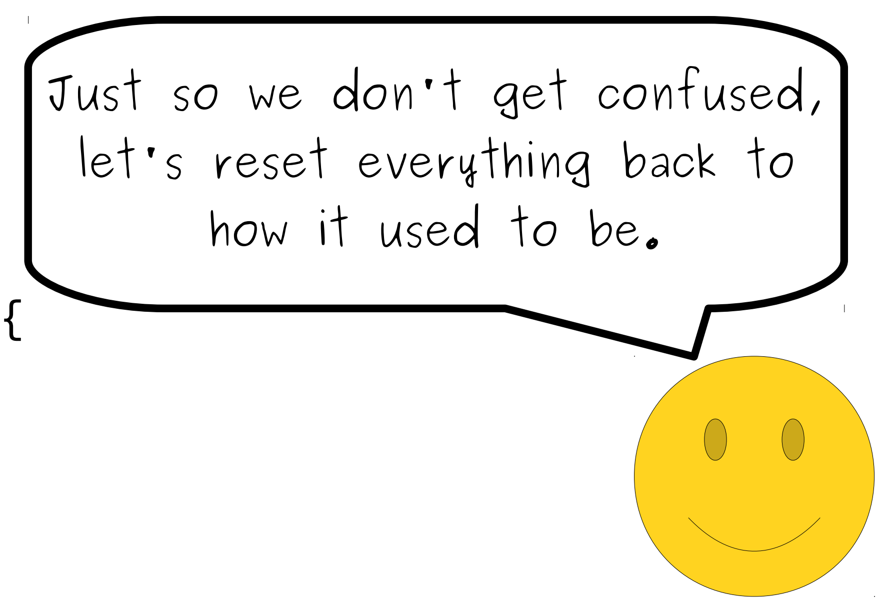

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
P does not accept its input.

✓ If P does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    accept();
  } else {
    reject();
  }
}
```



$A_{TM} \in \mathbf{R}$

↓

There is a decider D for A_{TM}

↓

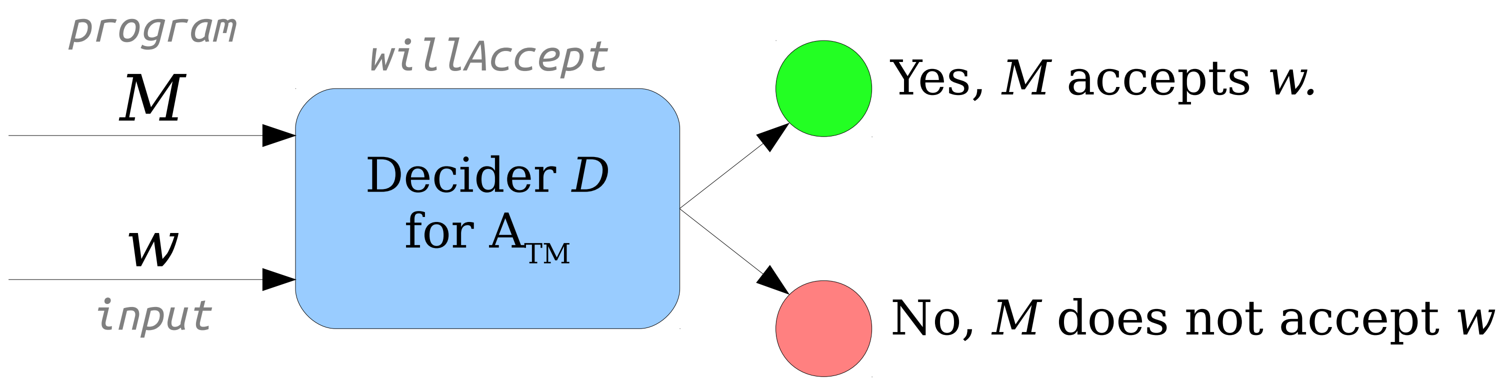
We can write programs that use D as a helper method

↓

Program P accepts its input if and only if program P does not accept its input

↓

Contradiction!



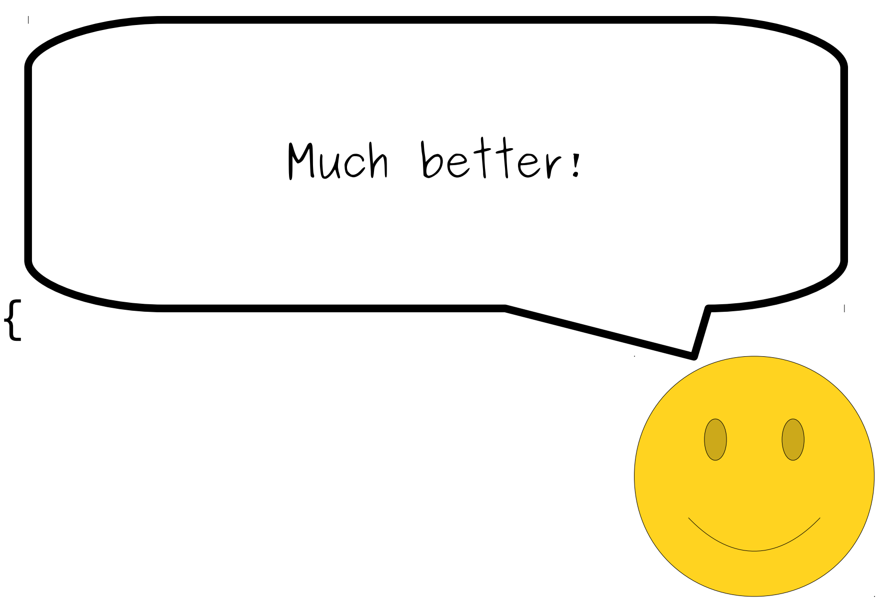

```
bool willAccept(string program, string input)
```

Program P design specification:

✓ If P accepts its input, then
  P does not accept its input.

✓ If P does not accept its input, then
  P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```



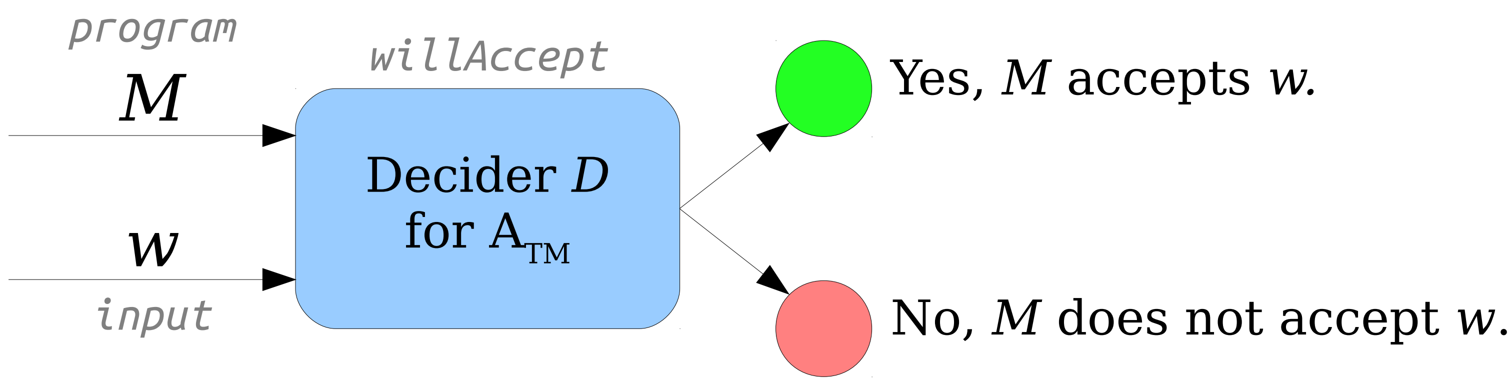
$A_{TM} \in \mathbf{R}$

There is a decider *D* for A_{TM}

We can write programs that use *D* as a helper method

Program *P* accepts its input if and only if program *P* does not accept its input

Contradiction!



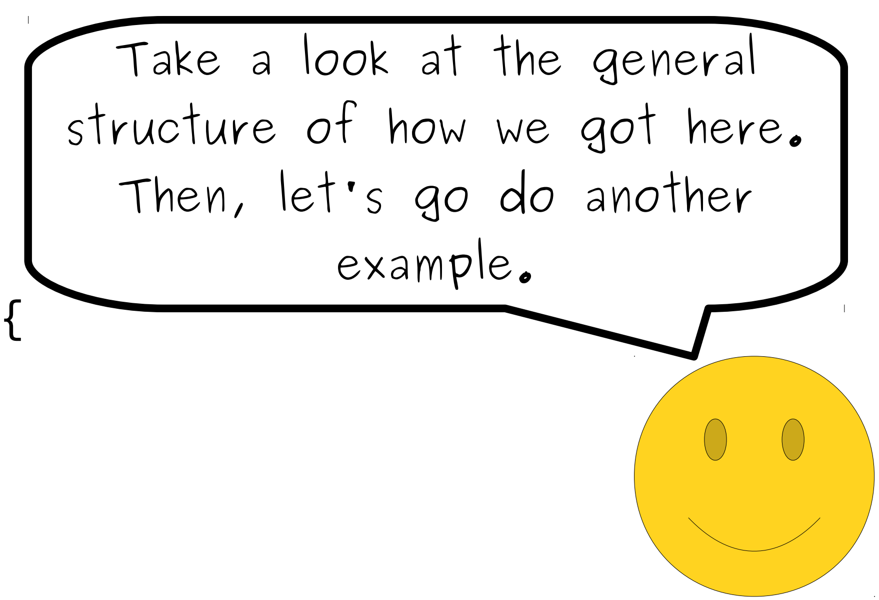

```
bool willAccept(string program, string input)
```

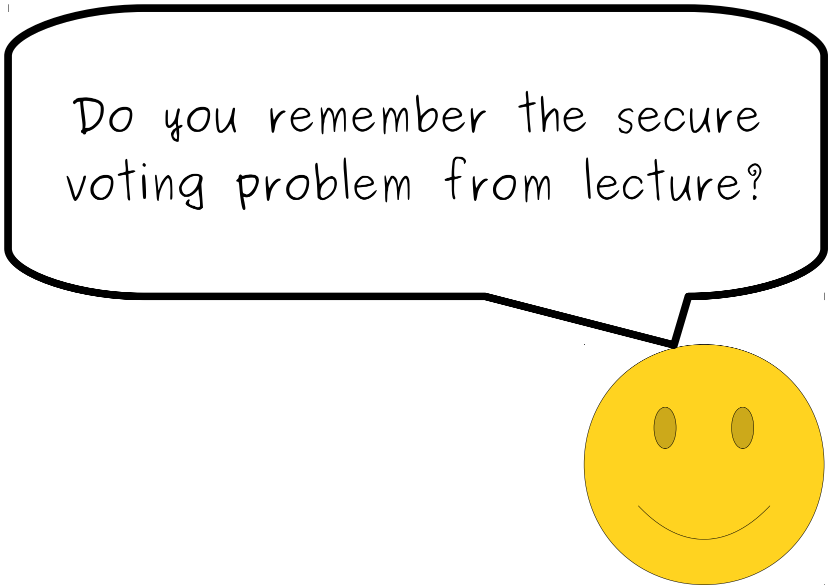
Program *P* design specification:

✓ If *P* accepts its input, then
P does not accept its input.

✓ If *P* does not accept its input, then
P accepts its input.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (willAccept(me, input)) {
    reject();
  } else {
    accept();
  }
}
```

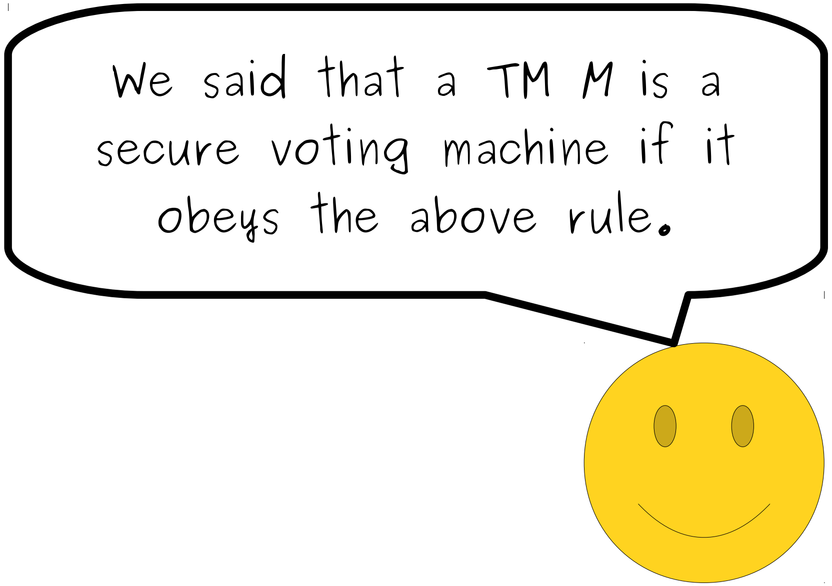



Do you remember the secure voting problem from lecture?

M is a secure voting machine

if and only if

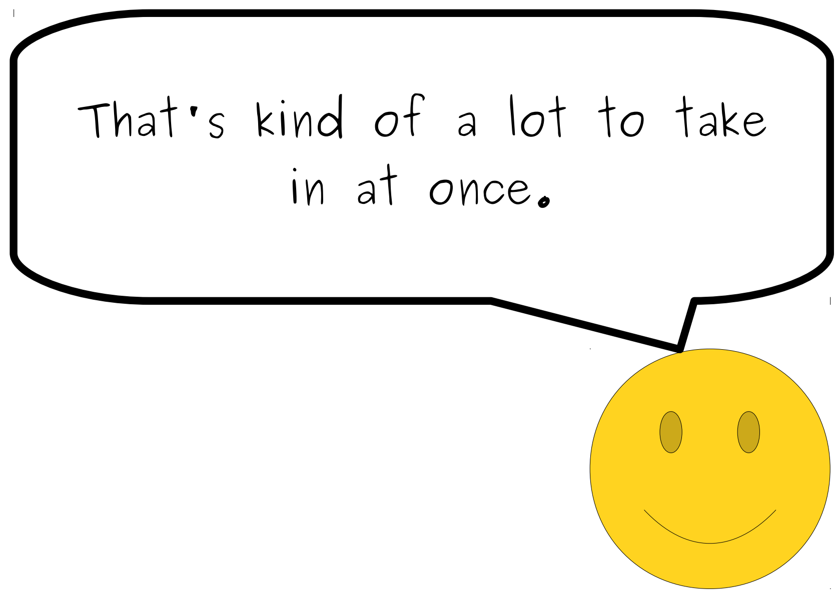
$\mathscr{L}(M) = \{ w \in \{\texttt{r}, \texttt{d}\}^* \mid w$ has more $\texttt{r}$'s than $\texttt{d}$'s $\}$



M is a secure voting machine

if and only if

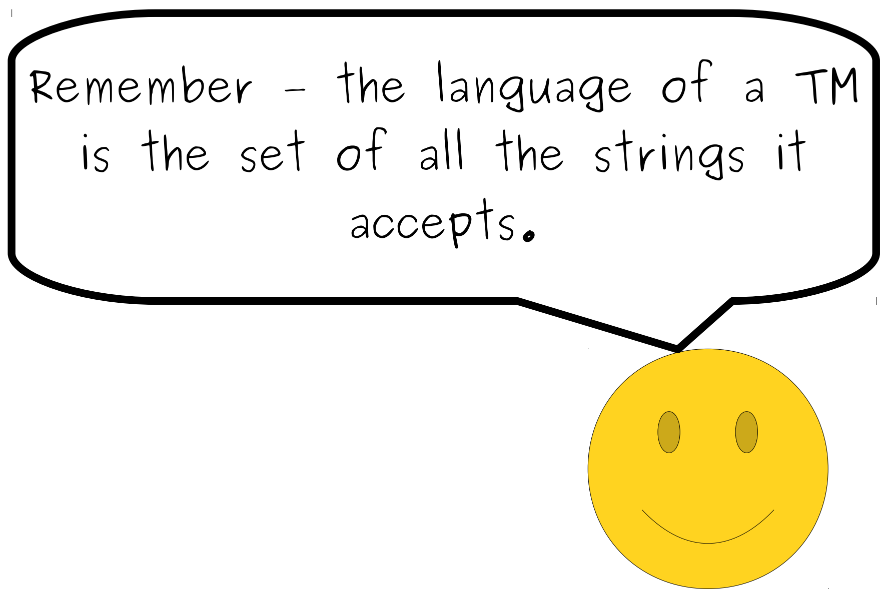
$$\mathscr{L}(M) = \{\ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s}\ \}$$



M is a secure voting machine

if and only if

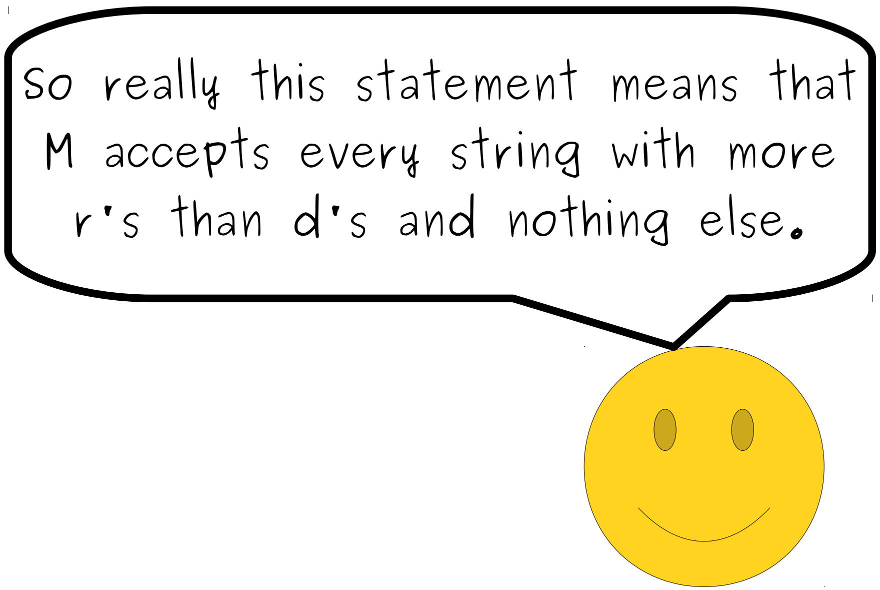
$\mathscr{L}(M) = \{\ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s}\ \}$



M is a secure voting machine

if and only if

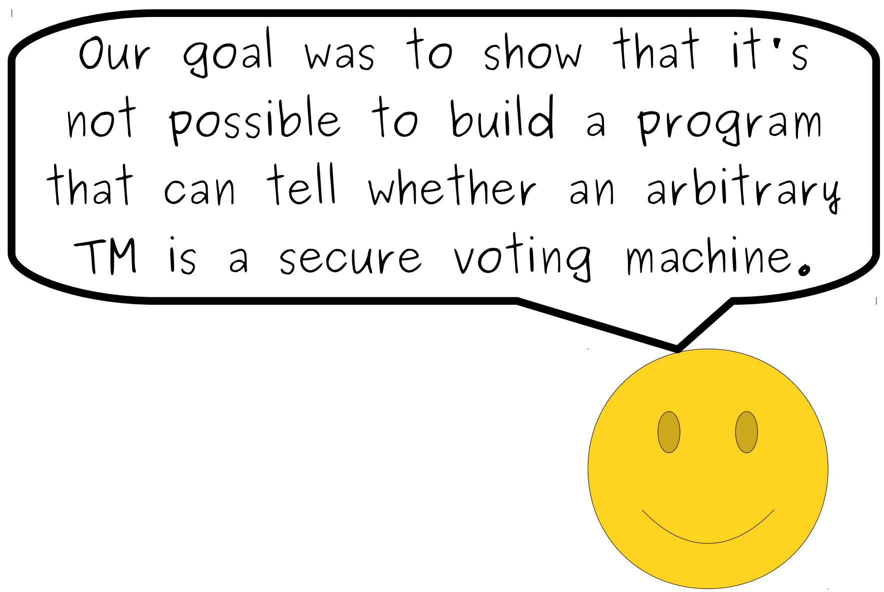
$\mathscr{L}(M) = \{ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s} \}$



M is a secure voting machine

if and only if

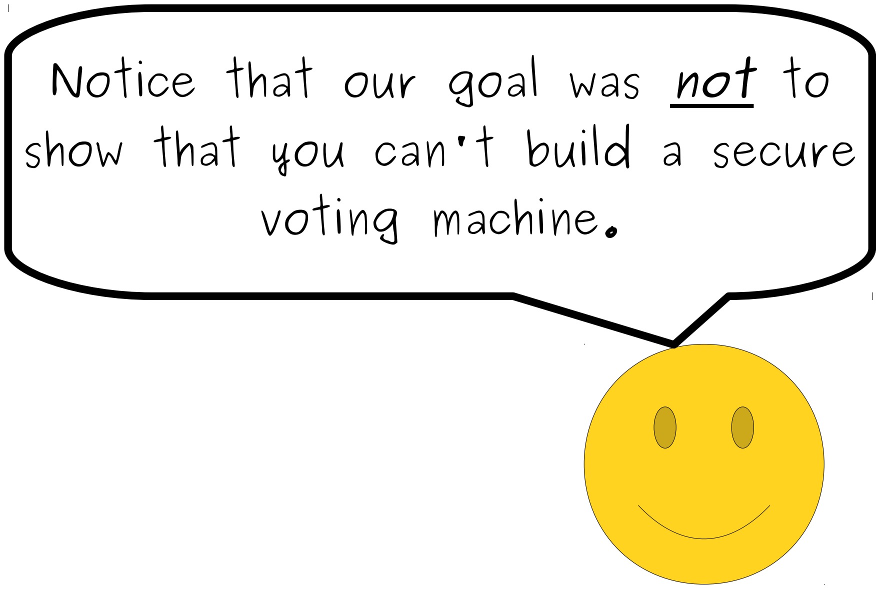
$\mathscr{L}(M) = \{ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s} \}$



M is a secure voting machine

if and only if

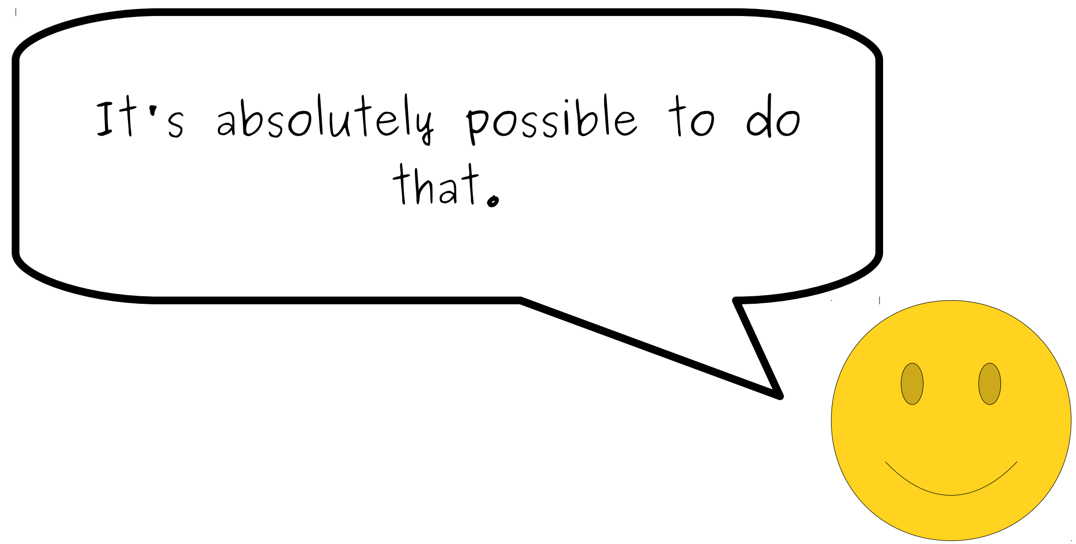
$\mathscr{L}(M) = \{ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s} \}$



M is a secure voting machine

if and only if

$\mathscr{L}(M) = \{ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s} \}$

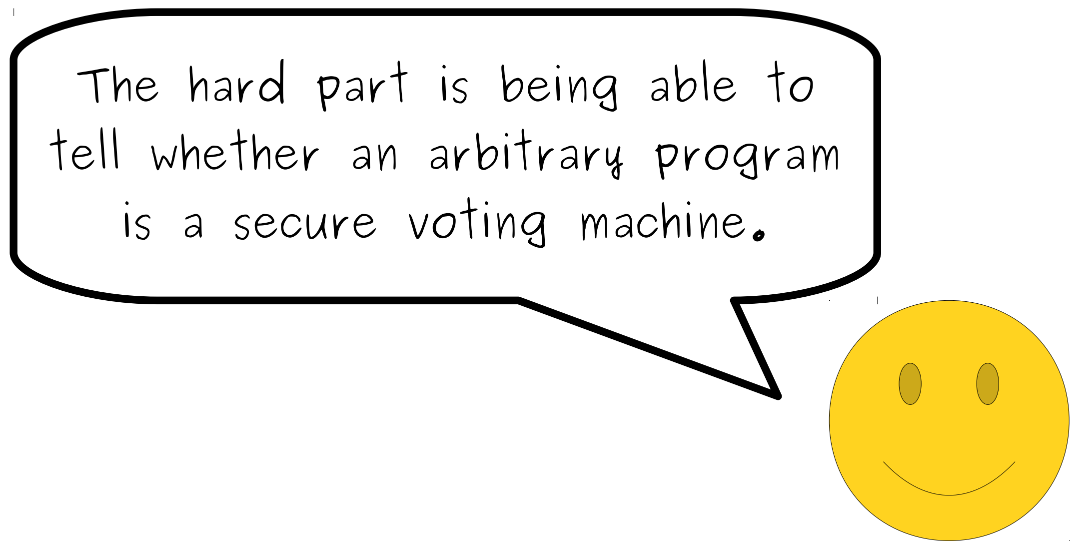



```
int main() {
  string input = getInput();
  if (countRs(input) > countDs(input)) {
    accept();
  } else {
    reject();
  }
}
```

M is a secure voting machine

if and only if

$\mathscr{L}(M) = \{\ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s}\ \}$

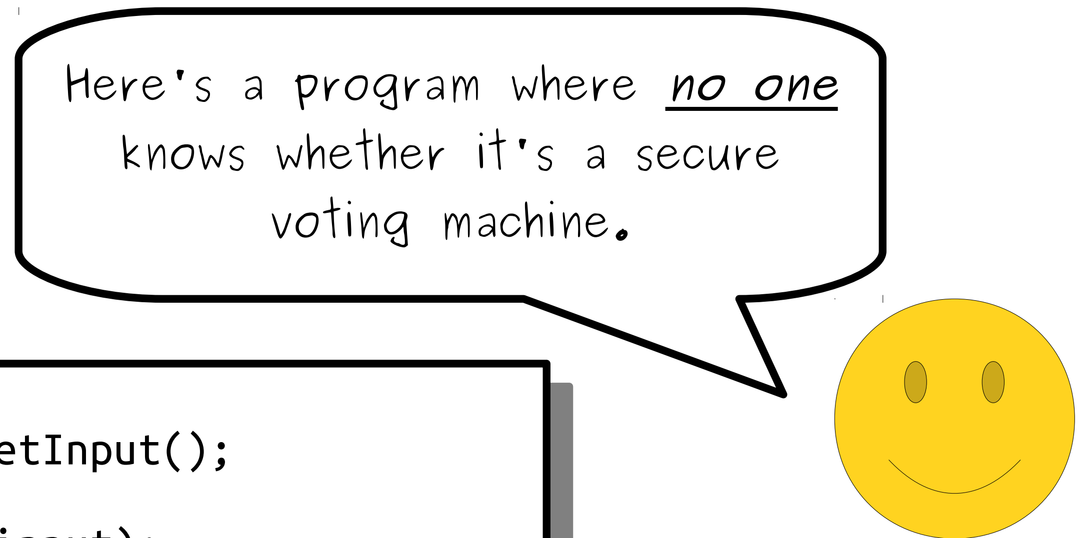



```
int main() {
  string input = getInput();
  if (countRs(input) > countDs(input)) {
    accept();
  } else {
    reject();
  }
}
```

M is a secure voting machine

if and only if

$\mathscr{L}(M)$ = { w ∈ {r, d}* | w has more r's than d's }

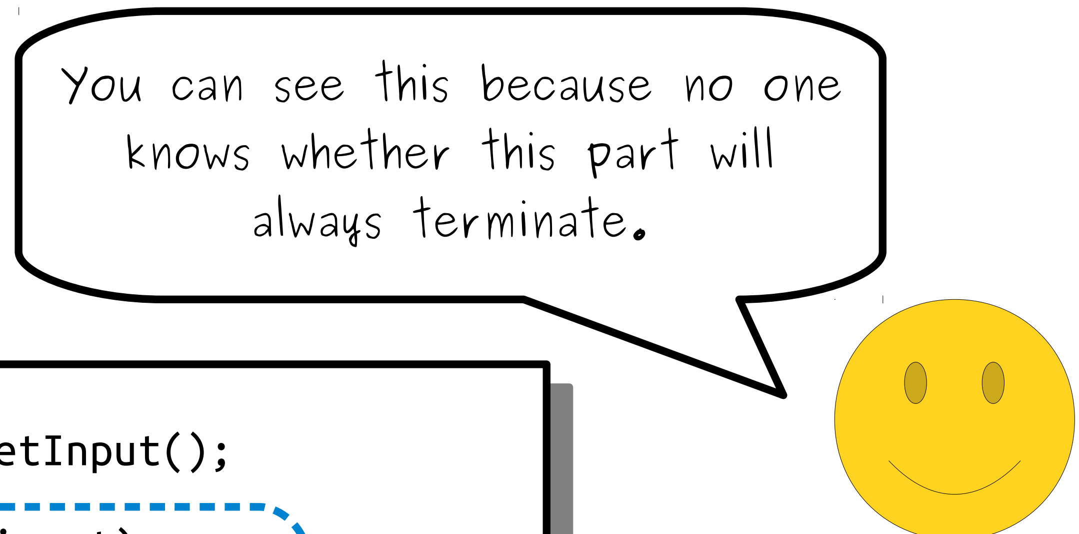



```
int main() {
  string input = getInput();

  int n = countRs(input);
  while (n > 1) {
    if (n % 2 == 0) n = n / 2;
    else n = 3*n + 1;
  }

  if (countRs(input) > countDs(input)) {
    accept();
  } else {
    reject();
  }
}
```

M is a secure voting machine

if and only if

$\mathscr{L}(M) = \{ w \in \{\texttt{r}, \texttt{d}\}^* \mid w \text{ has more } \texttt{r}\text{'s than } \texttt{d}\text{'s} \}$

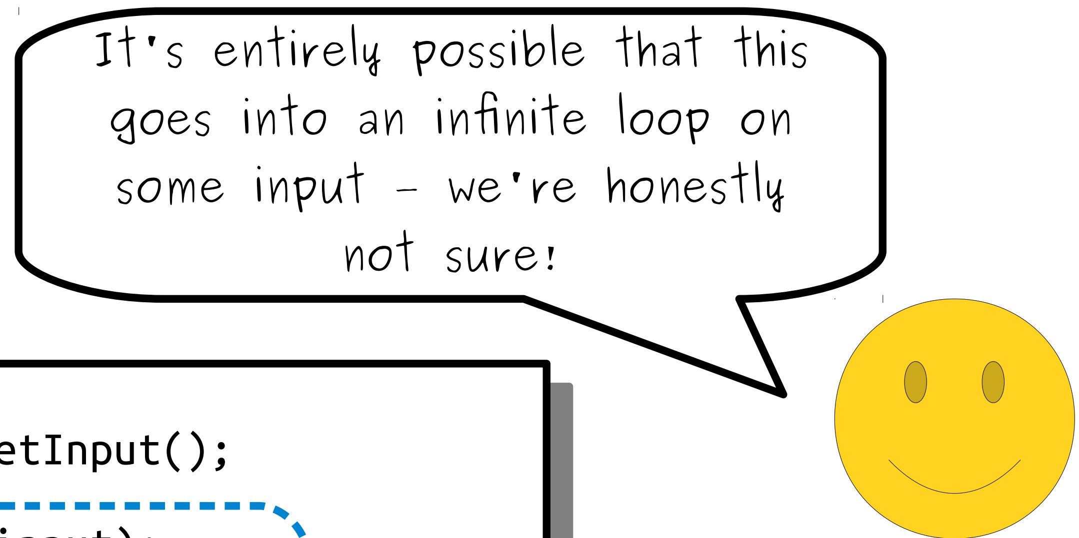



```
int main() {
  string input = getInput();

  int n = countRs(input);
  while (n > 1) {
    if (n % 2 == 0) n = n / 2;
    else n = 3*n + 1;
  }

  if (countRs(input) > countDs(input)) {
    accept();
  } else {
    reject();
  }
}
```

M is a secure voting machine

if and only if

$\mathscr{L}(M) = \{\ w \in \{r, d\}^* \mid w$ has more r's than d's $\}$



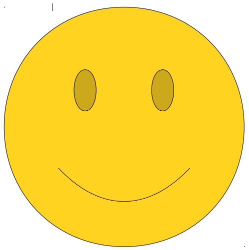

```
int main() {
  string input = getInput();

  int n = countRs(input);
  while (n > 1) {
    if (n % 2 == 0) n = n / 2;
    else n = 3*n + 1;
  }

  if (countRs(input) > countDs(input)) {
    accept();
  } else {
    reject();
  }
}
```

M is a secure voting machine

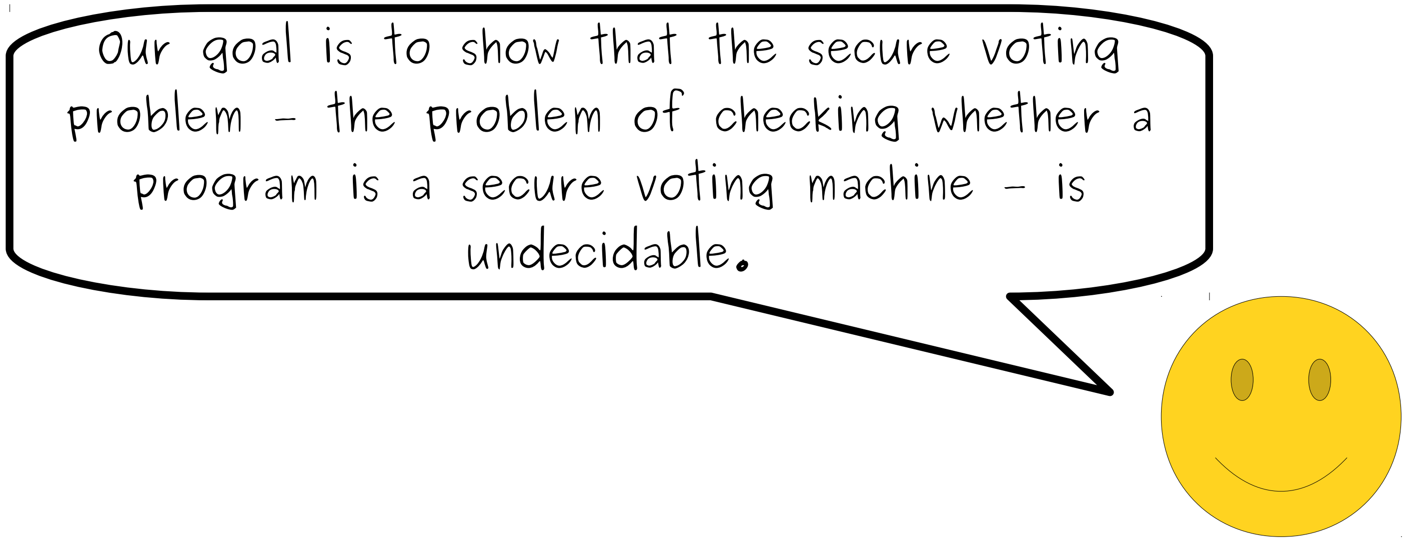
if and only if

$\mathscr{L}(M) = \{\ w \in \{$`r`, `d`$\}^* \mid w$ has more `r`'s than `d`'s $\}$

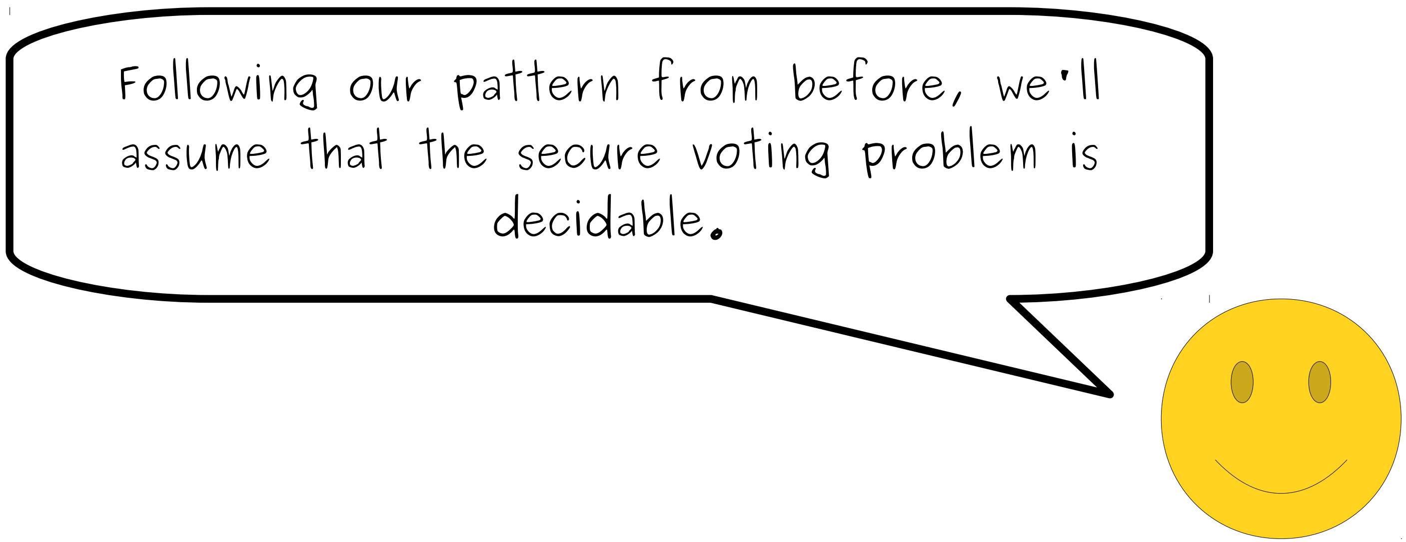
So, to recap:
Building a secure voting machine isn't hard.
Checking whether an arbitrary program is a secure voting machine is really hard.



```
int main() {
  string input = getInput();

  int n = countRs(input);
  while (n > 1) {
    if (n % 2 == 0) n = n / 2;
    else n = 3*n + 1;
  }

  if (countRs(input) > countDs(input)) {
    accept();
  } else {
    reject();
  }
}
```


Our goal is to show that the secure voting problem – the problem of checking whether a program is a secure voting machine – is undecidable.

The secure voting problem is decidable.

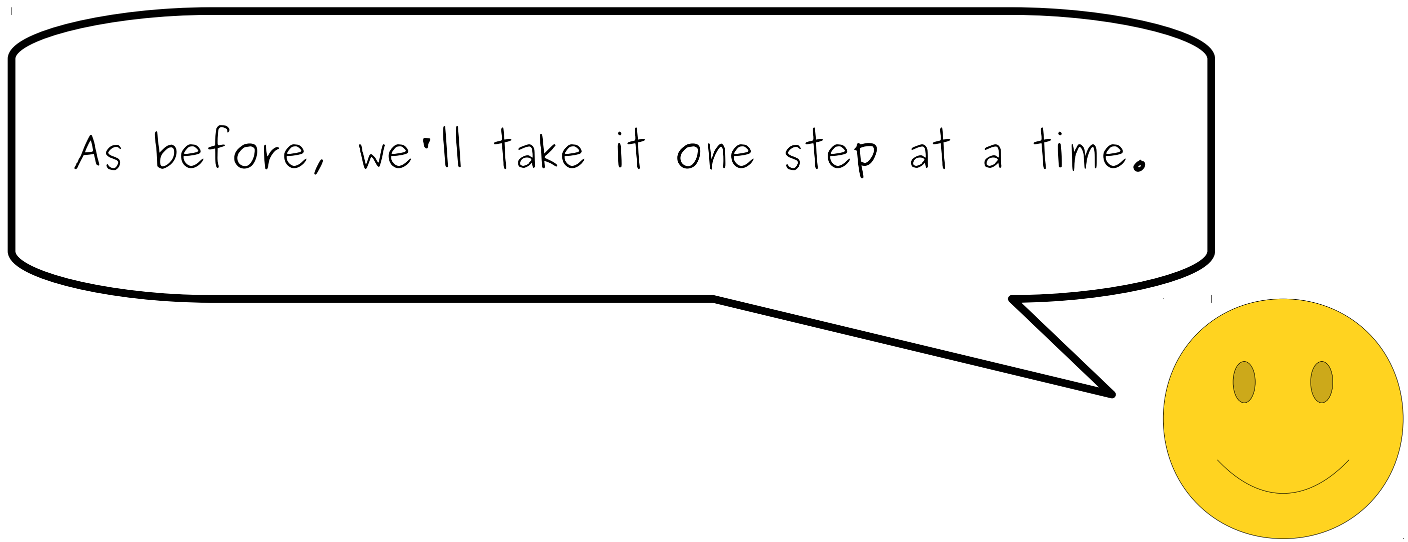


The secure voting problem is decidable.

Contradiction!

The secure voting problem is decidable.

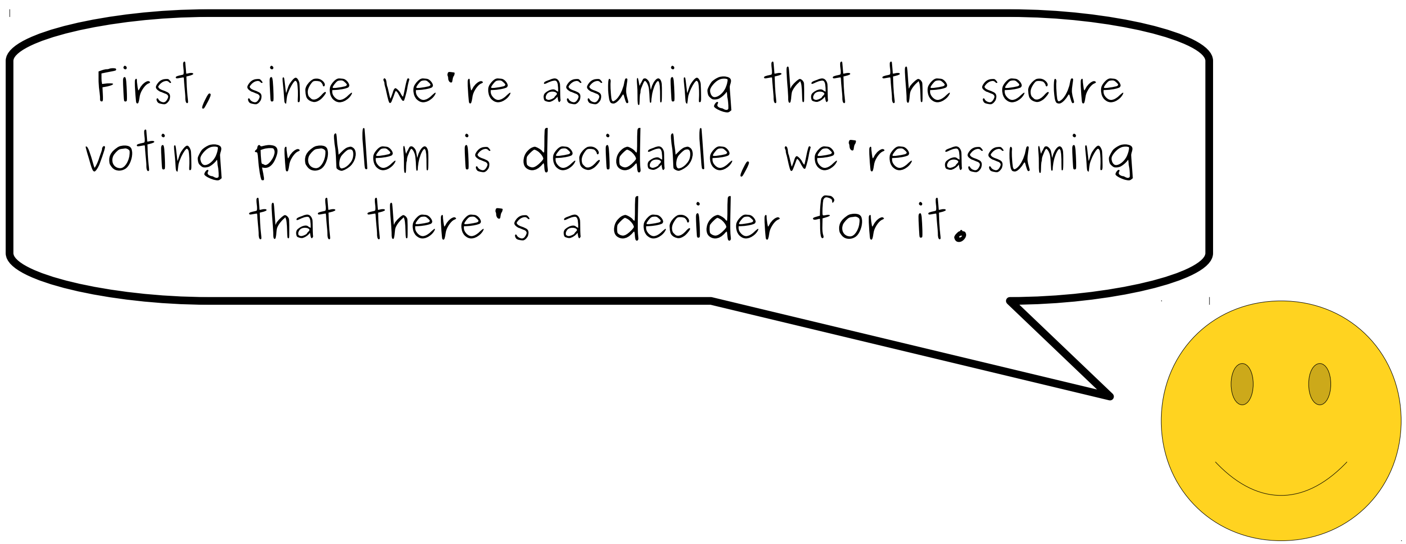
Contradiction!



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem



Contradiction!

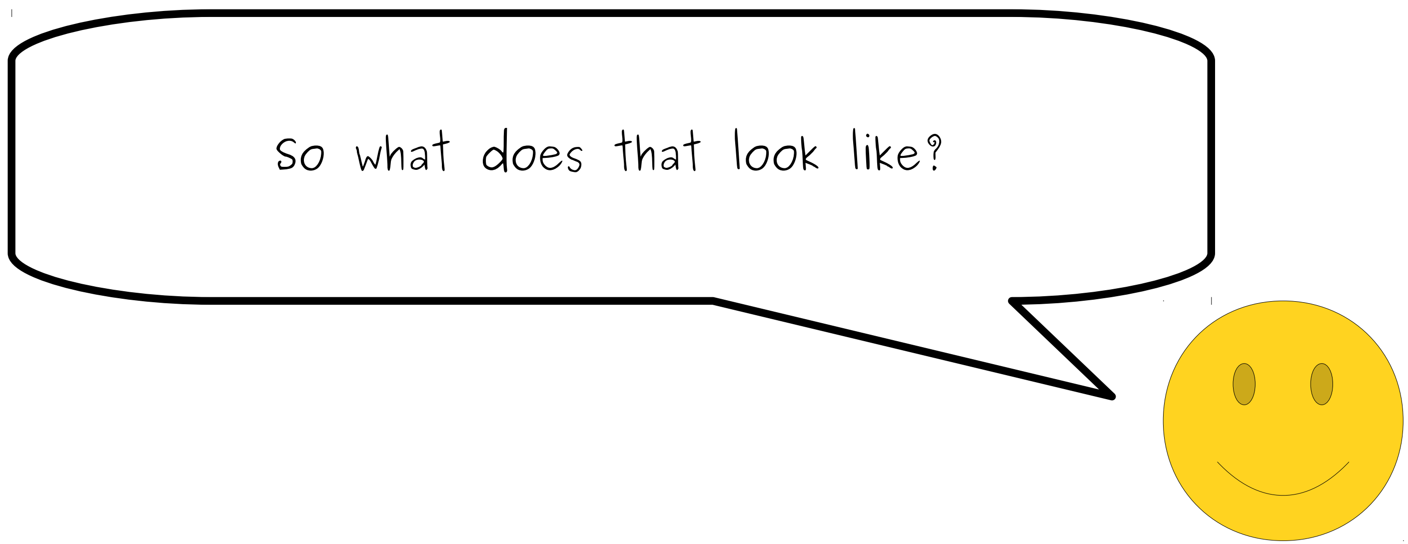
The secure voting problem is decidable.

↓

There is a decider D for the secure voting problem

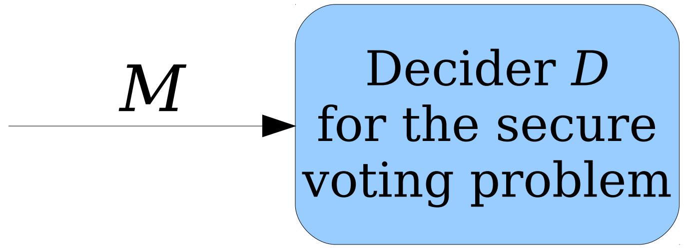
Contradiction!

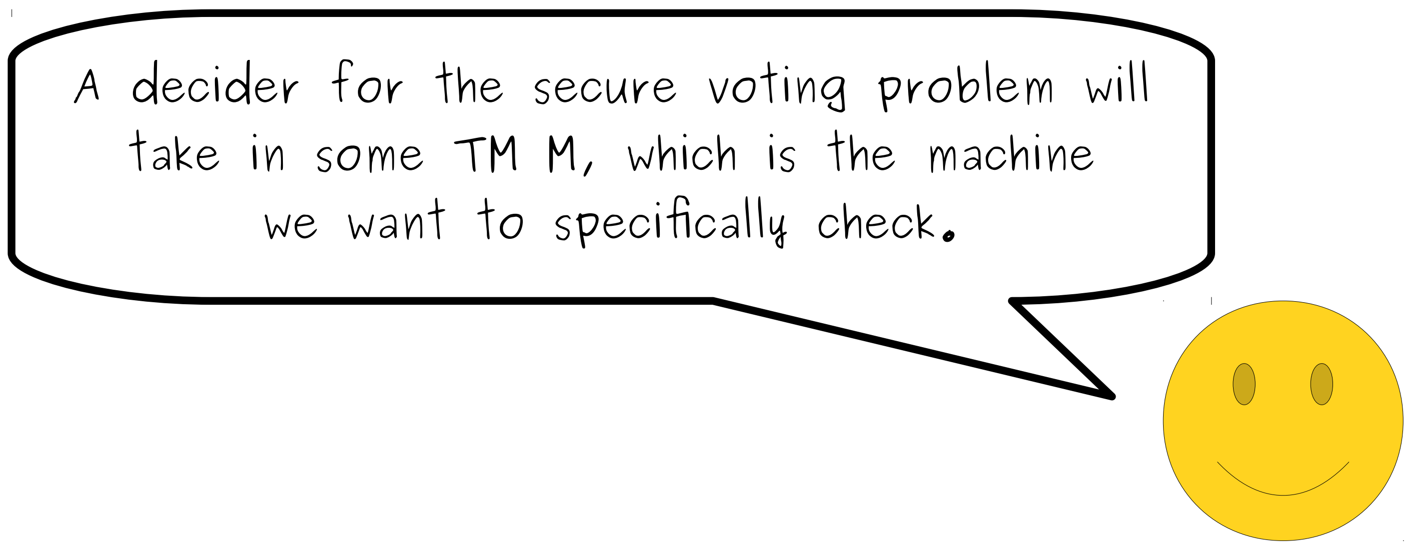
Decider D for the secure voting problem



The secure voting problem is decidable.

There is a decider D for the secure voting problem



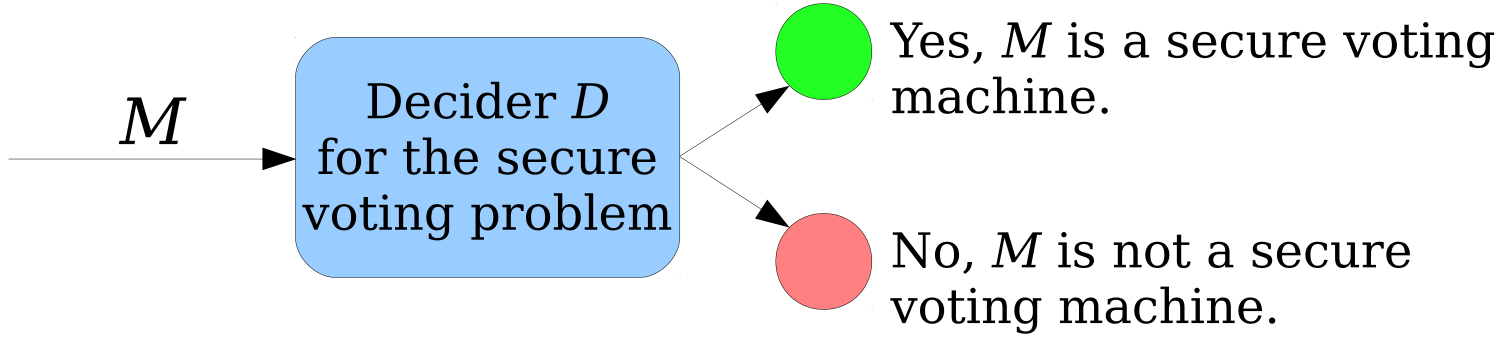


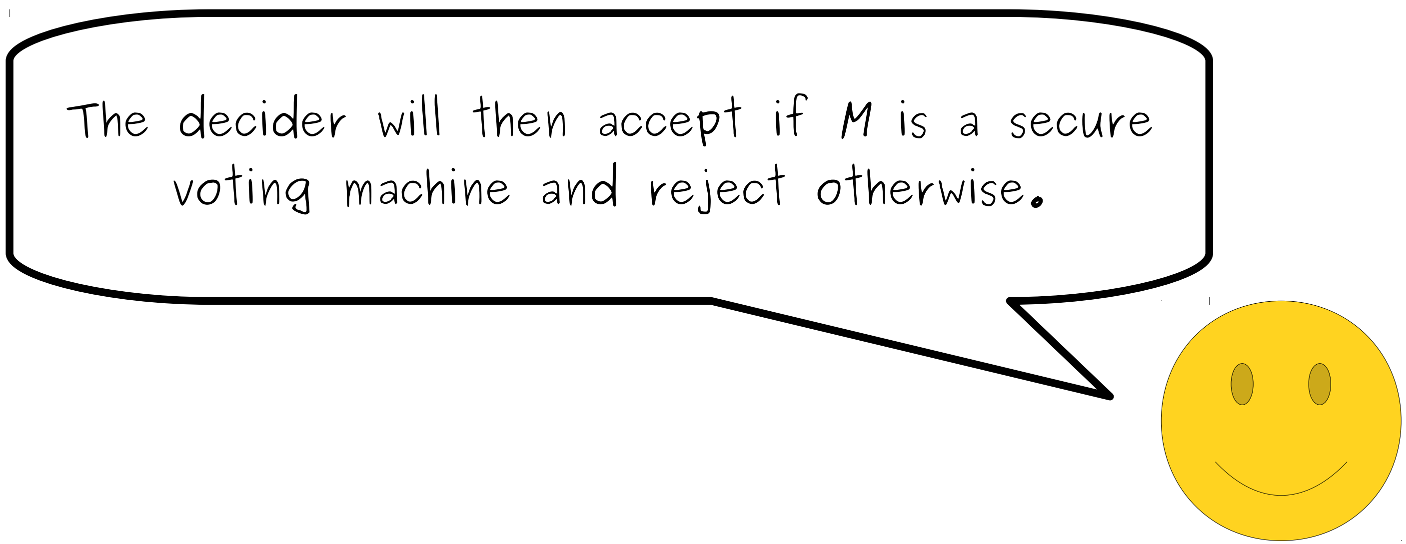
Contradiction!

The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem





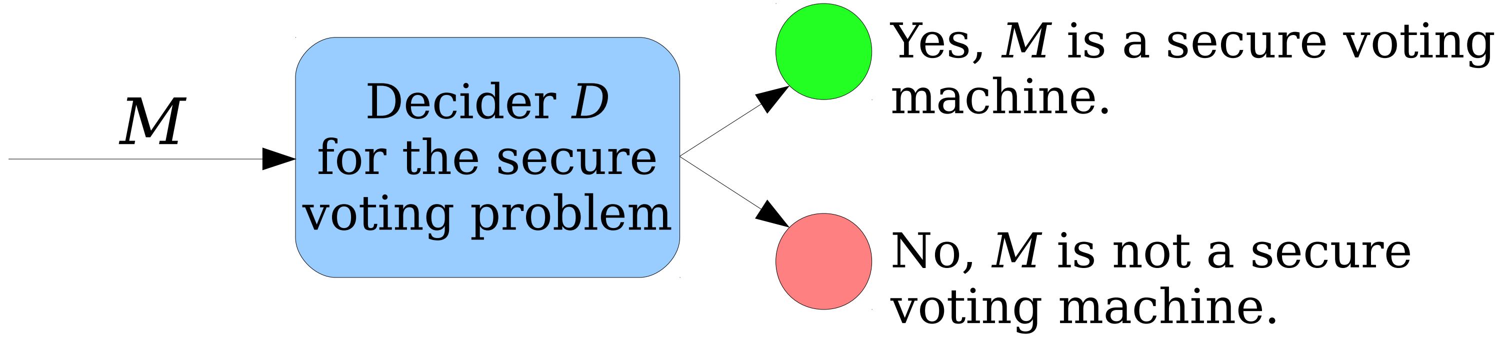
Contradiction!

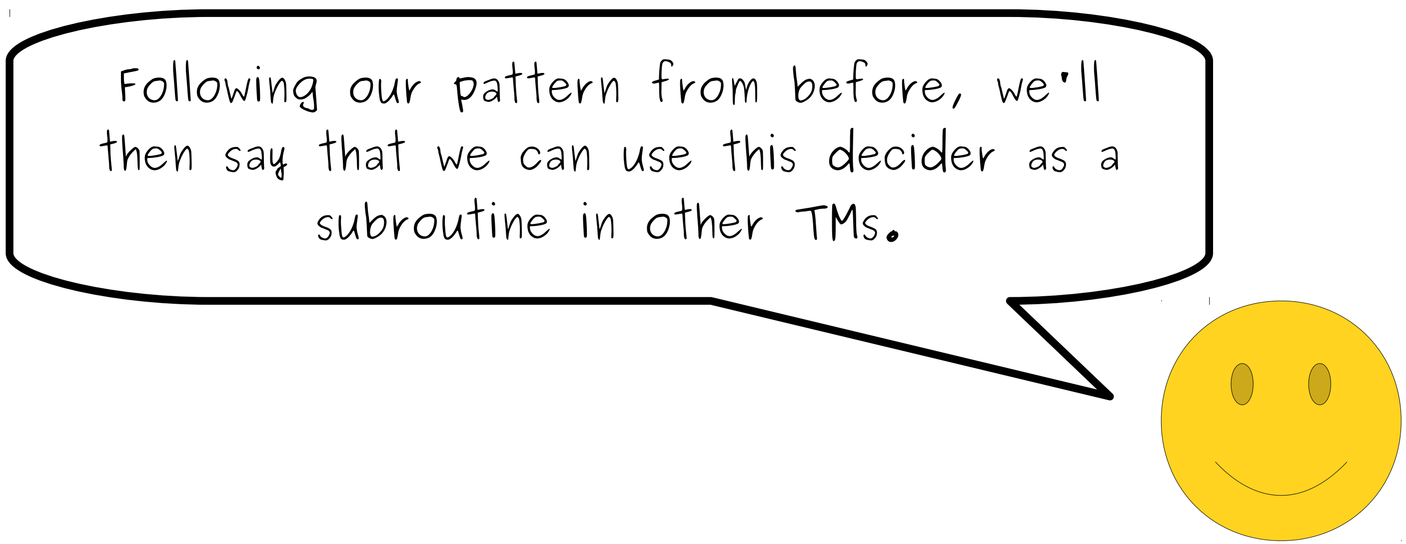
The secure voting problem is decidable.

There is a decider D for the secure voting problem

We can write programs that use D as a helper method

Contradiction!



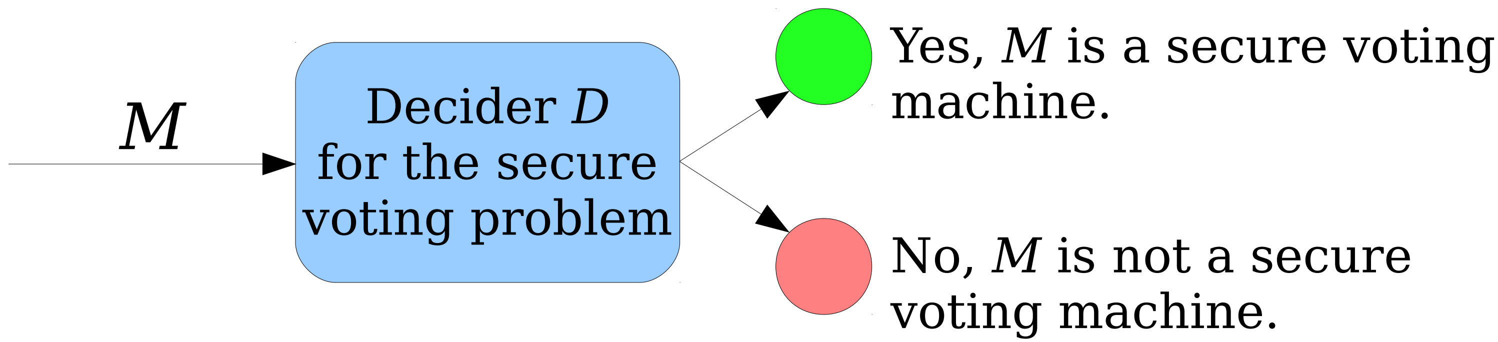


The secure voting problem is decidable.

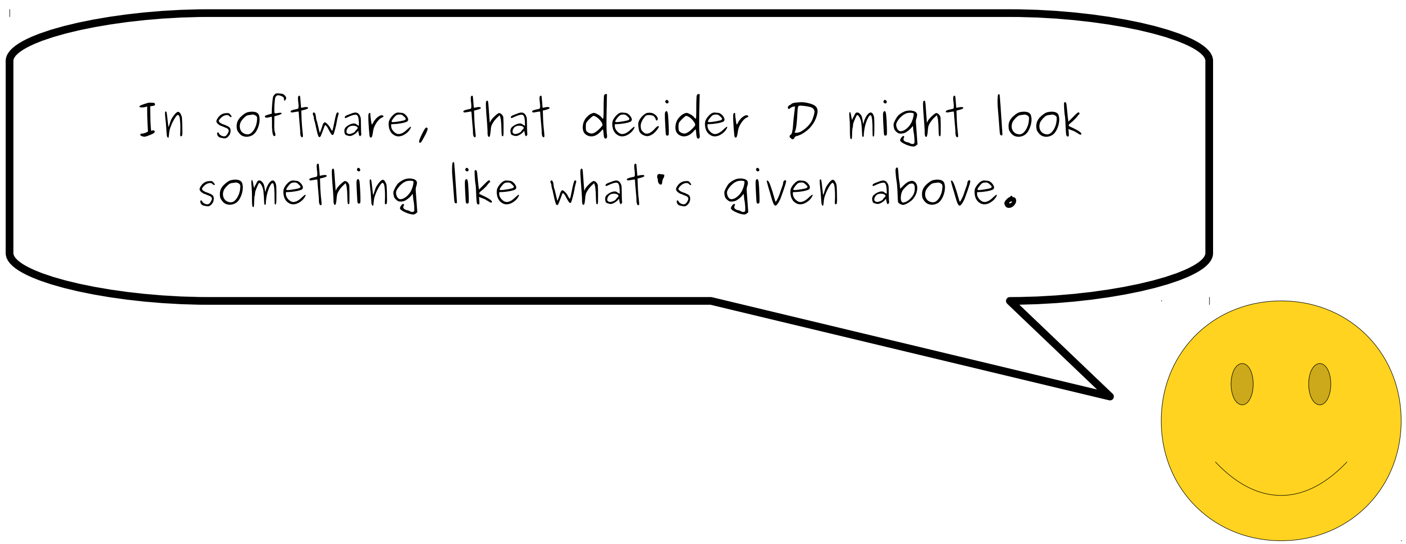
There is a decider D for the secure voting problem

We can write programs that use D as a helper method

Contradiction!




```
bool isSecure(string program)
```

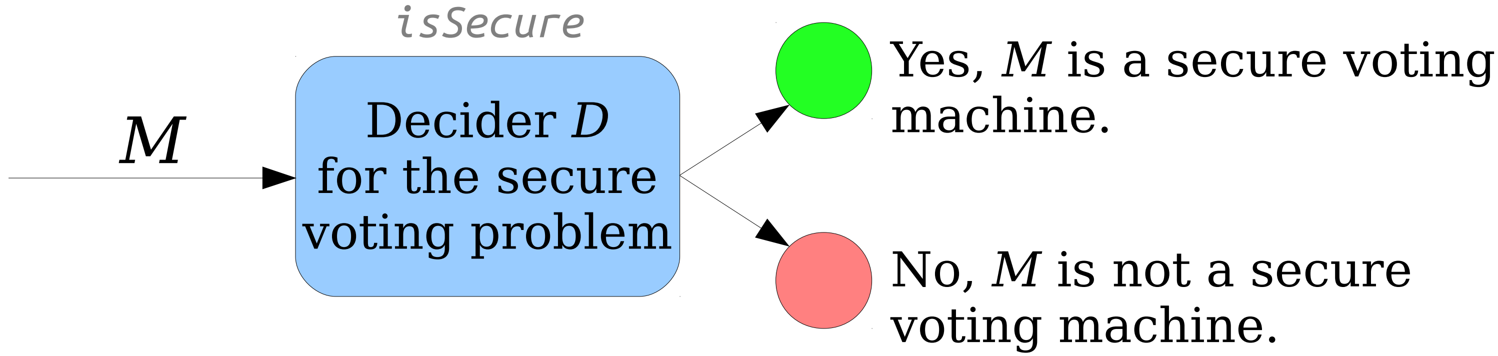


The secure voting problem is decidable.

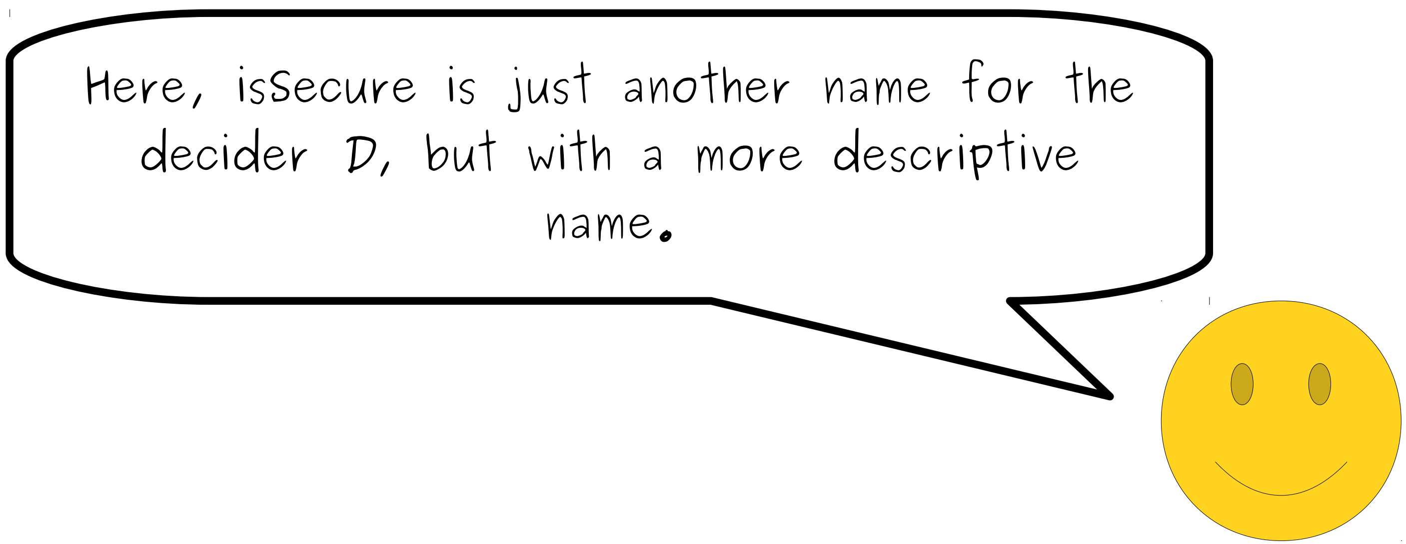
There is a decider D for the secure voting problem

We can write programs that use D as a helper method

Contradiction!




```
bool isSecure(string program)
```

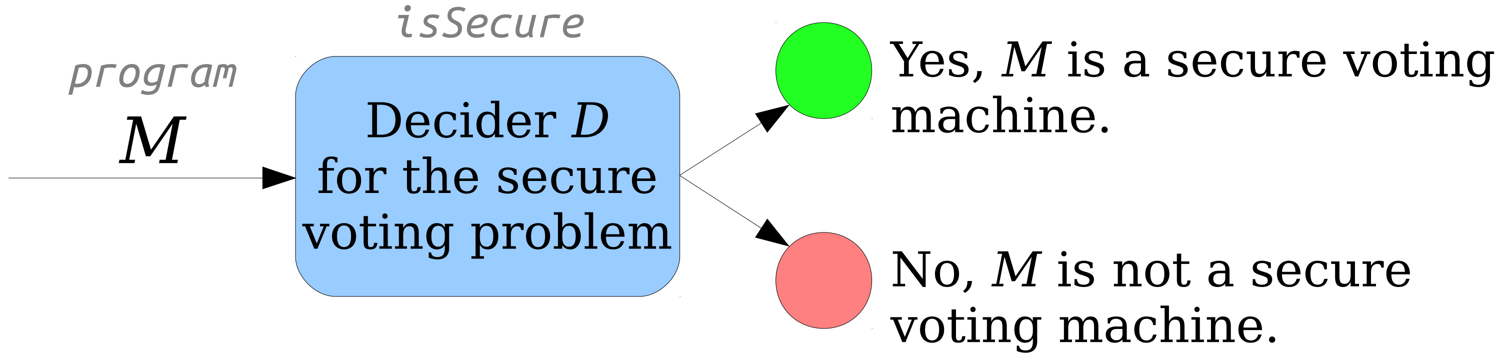


The secure voting problem is decidable.

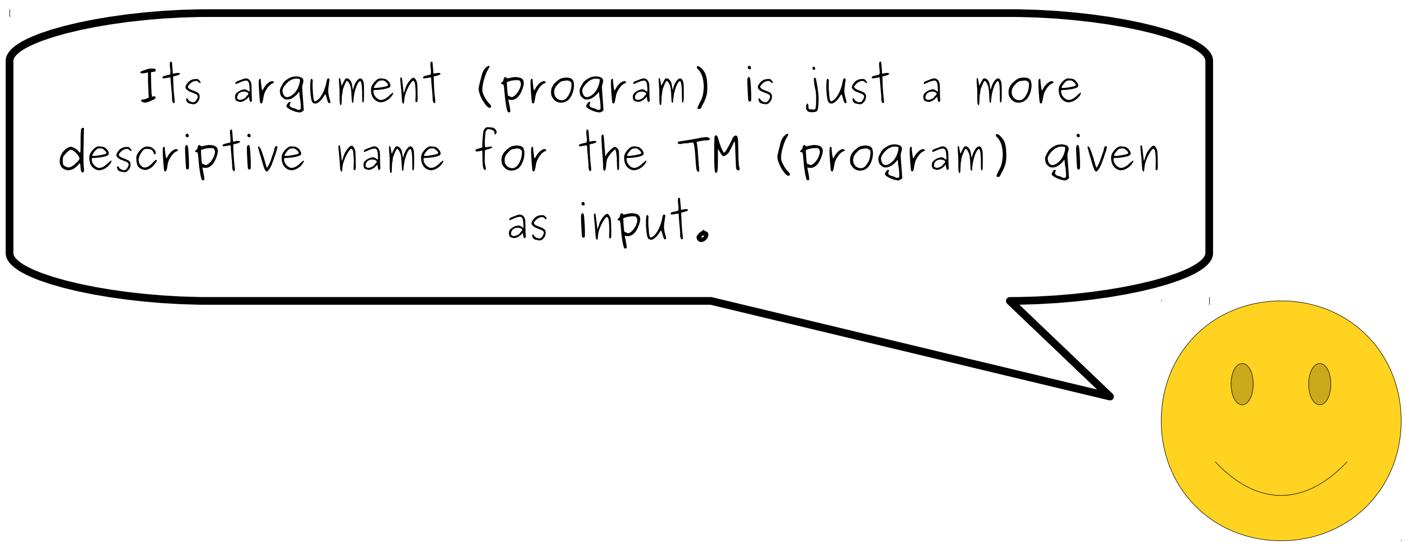
There is a decider *D* for the secure voting problem

We can write programs that use *D* as a helper method

Contradiction!




```
bool isSecure(string program)
```



The secure voting problem is decidable.

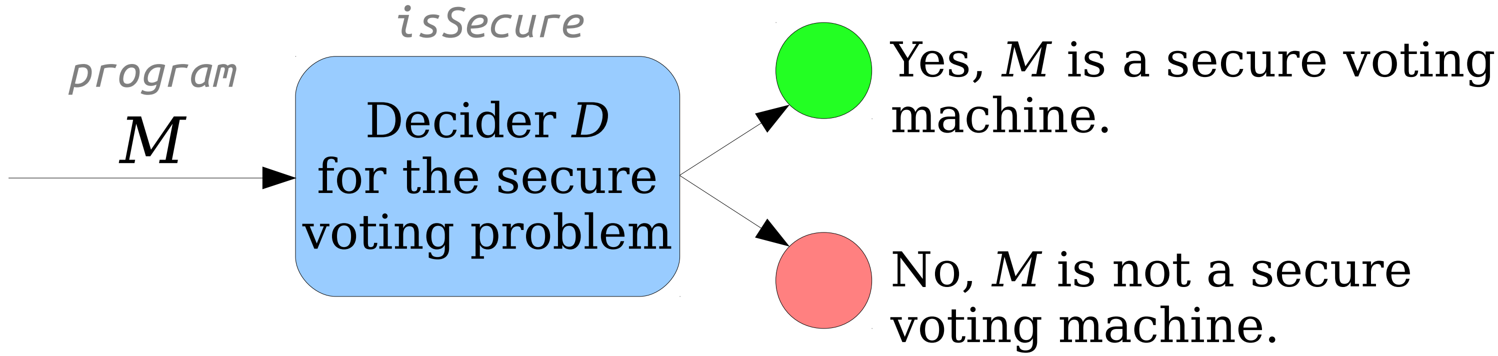
↓

There is a decider *D* for the secure voting problem

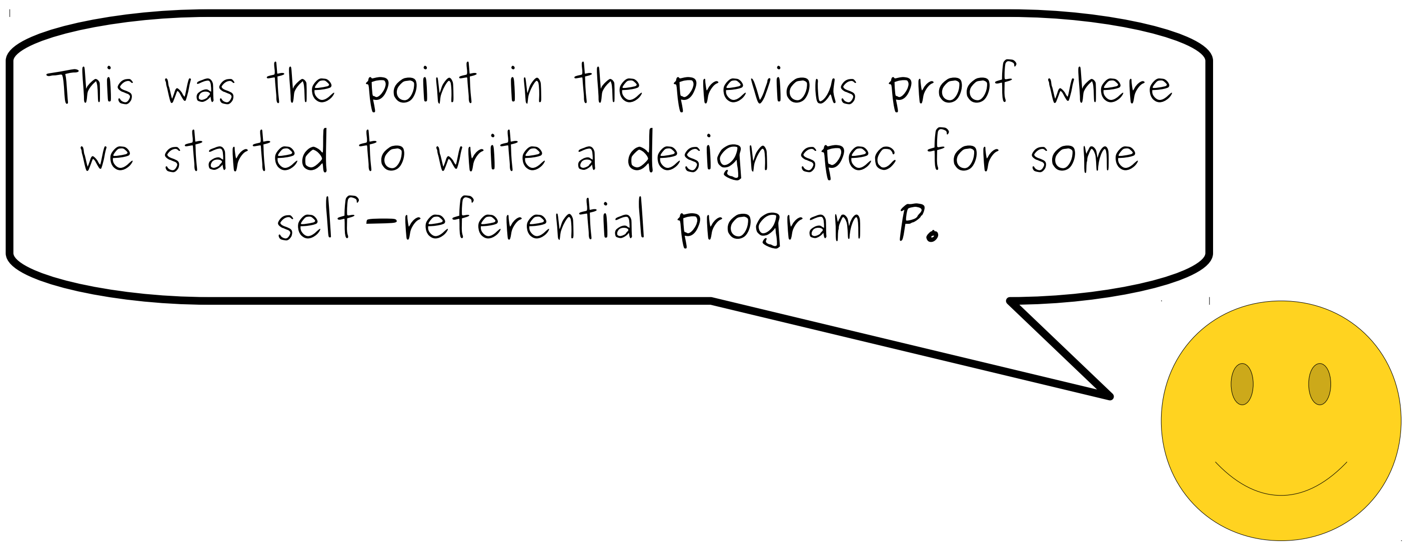
↓

We can write programs that use *D* as a helper method

Contradiction!




```
bool isSecure(string program)
```



The secure voting problem is decidable.

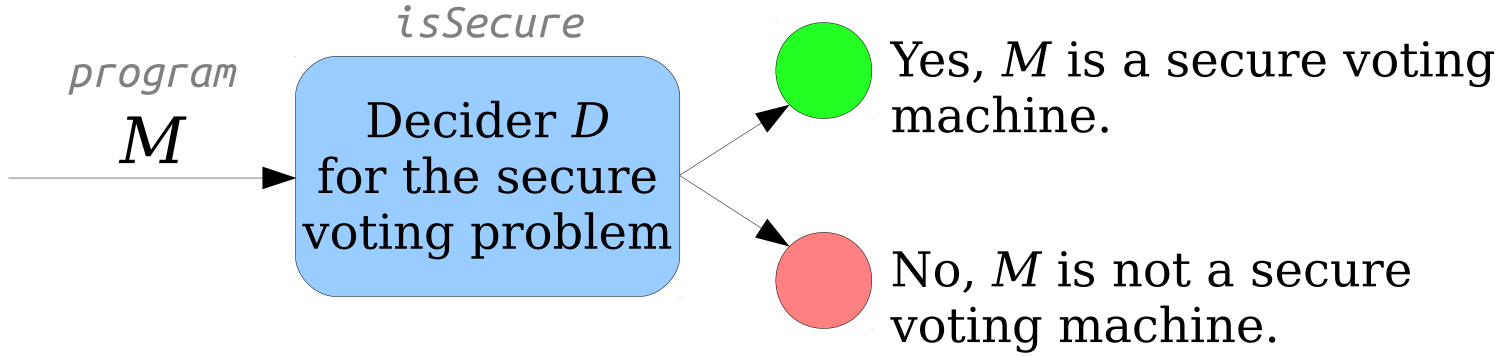
↓

There is a decider *D* for the secure voting problem

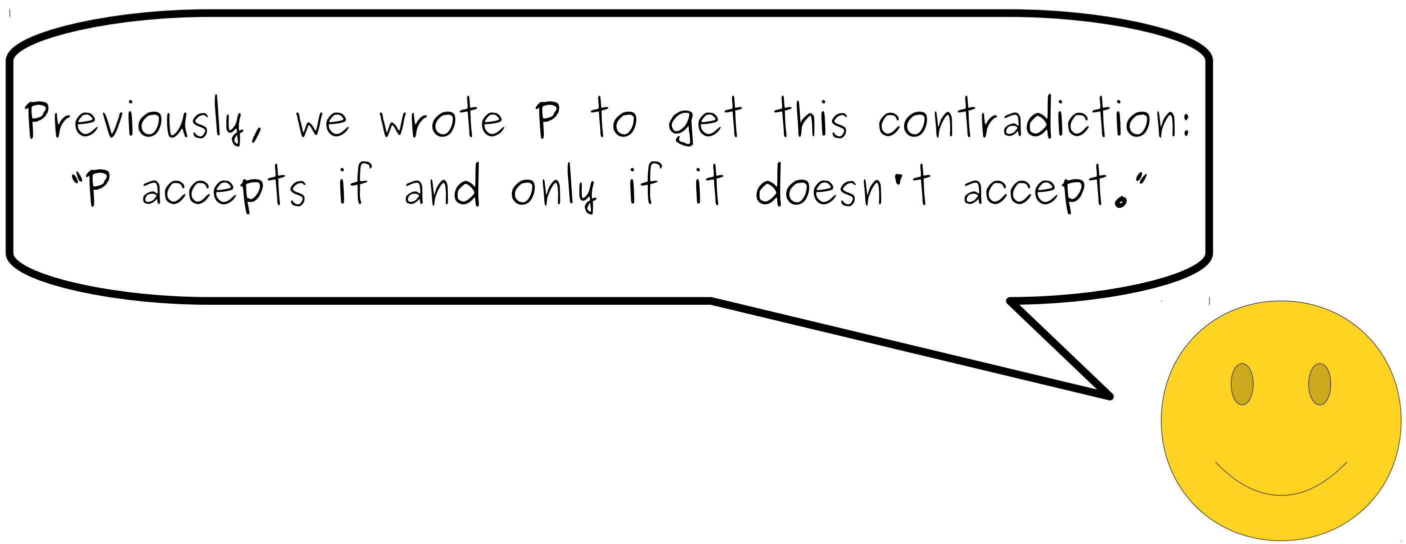
↓

We can write programs that use *D* as a helper method

Contradiction!




```
bool isSecure(string program)
```



The secure voting problem is decidable.

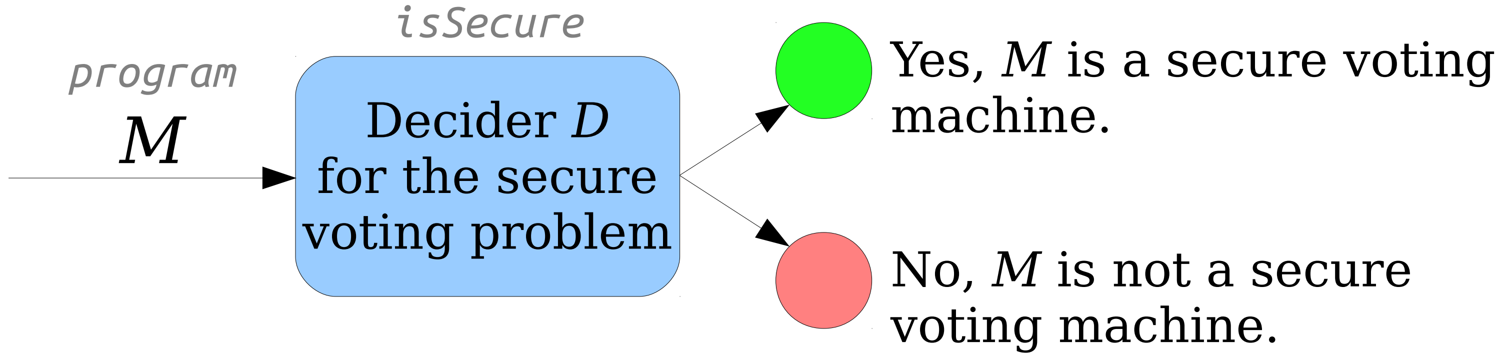
↓

There is a decider *D* for the secure voting problem

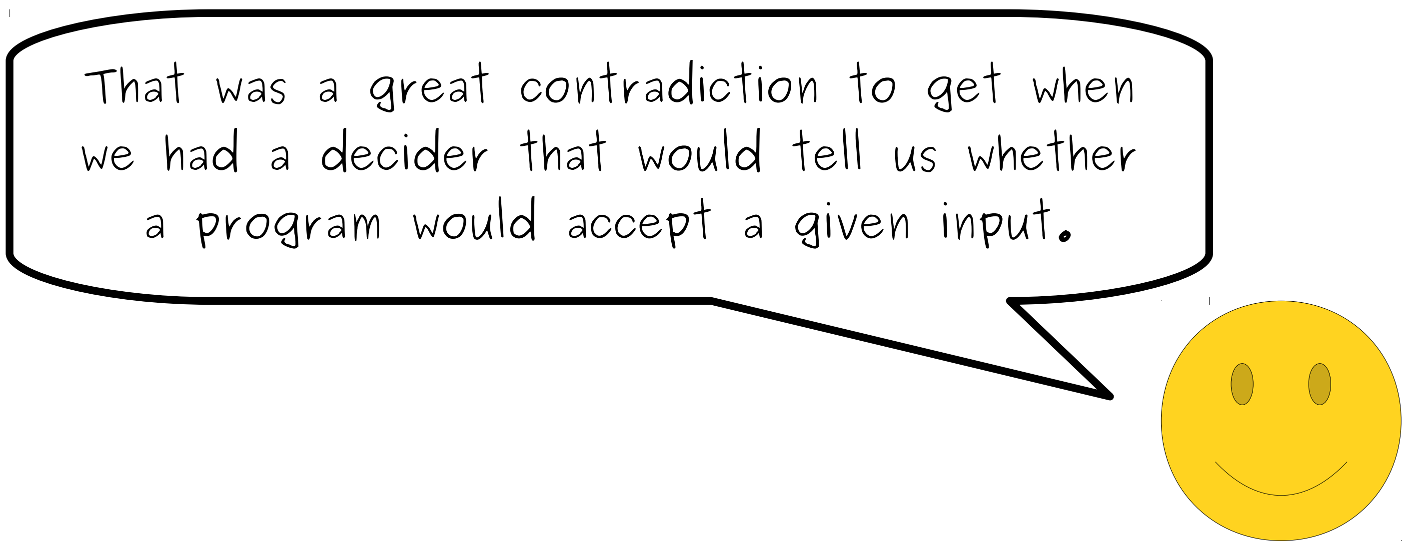
↓

We can write programs that use *D* as a helper method

Contradiction!




```
bool isSecure(string program)
```

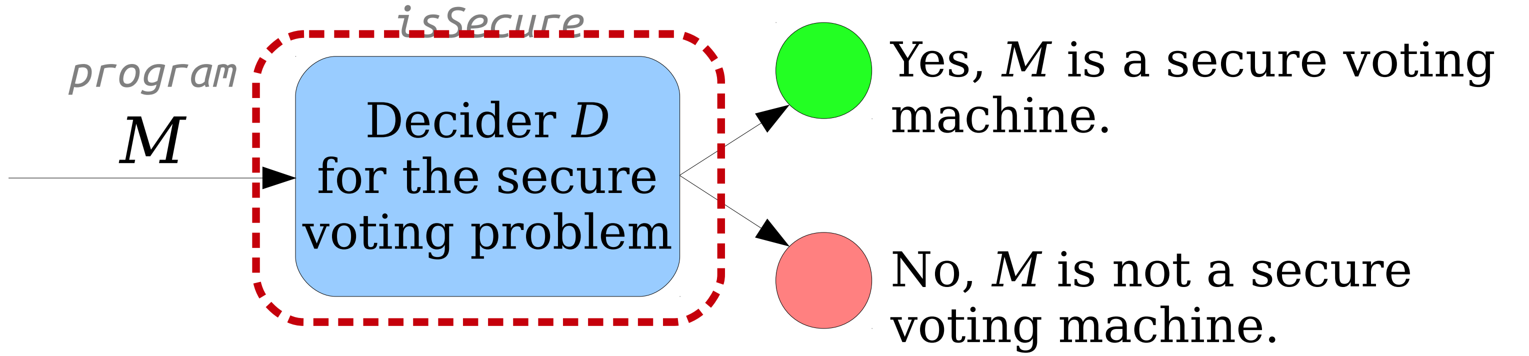


The secure voting problem is decidable.

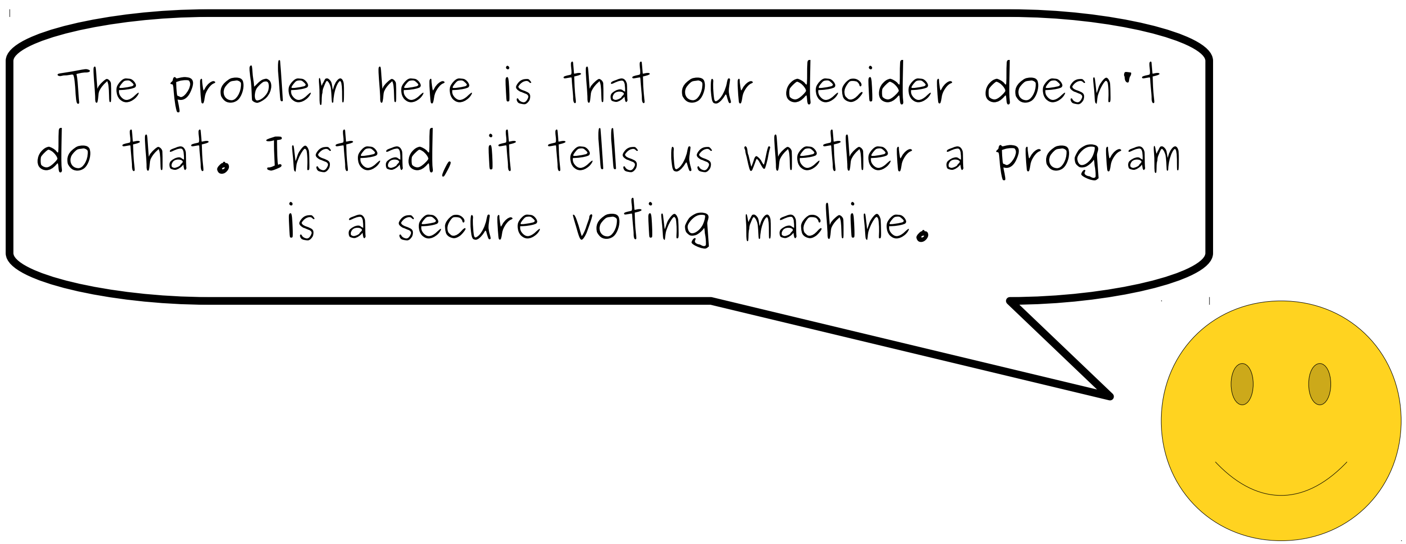
There is a decider D for the secure voting problem

We can write programs that use D as a helper method

Contradiction!




```
bool isSecure(string program)
```

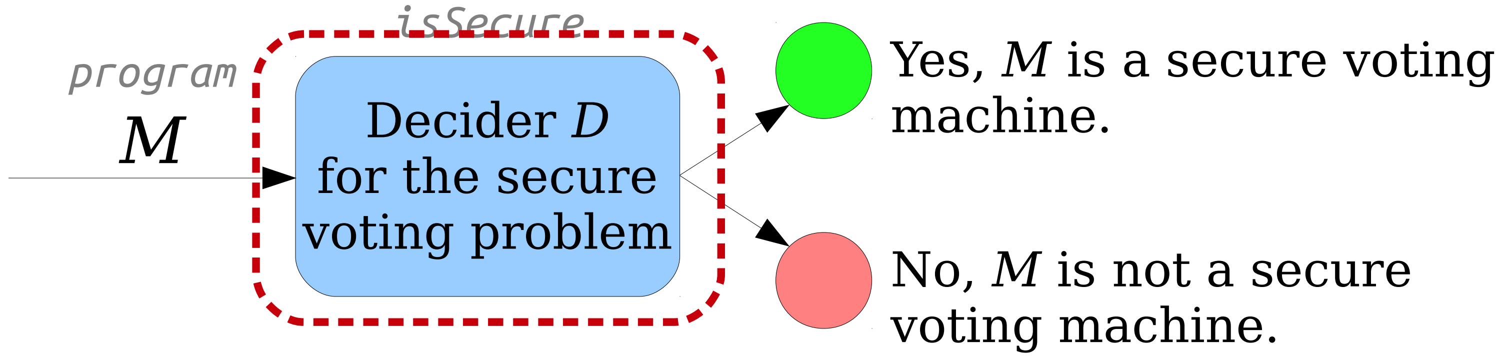


The secure voting problem is decidable.

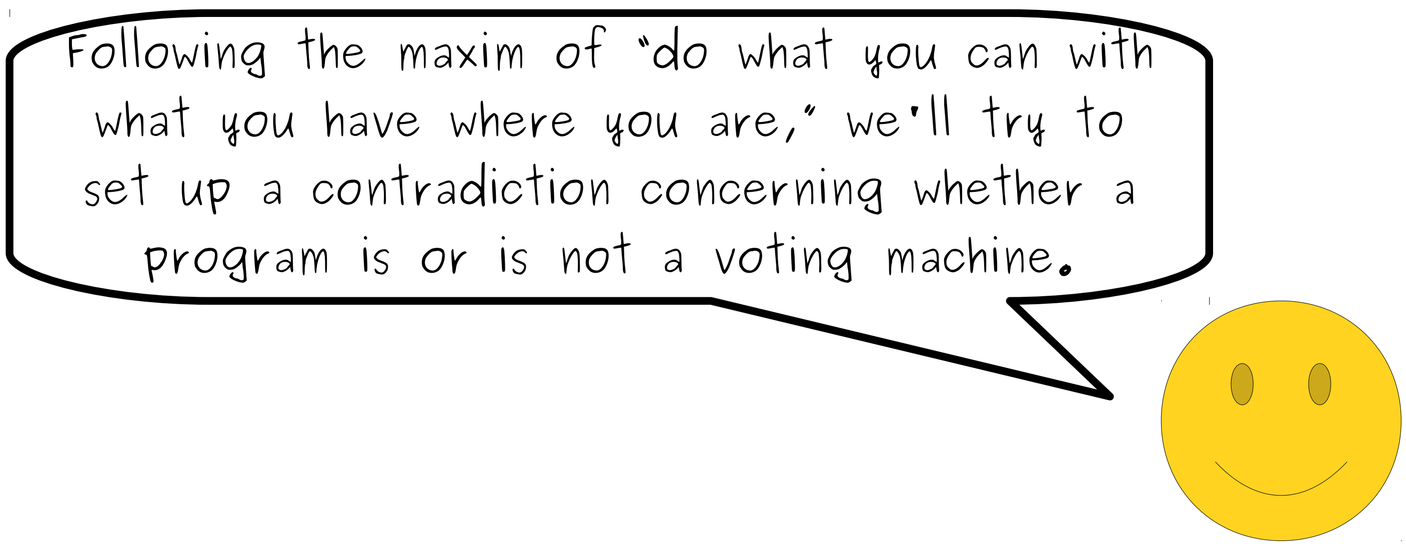
There is a decider *D* for the secure voting problem

We can write programs that use *D* as a helper method

Contradiction!




```
bool isSecure(string program)
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

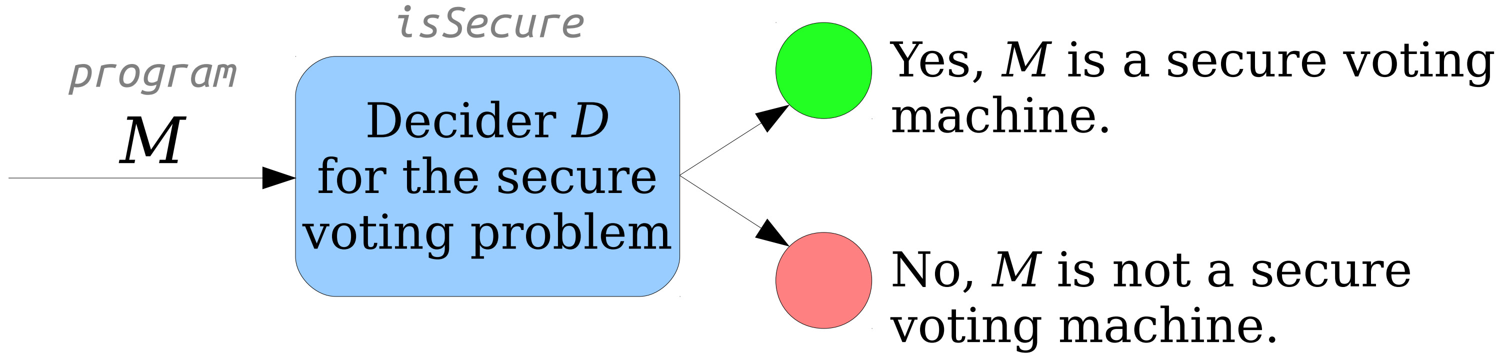
↓

We can write programs that use *D* as a helper method

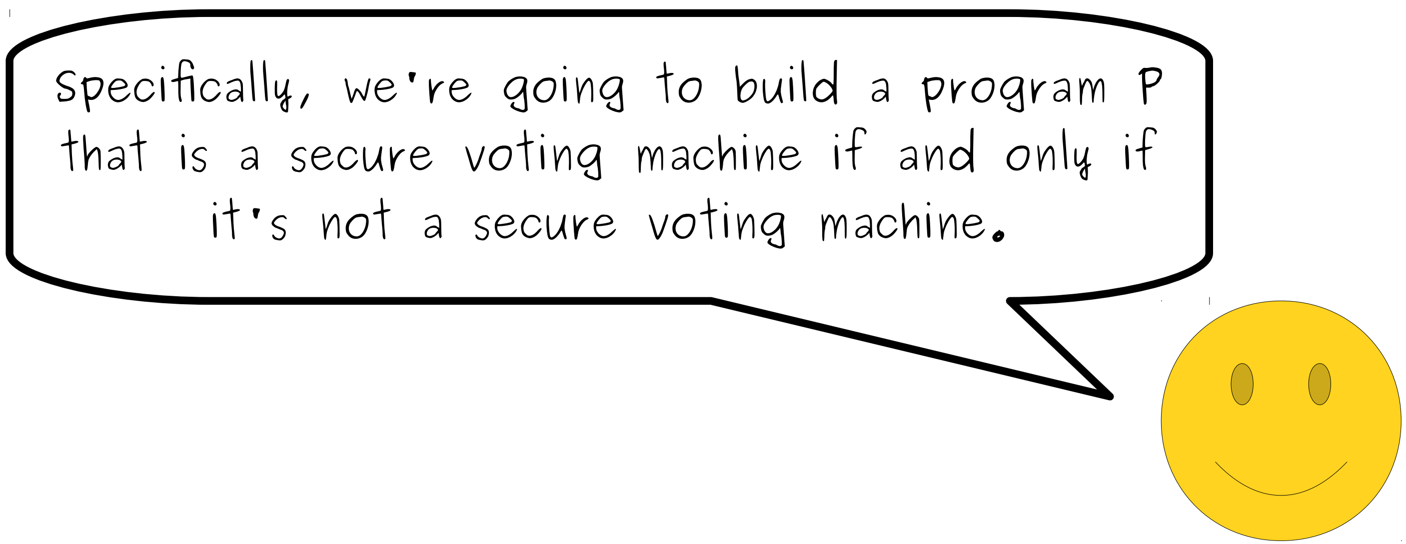
↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!




```
bool isSecure(string program)
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

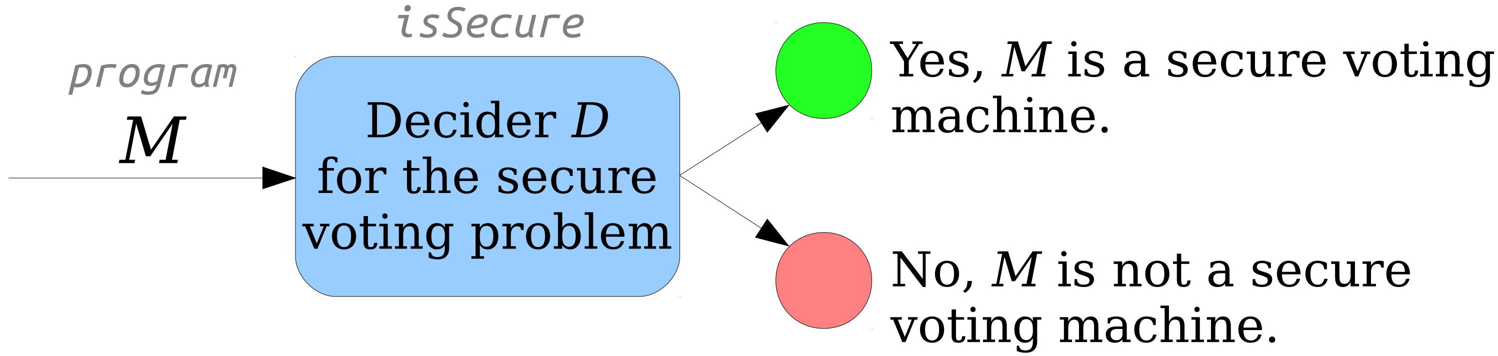
↓

We can write programs that use *D* as a helper method

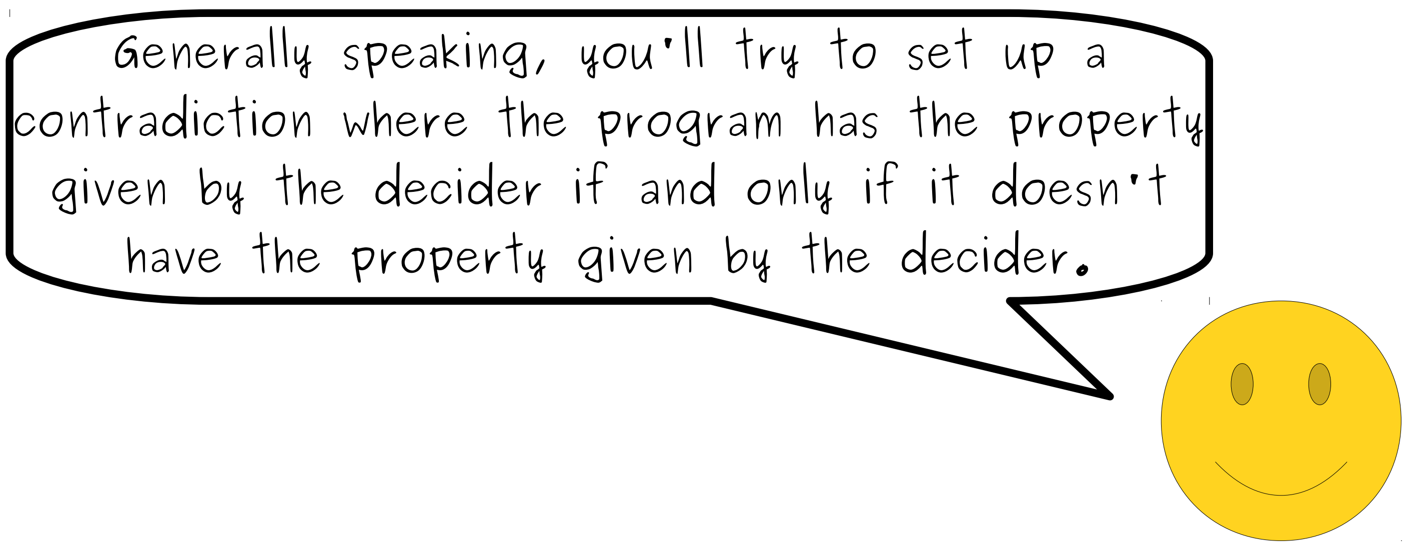
↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!




```
bool isSecure(string program)
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

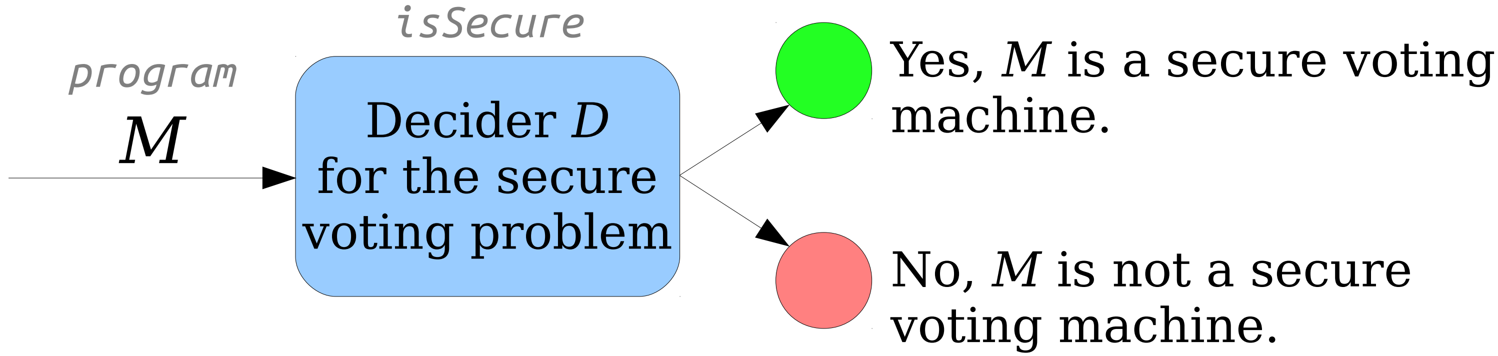
↓

We can write programs that use *D* as a helper method

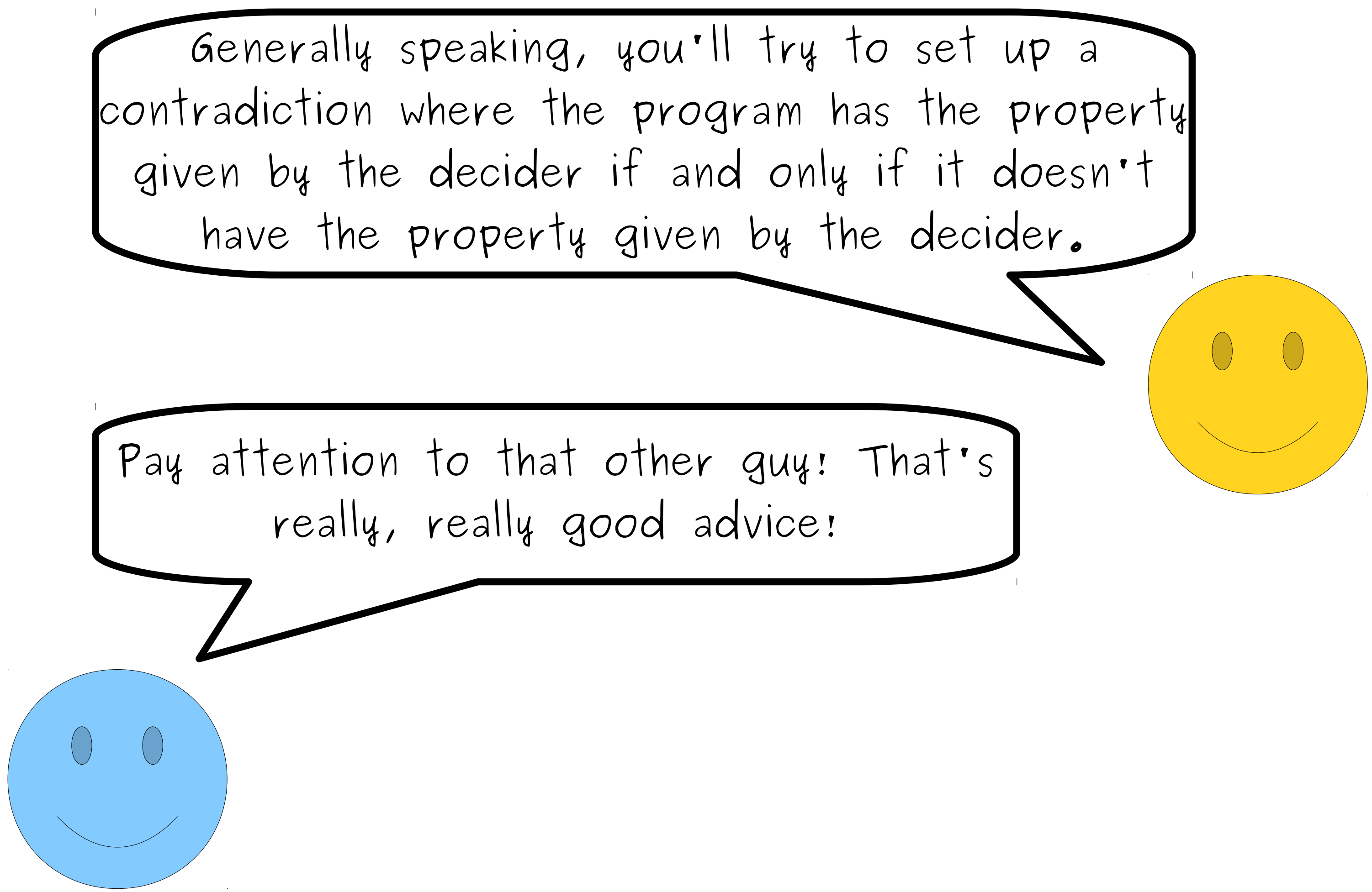
↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!




```
bool isSecure(string program)
```



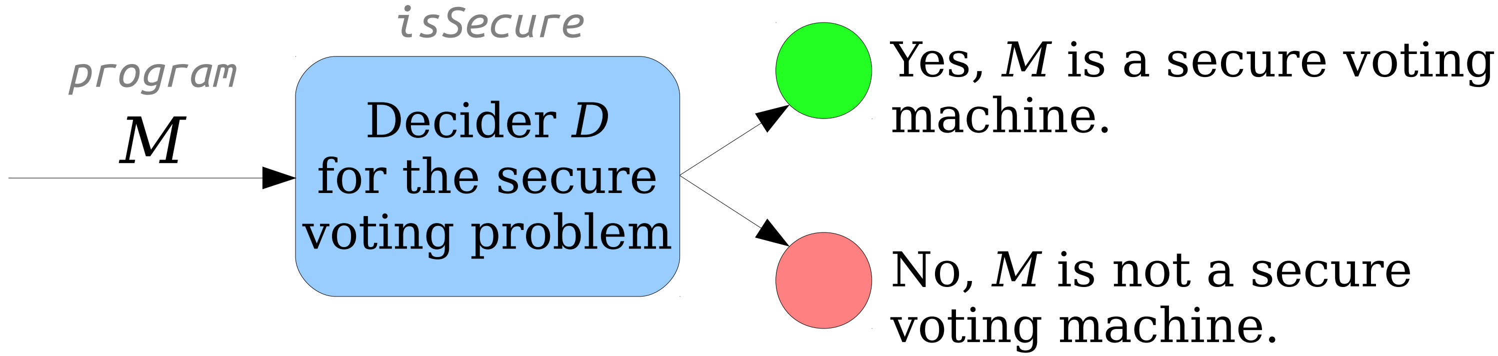
The secure voting problem is decidable.

There is a decider *D* for the secure voting problem

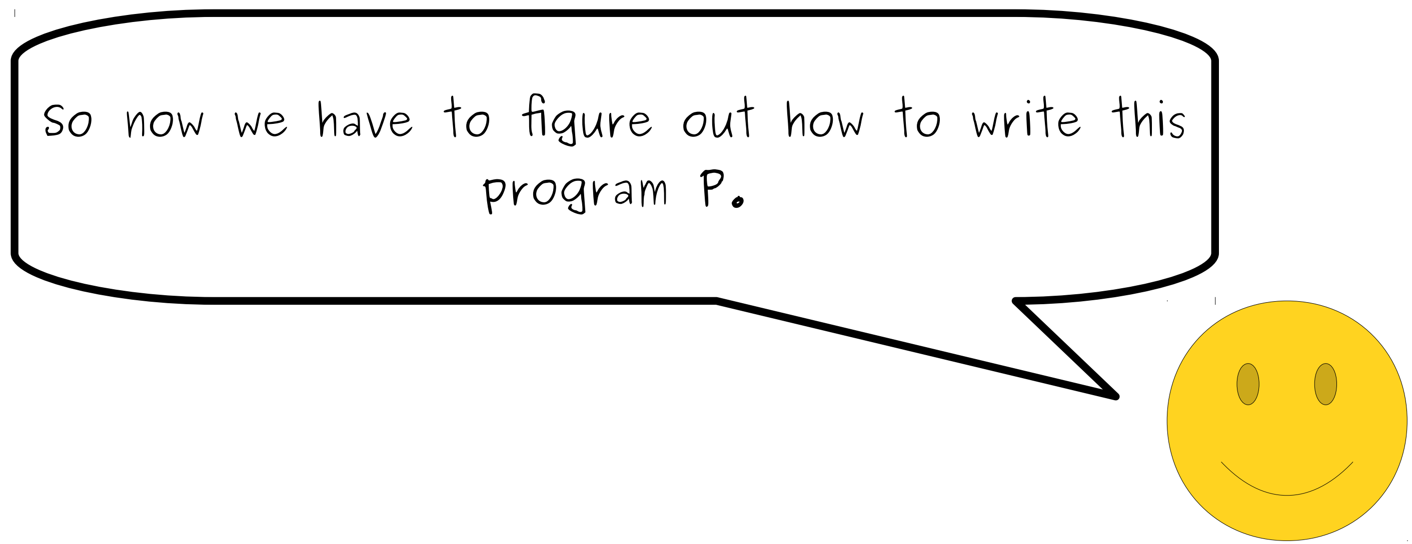
We can write programs that use *D* as a helper method

Program *P* is secure if and only if program *P* is not secure.

Contradiction!




```
bool isSecure(string program)
```



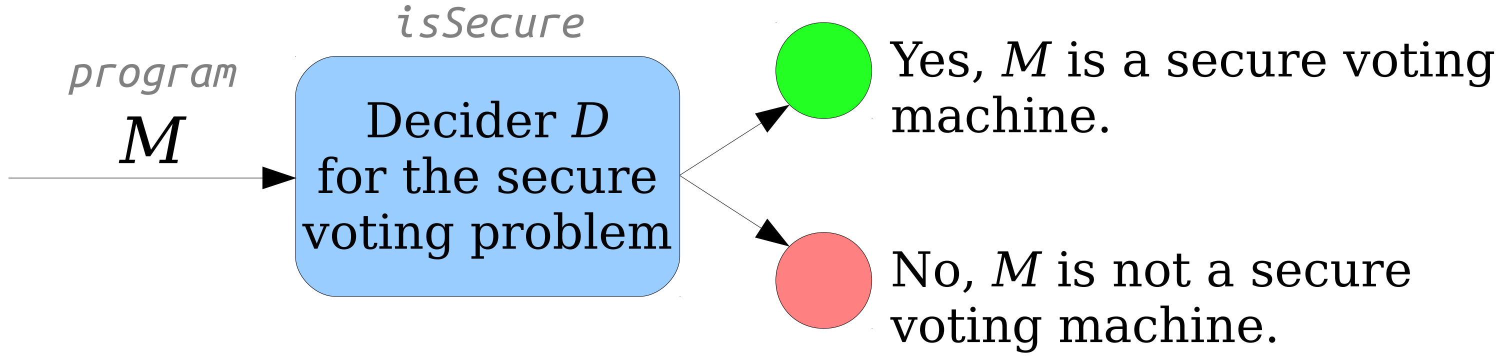
The secure voting problem is decidable.

There is a decider D for the secure voting problem

We can write programs that use D as a helper method

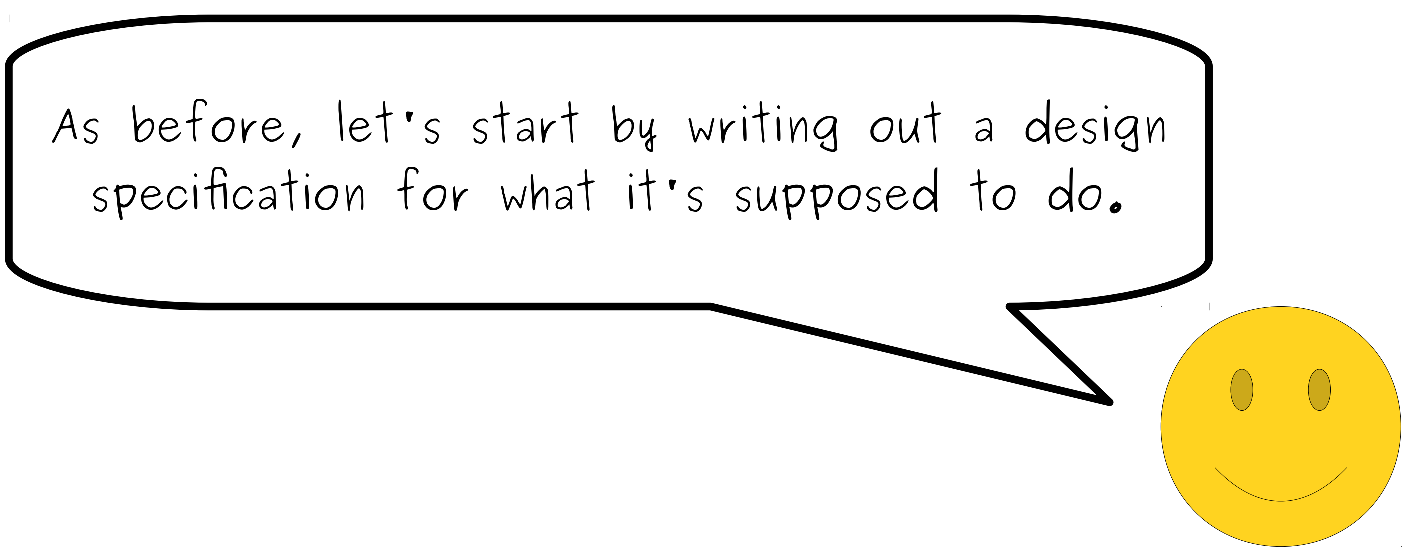
Program P is secure if and only if program P is not secure.

Contradiction!




```
bool isSecure(string program)
```

Program P design specification:



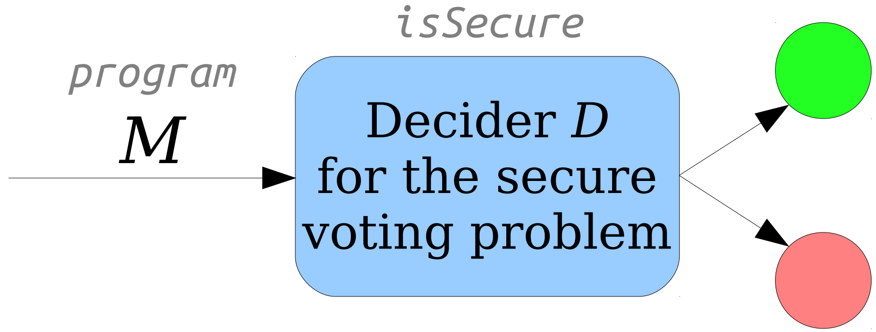
The secure voting problem is decidable.

There is a decider *D* for the secure voting problem

We can write programs that use *D* as a helper method

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



Yes, *M* is a secure voting machine.

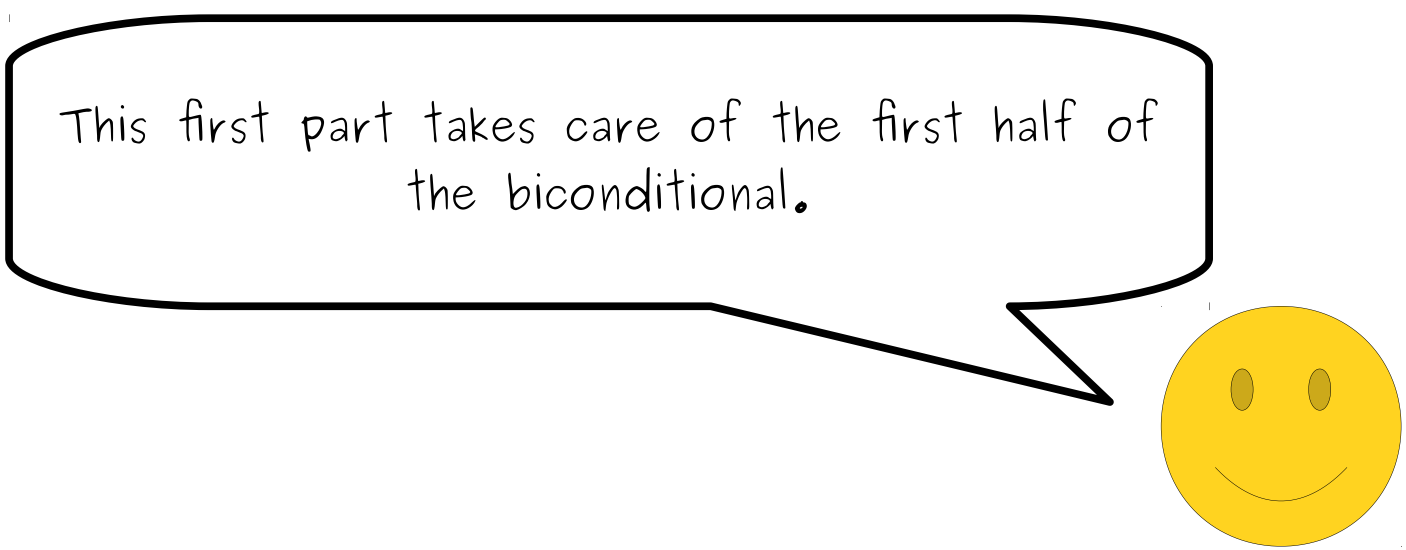
No, *M* is not a secure voting machine.

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then

P is not a secure voting machine.



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

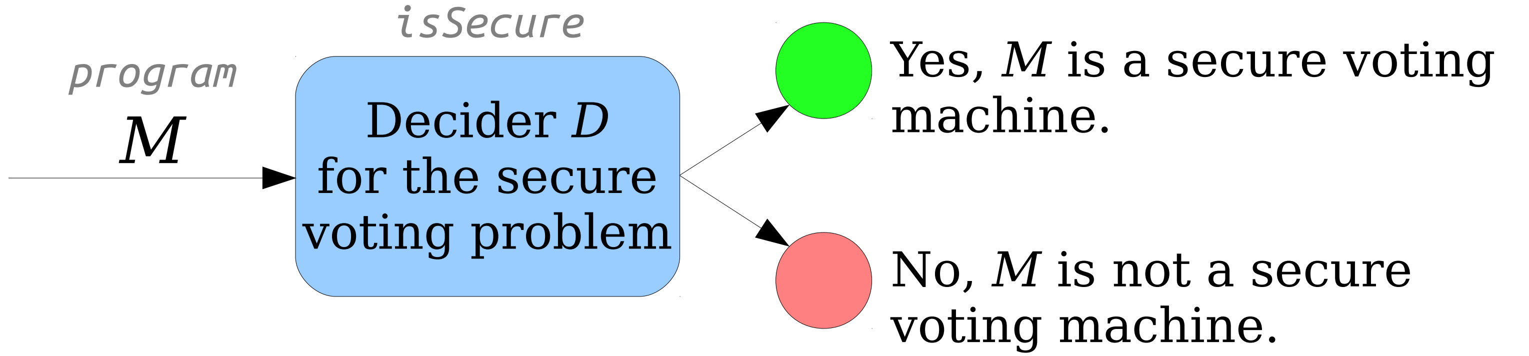
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

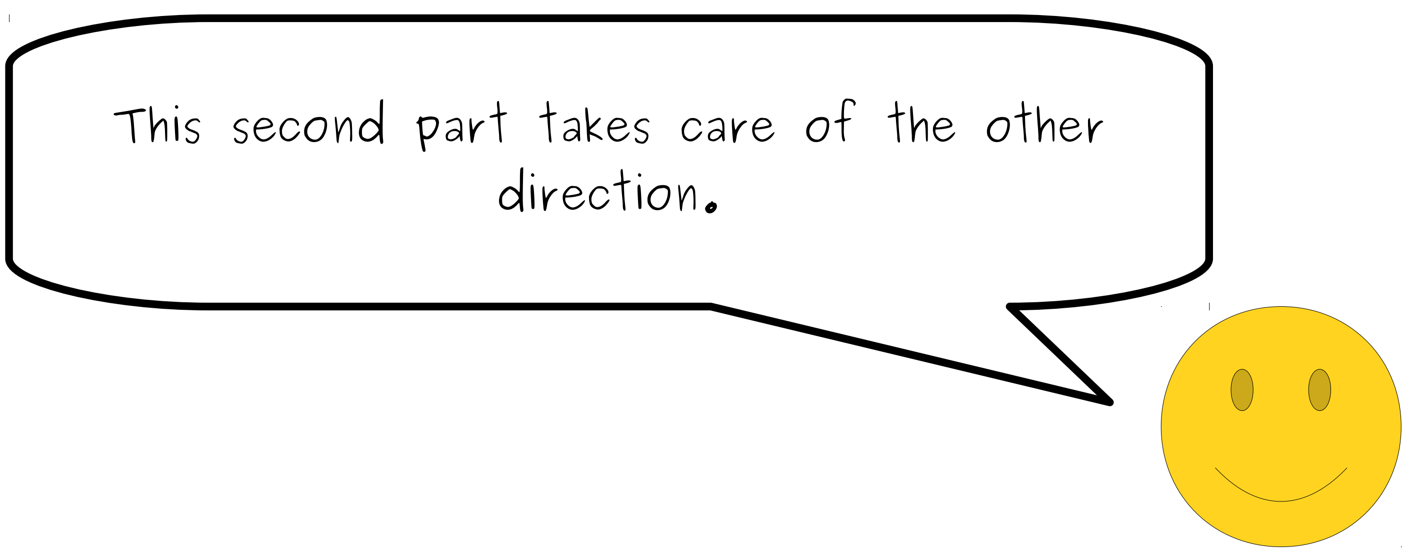
Contradiction!




```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.



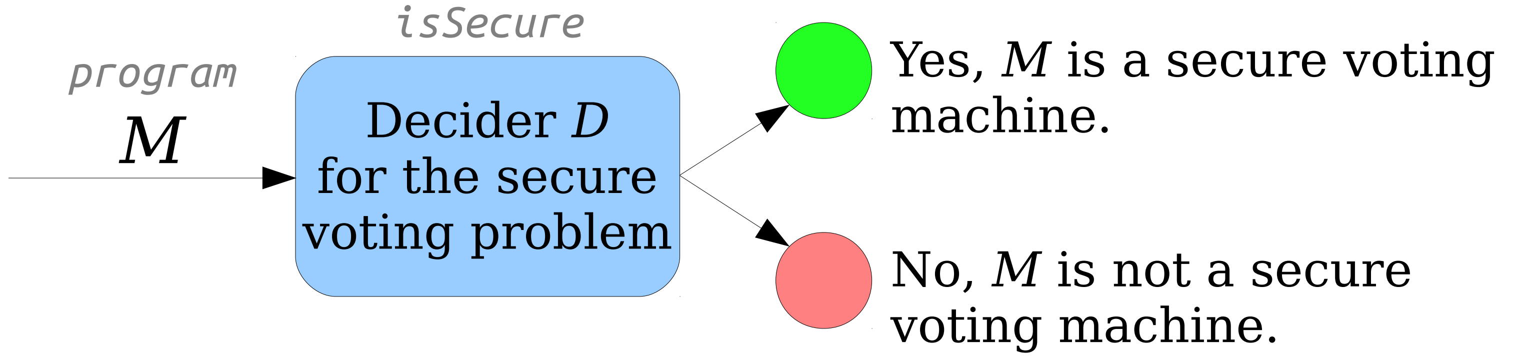
The secure voting problem is decidable.

There is a decider *D* for the secure voting problem

We can write programs that use *D* as a helper method

Program *P* is secure if and only if program *P* is not secure.

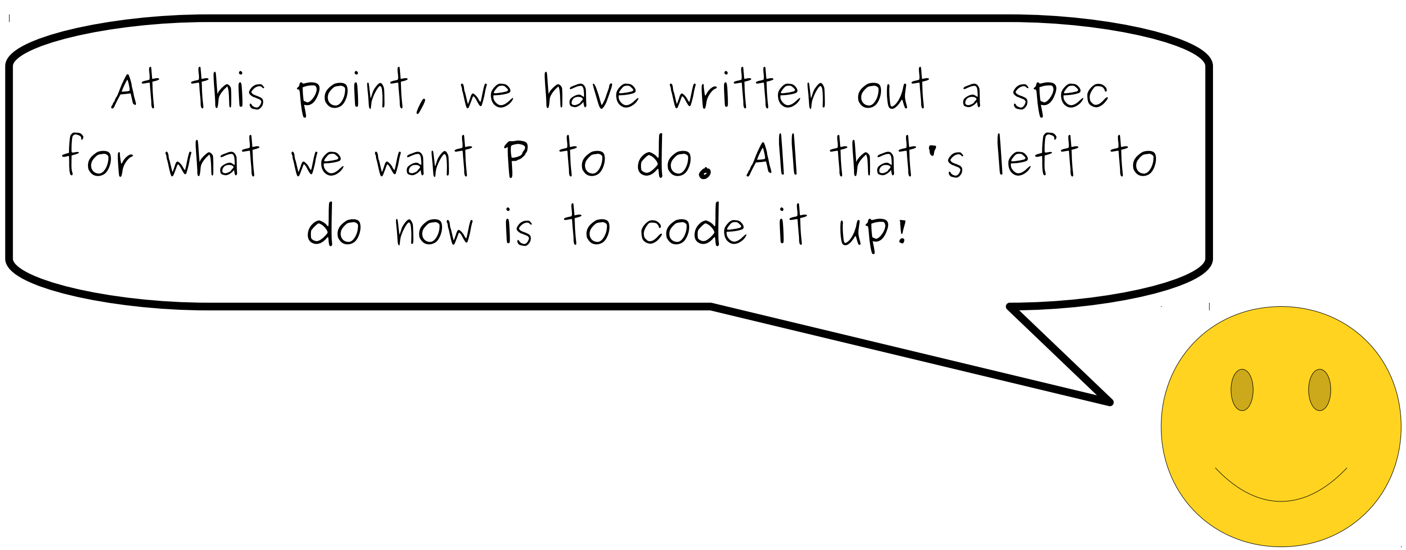
Contradiction!




```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

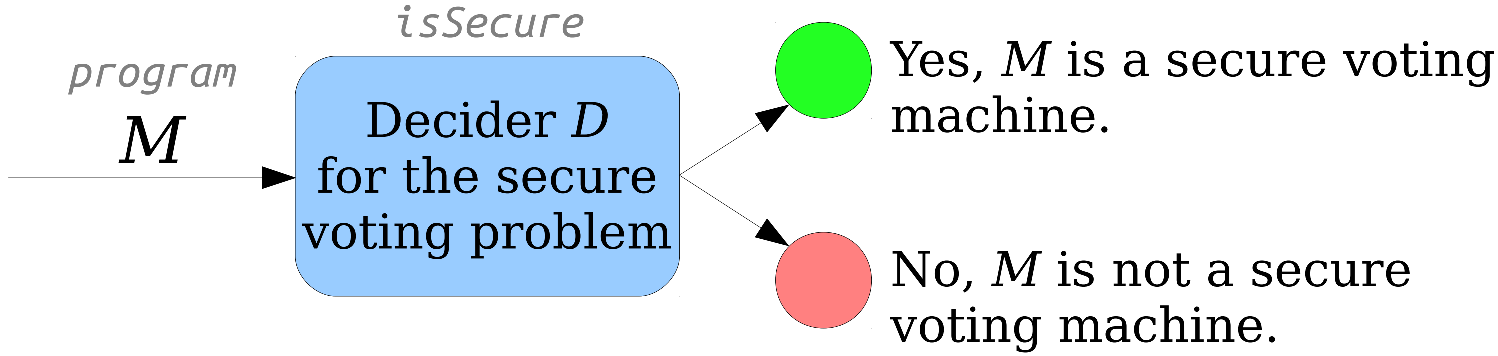
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

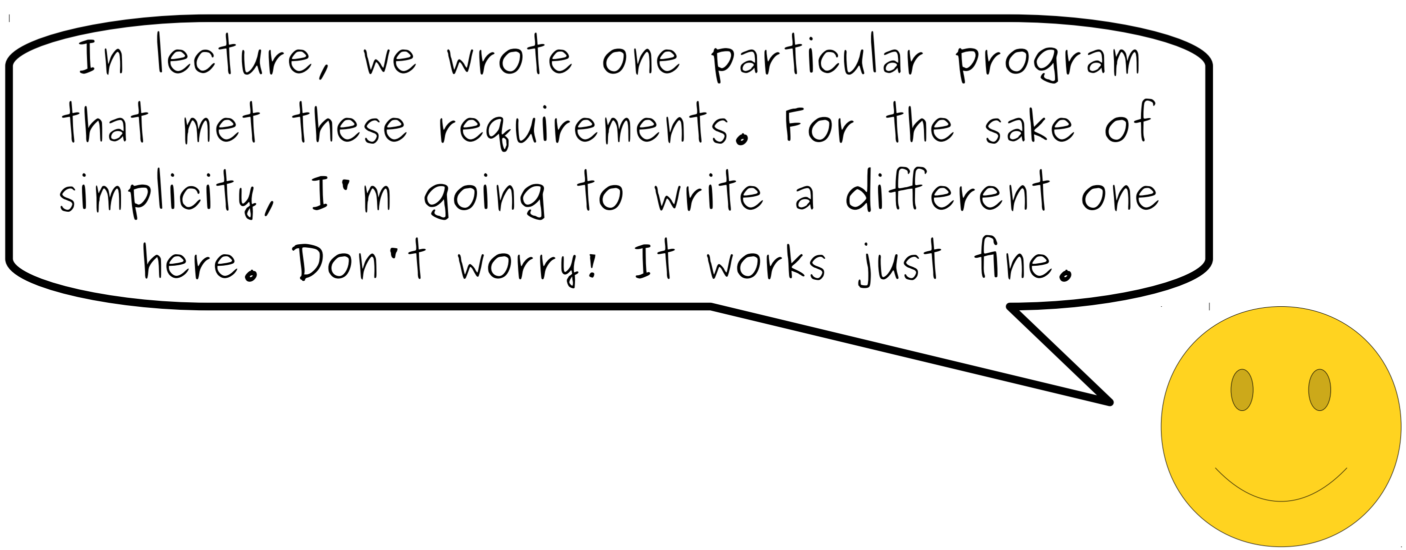
Contradiction!




```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.



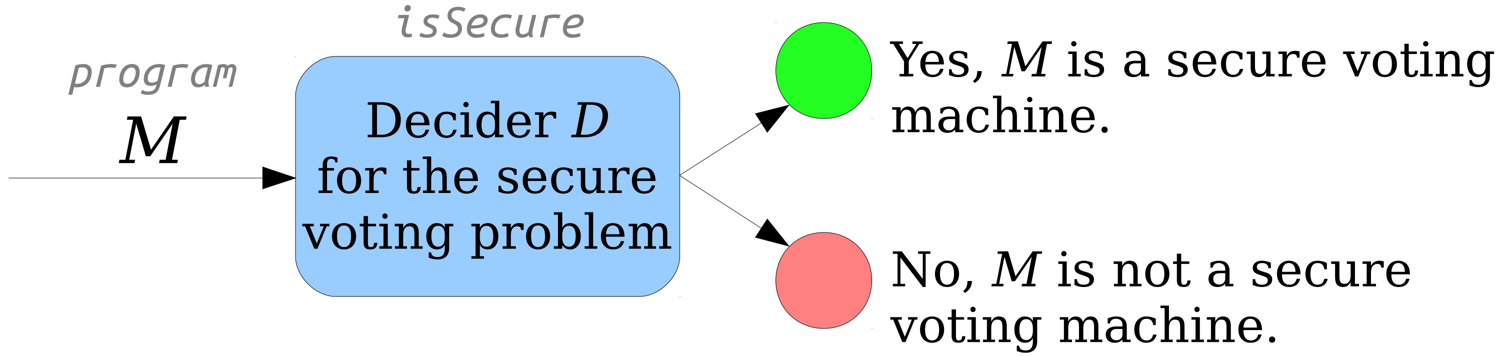
The secure voting problem is decidable.

There is a decider *D* for the secure voting problem

We can write programs that use *D* as a helper method

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

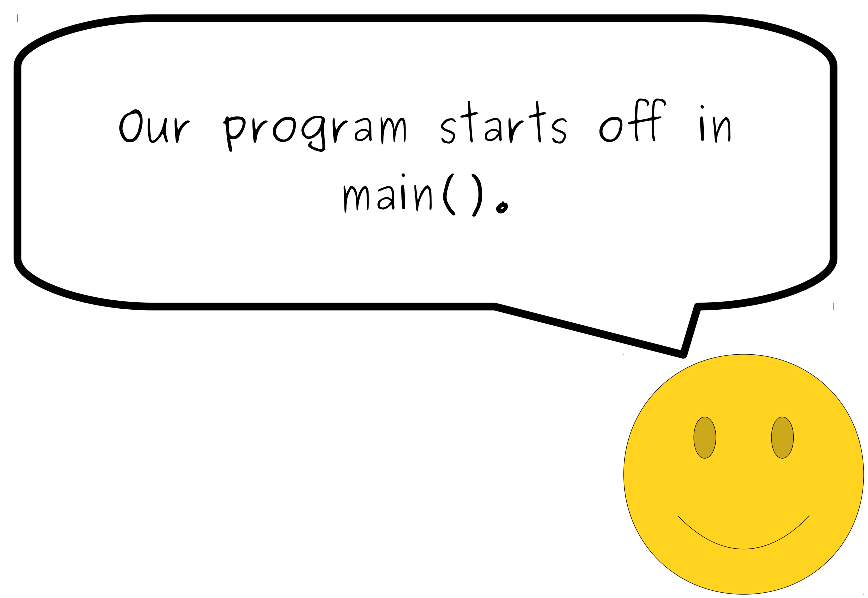



```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {



}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

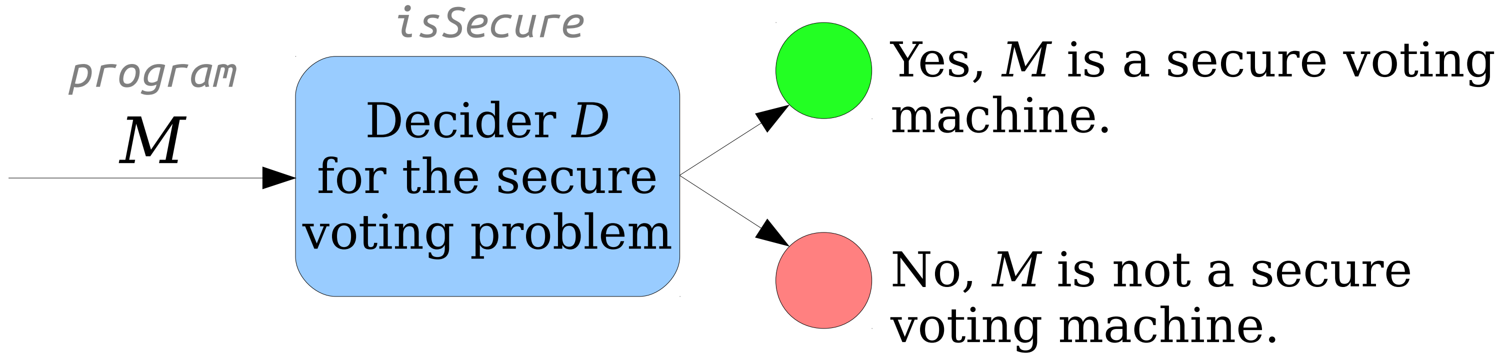
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

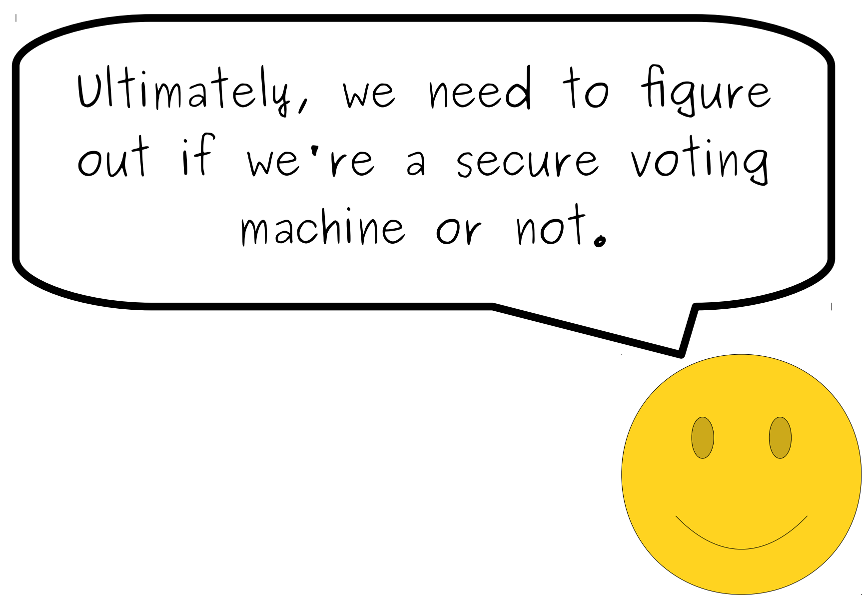



```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

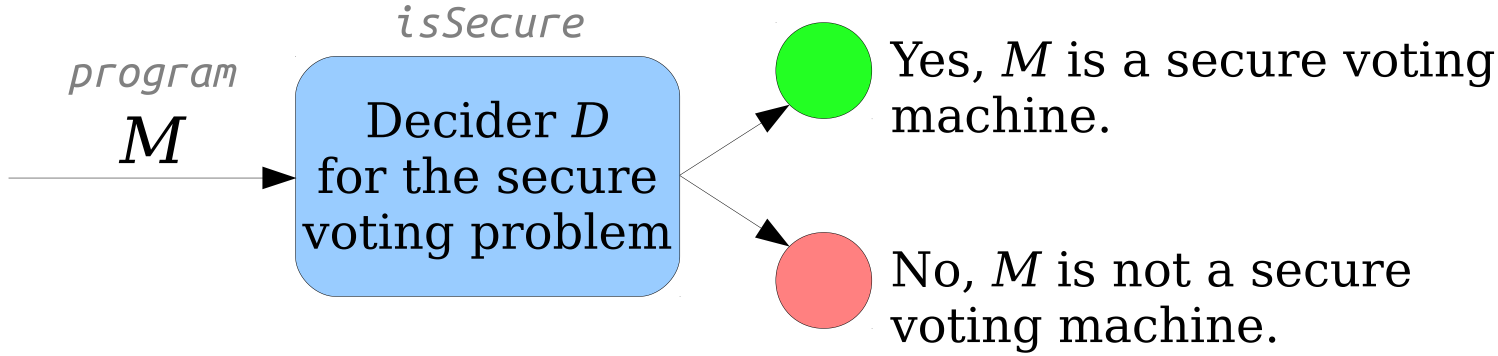
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!




```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();



}
```

The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

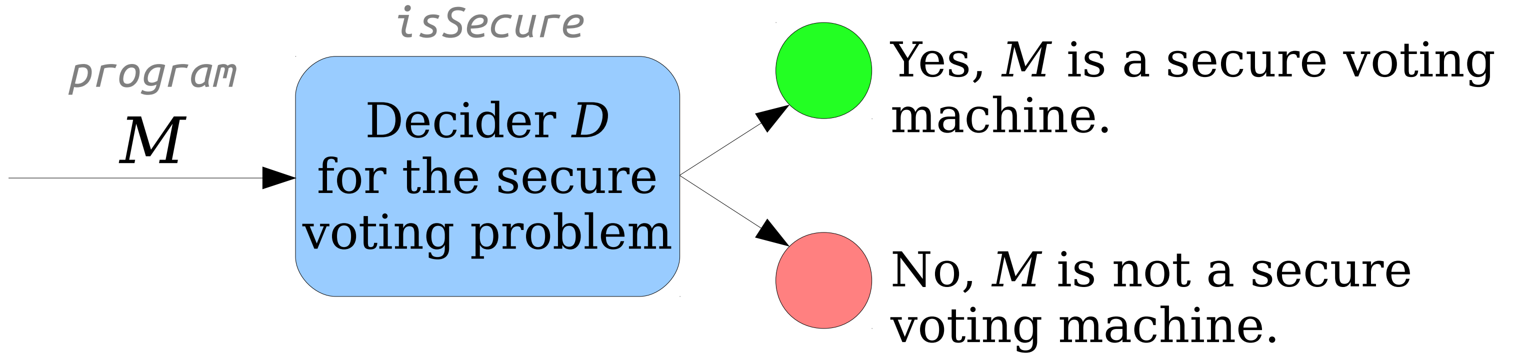
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

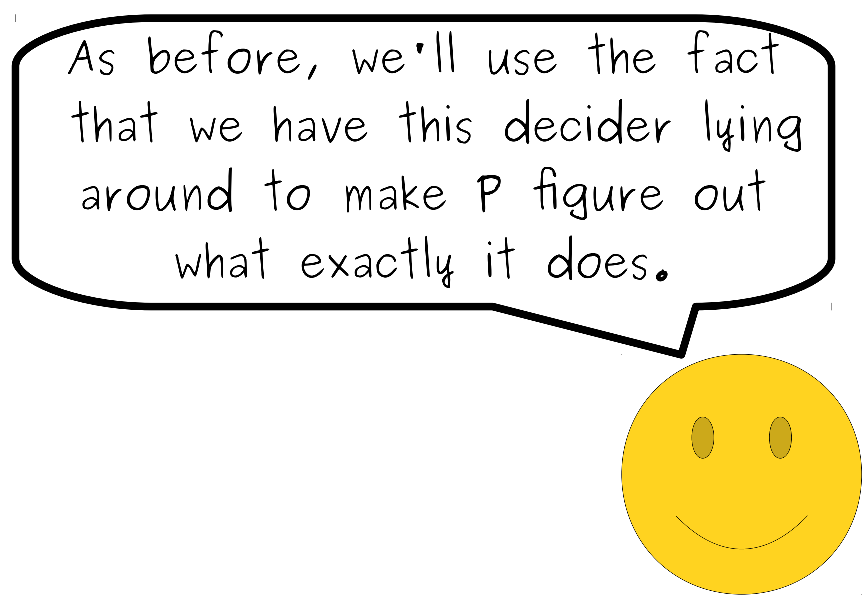



```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();




}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

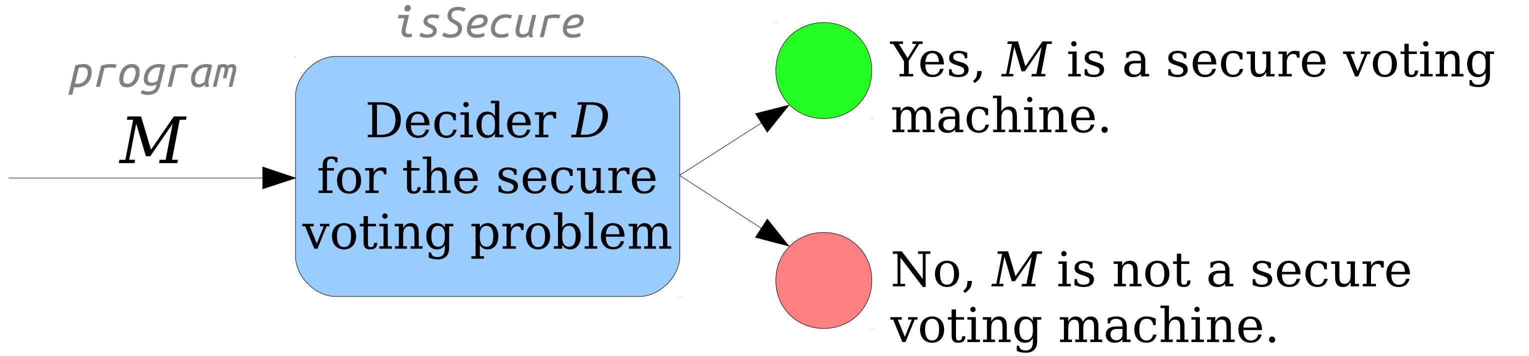
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

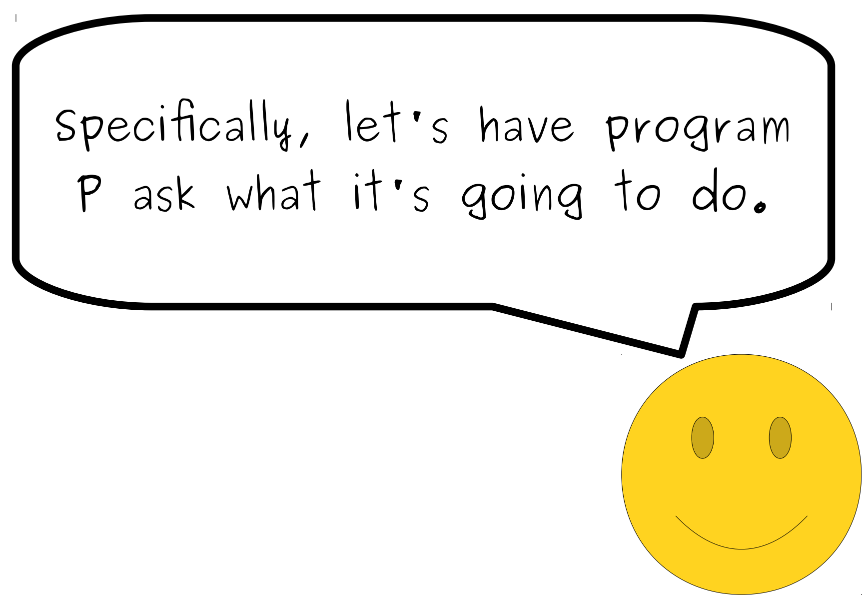



```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();

  if (isSecure(me)) {

  } else {


  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

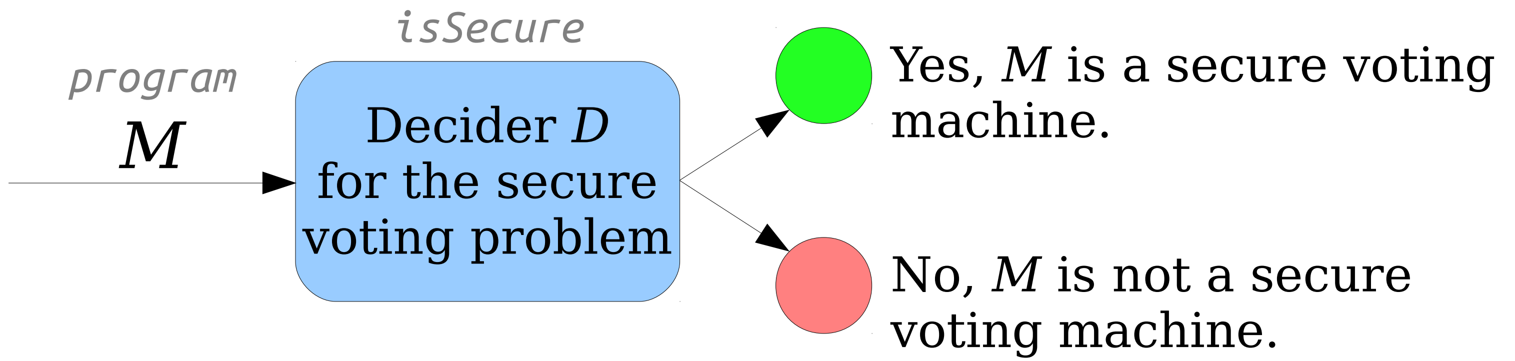
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

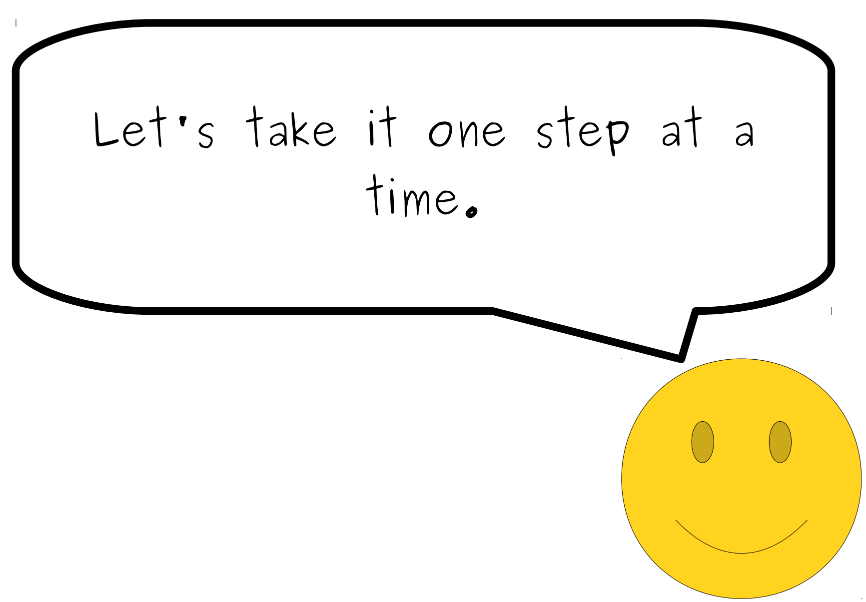



```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.
If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();

  if (isSecure(me)) {

  } else {


  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

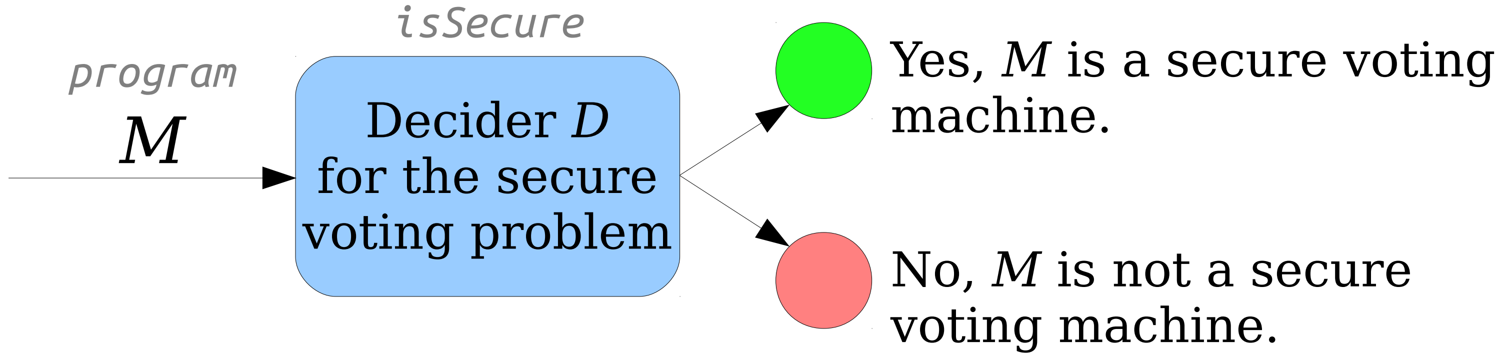
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



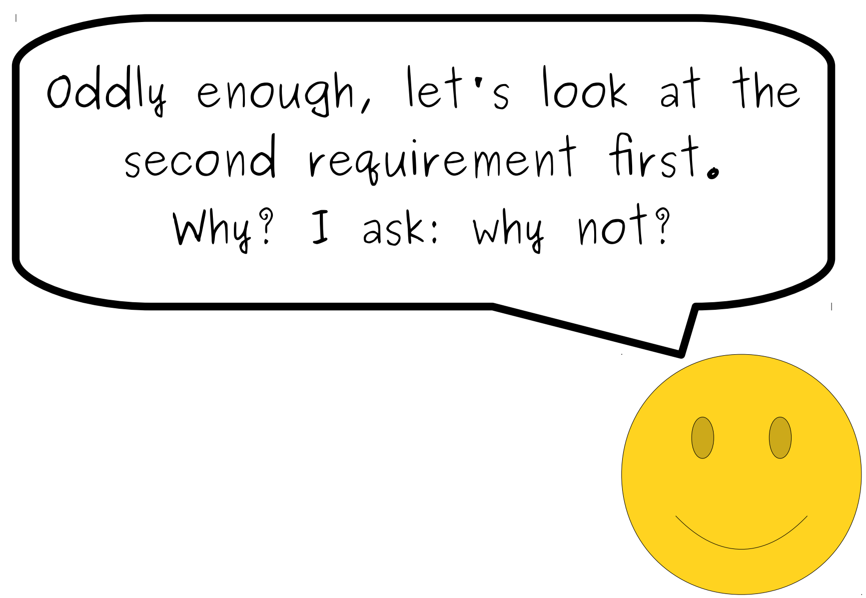

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();

  if (isSecure(me)) {

  } else {


  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

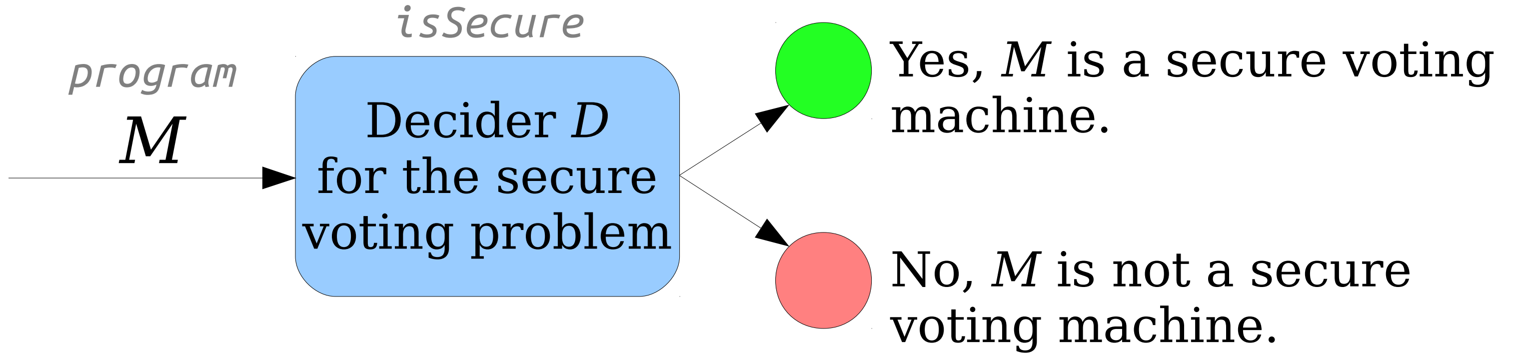
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



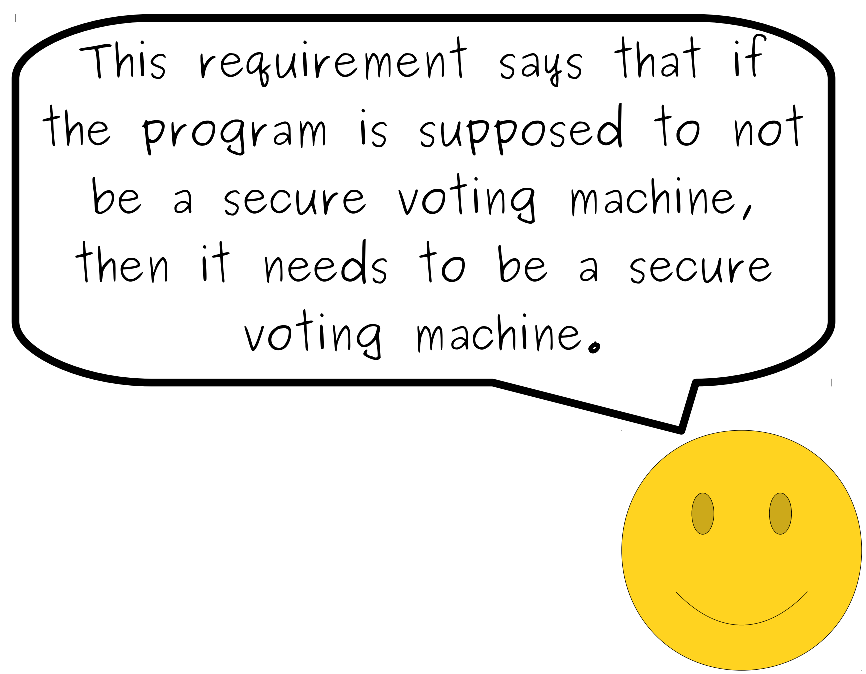

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then *P* is not a secure voting machine.

If *P* is not a secure voting machine, then *P* is a secure voting machine.

```
// Program P

int main() {

   string me = mySource();

   if (isSecure(me)) {

   } else {


   }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

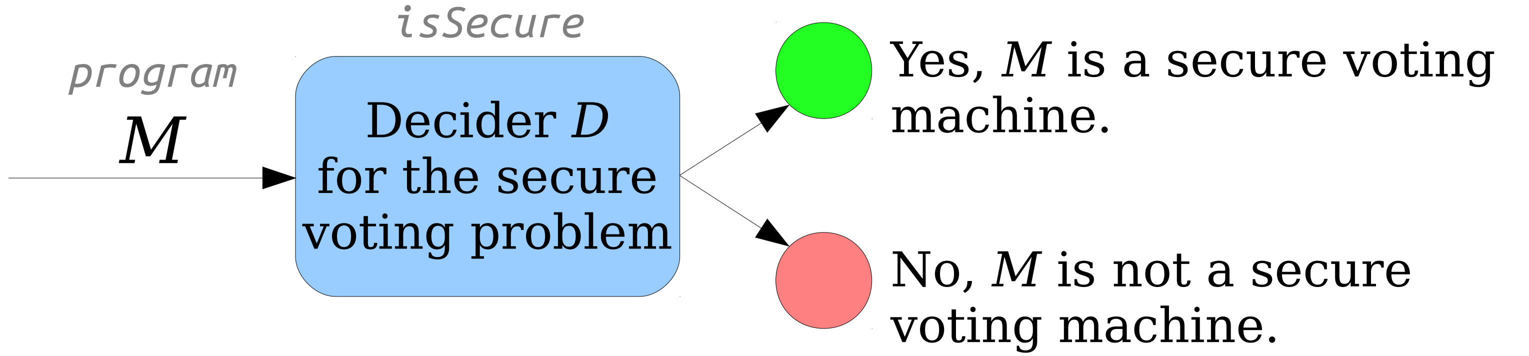
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!




```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();

  if (isSecure(me)) {

  } else {


  }
}
```

The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

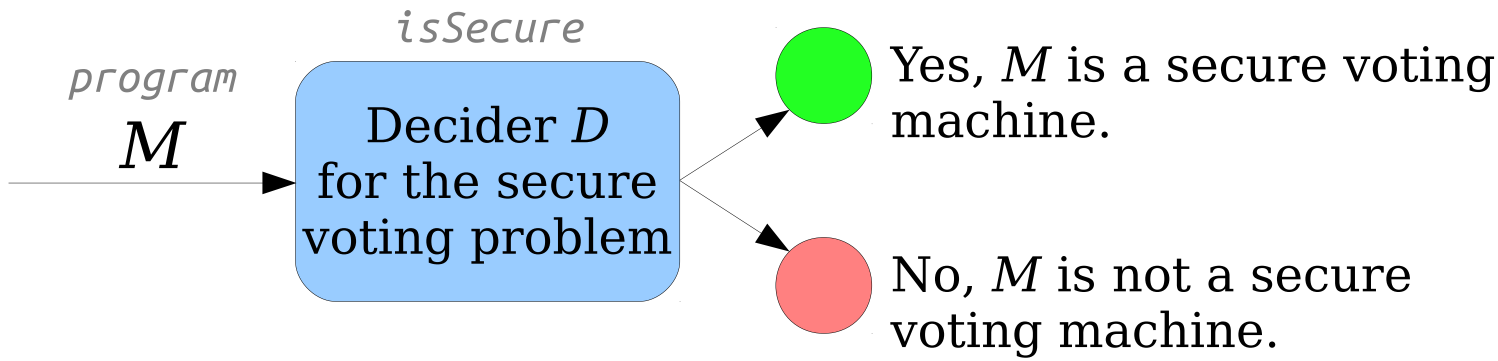
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



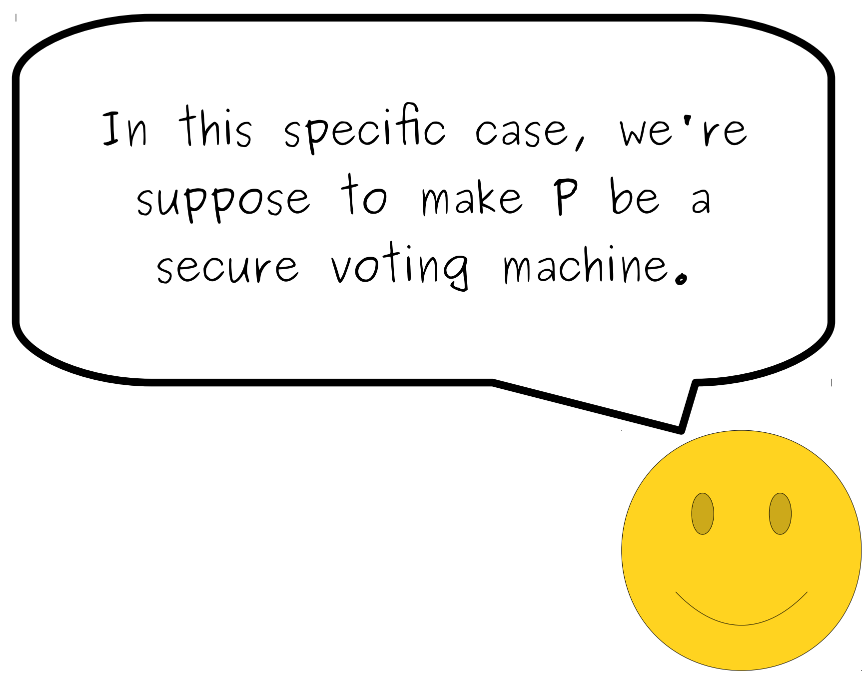

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();

  if (isSecure(me)) {

  } else {


  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

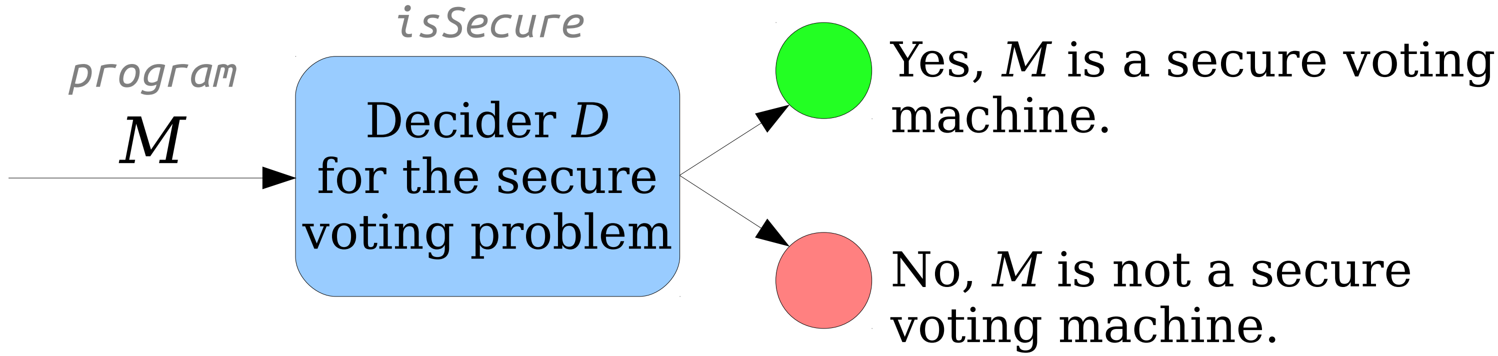
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



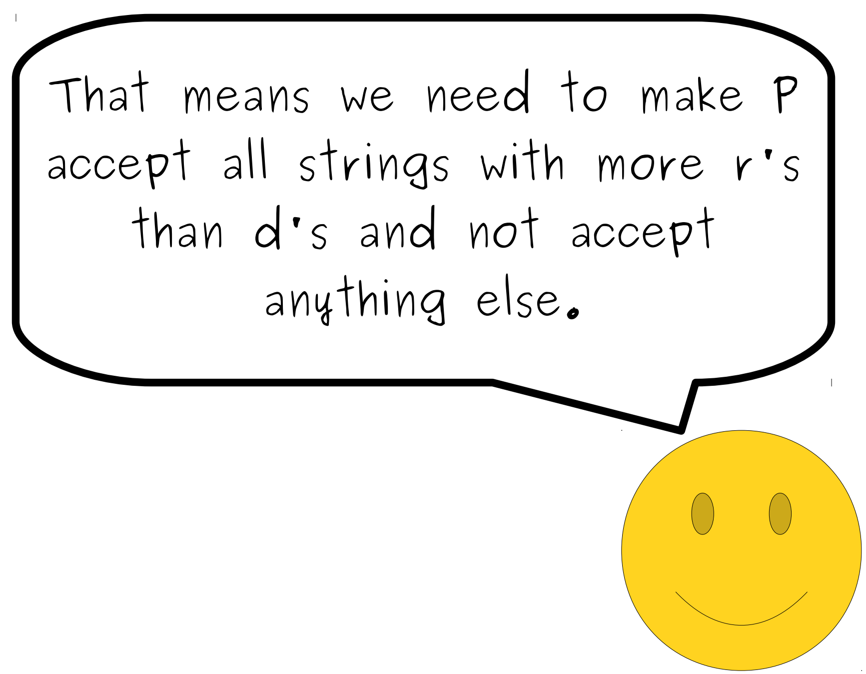

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then *P* is not a secure voting machine.

If *P* is not a secure voting machine, then *P* is a secure voting machine.

```
// Program P

int main() {

   string me = mySource();

   if (isSecure(me)) {

   } else {


   }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

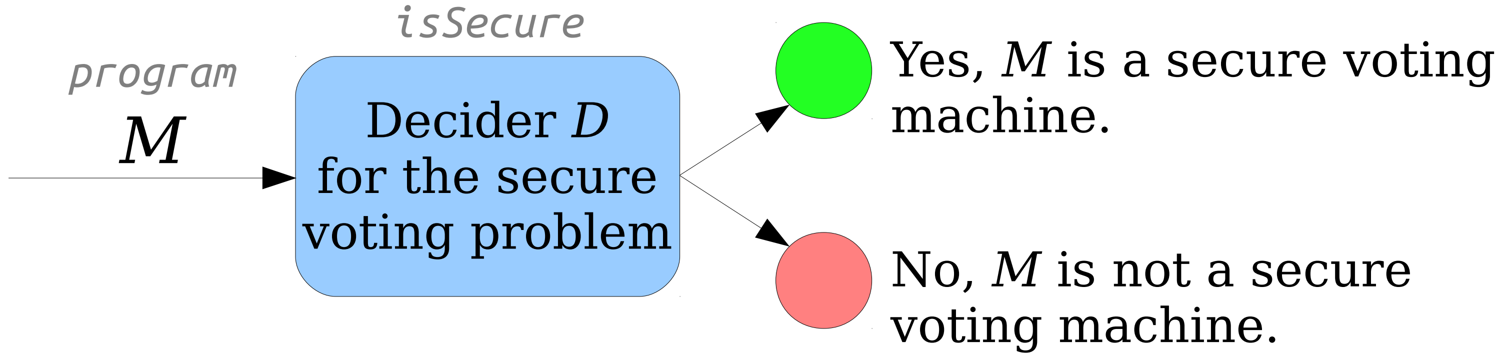
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!




```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {

  string me = mySource();

  if (isSecure(me)) {

  } else {


  }
}
```

The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

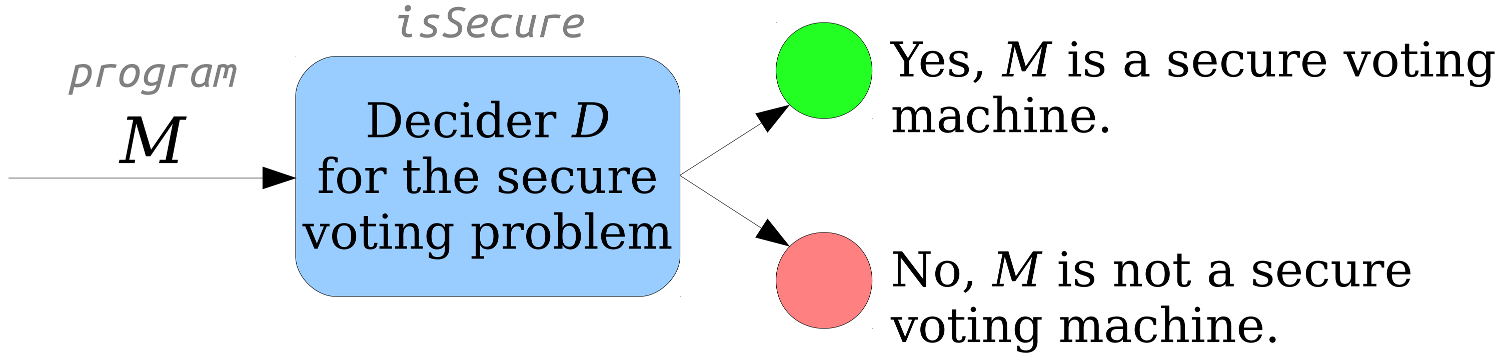
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



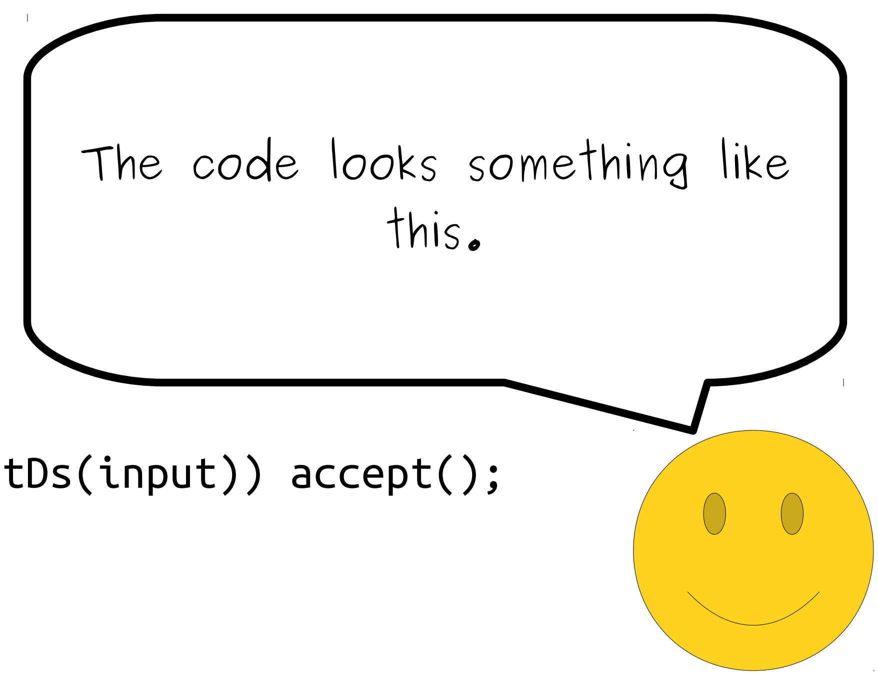

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {

  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

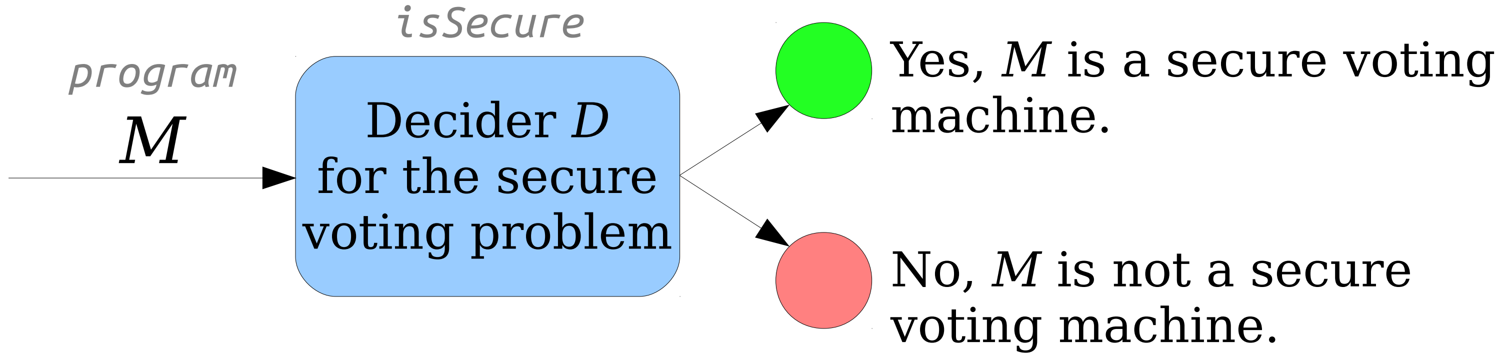
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



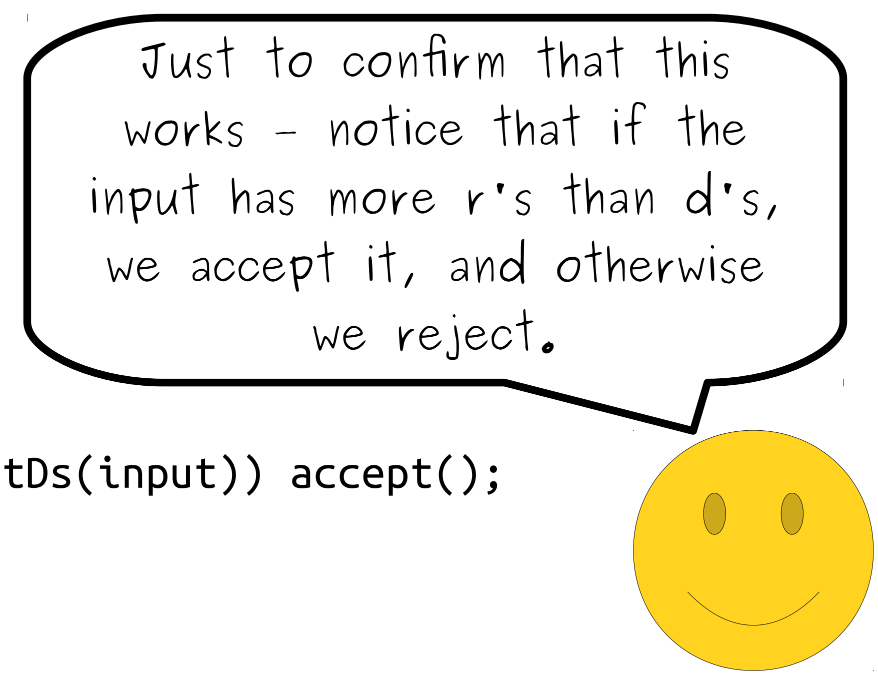

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {

  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

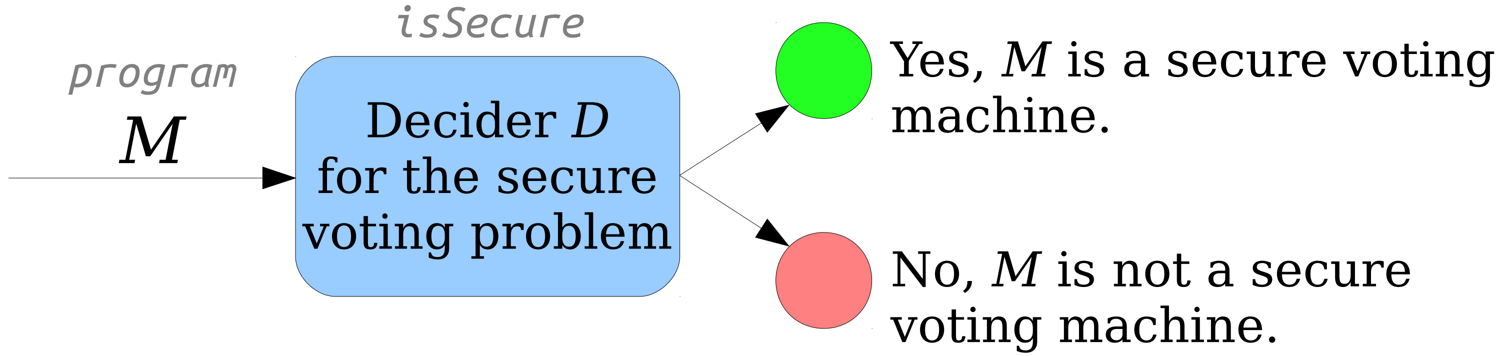
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



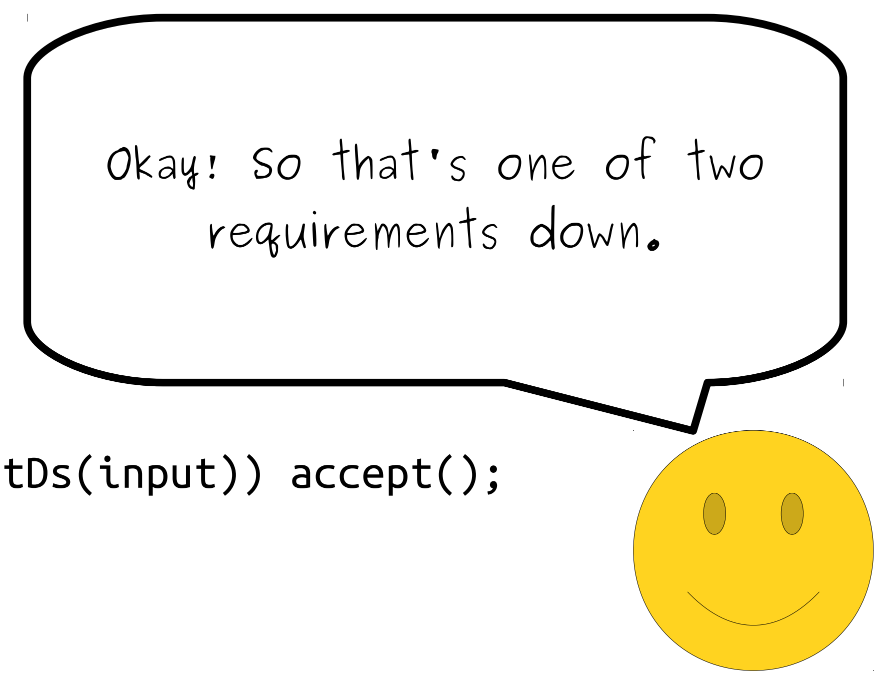

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

✓ If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {

  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

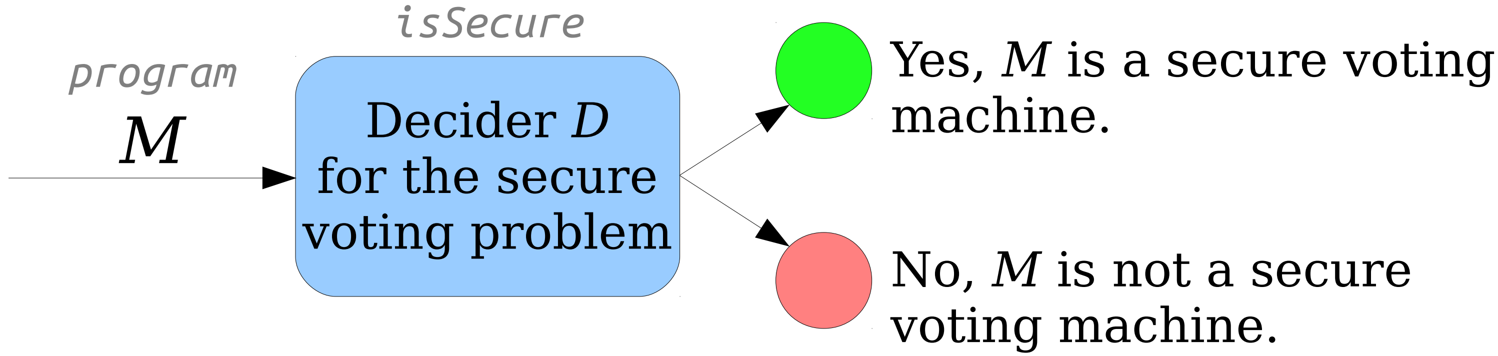
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



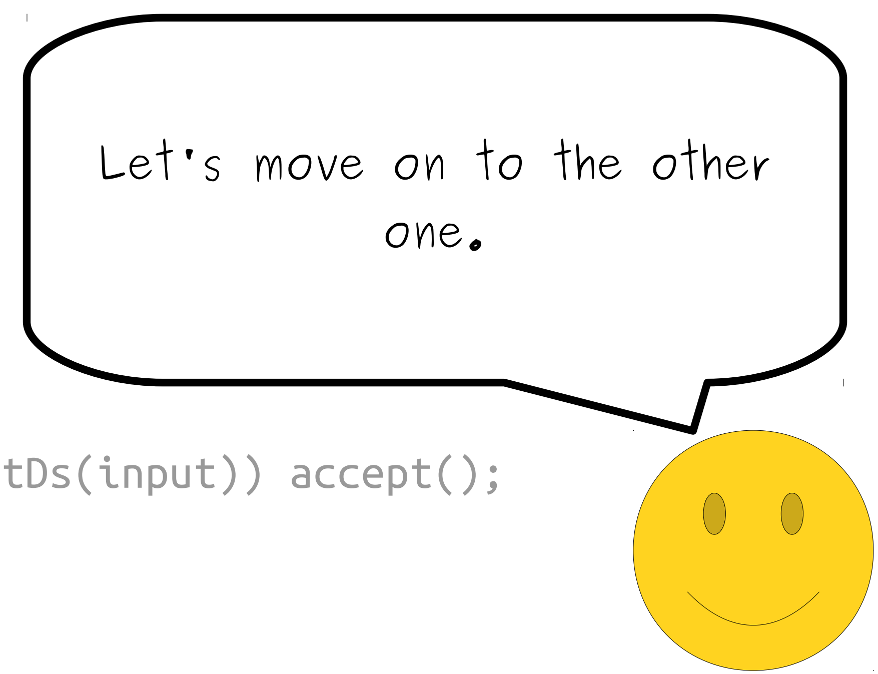

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

✓ If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {

  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

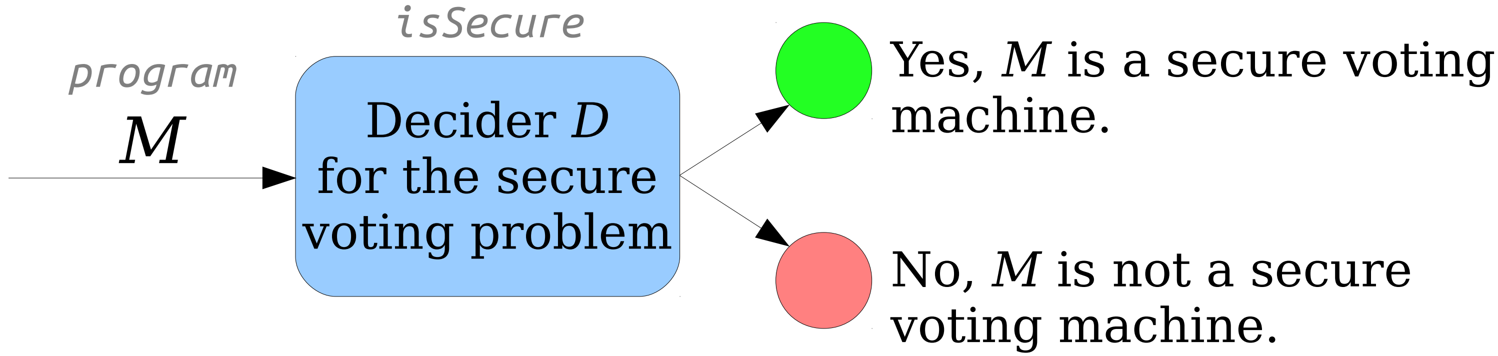
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



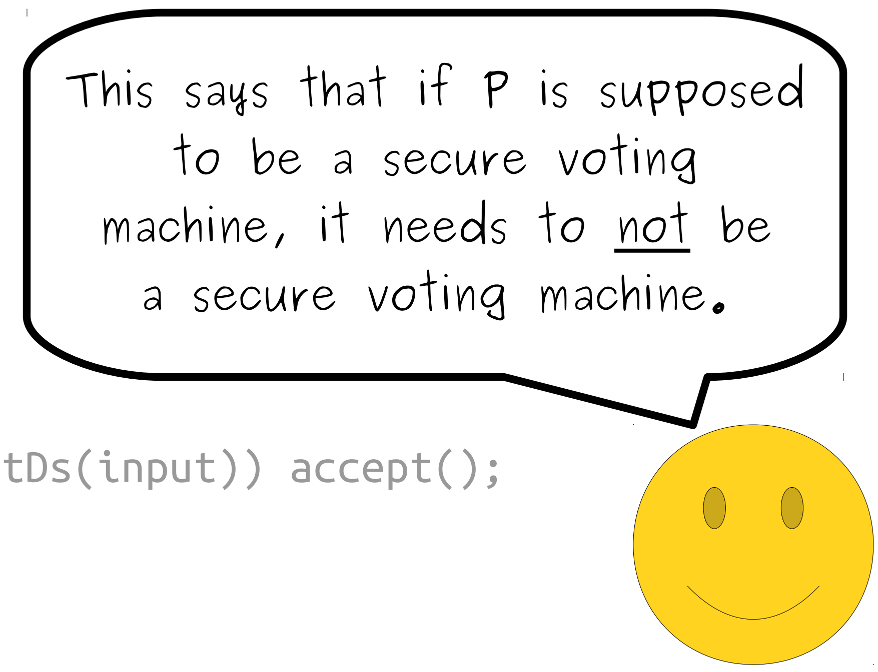

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

✓ If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {

  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

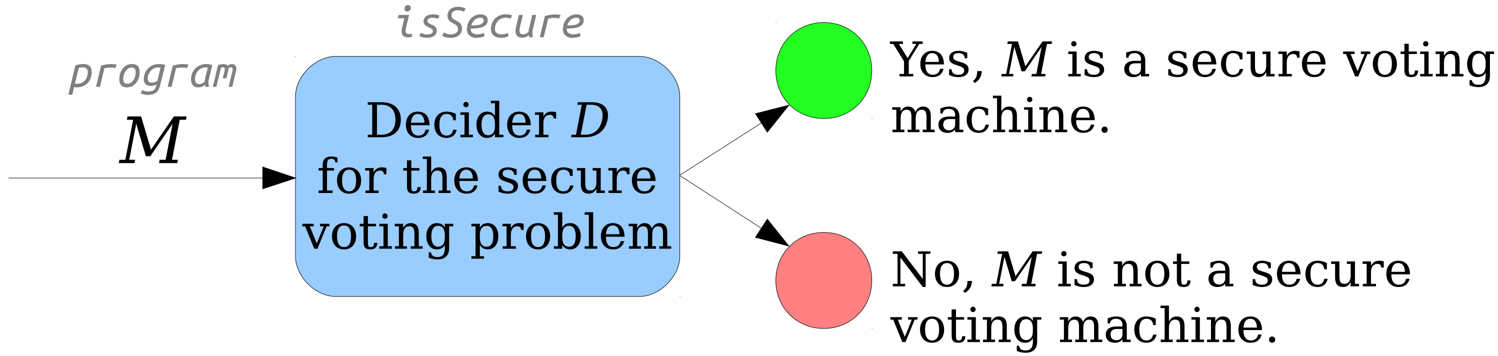
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



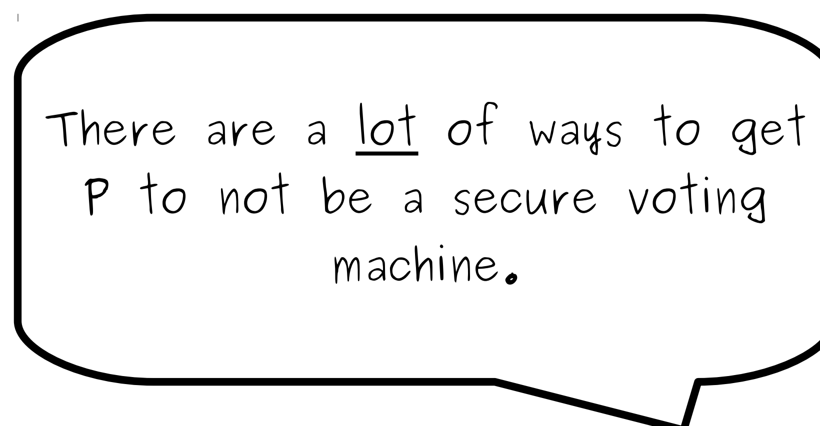

```
bool isSecure(string program)
```

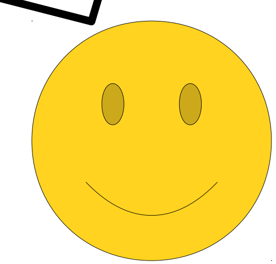
Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

✓ If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {

  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```





The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

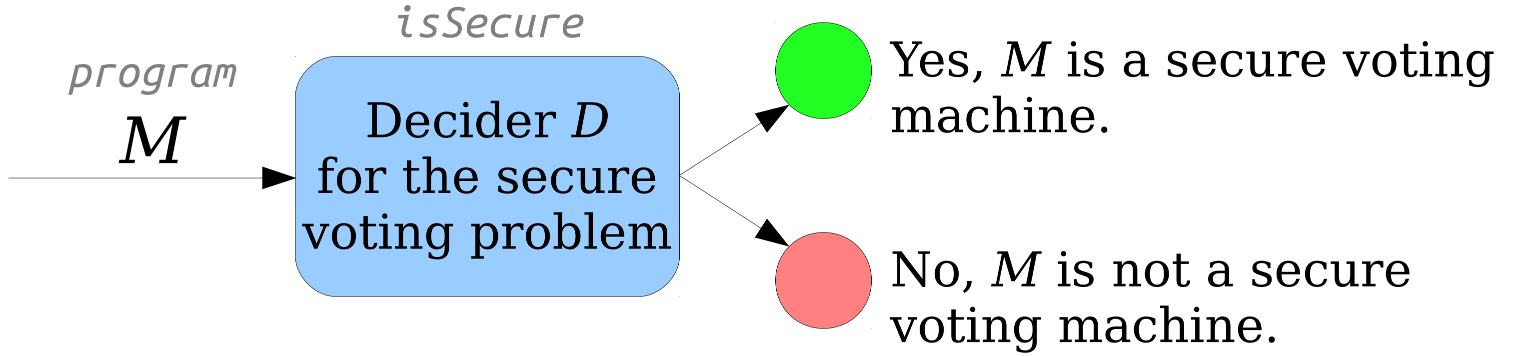
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

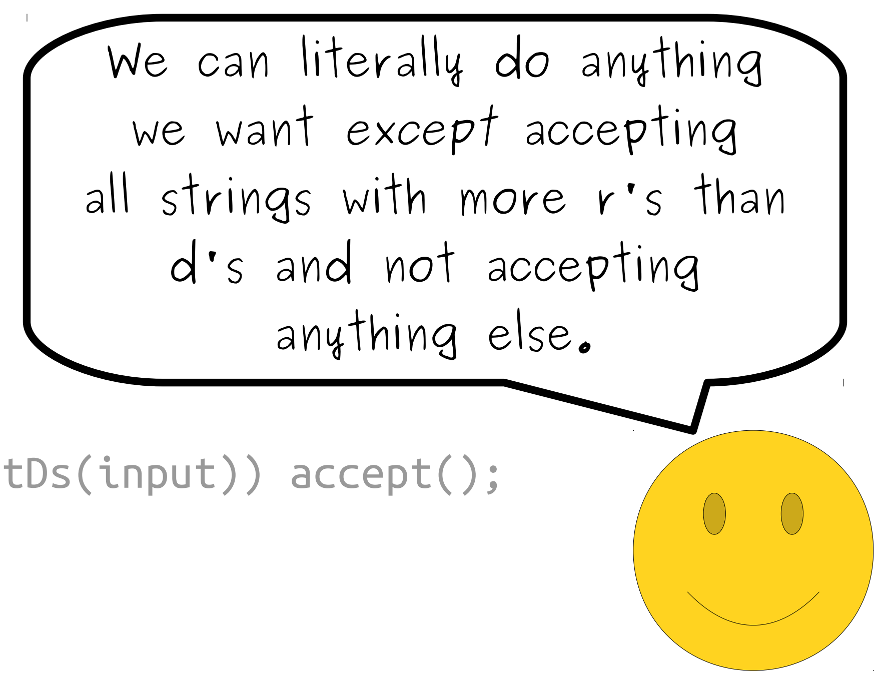



```
bool isSecure(string program)
```

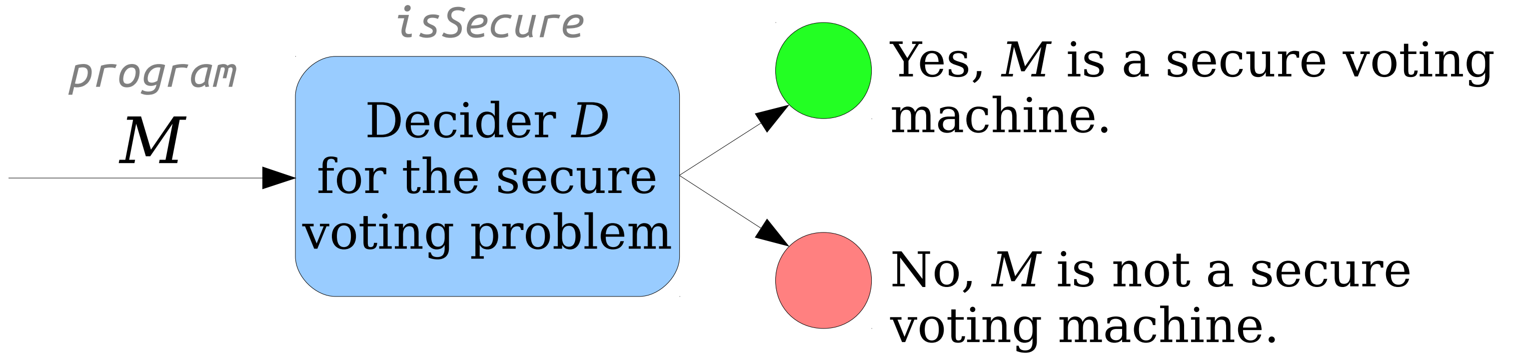
Program *P* design specification:

- If *P* is a secure voting machine, then *P* is not a secure voting machine.
- ✓ If *P* is not a secure voting machine, then *P* is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {

  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.



There is a decider *D* for the secure voting problem

```
bool isSecure(string program)
```

We can write programs that use *D* as a helper method

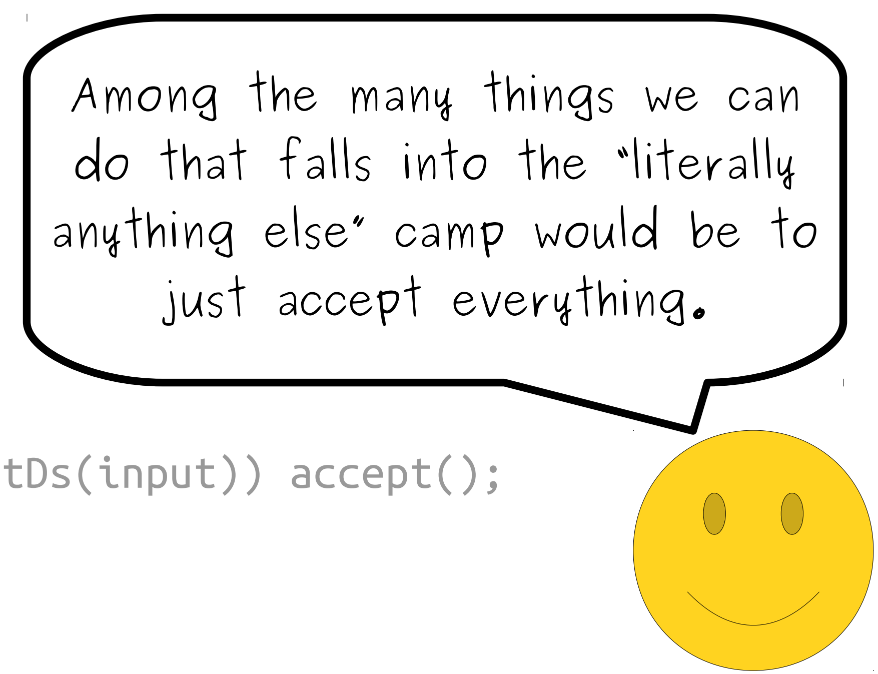
Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

✓ If *P* is not a secure voting machine, then
P is a secure voting machine.

Program *P* is secure if and only if program *P* is not secure.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {
    accept();
  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



Contradiction!

The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

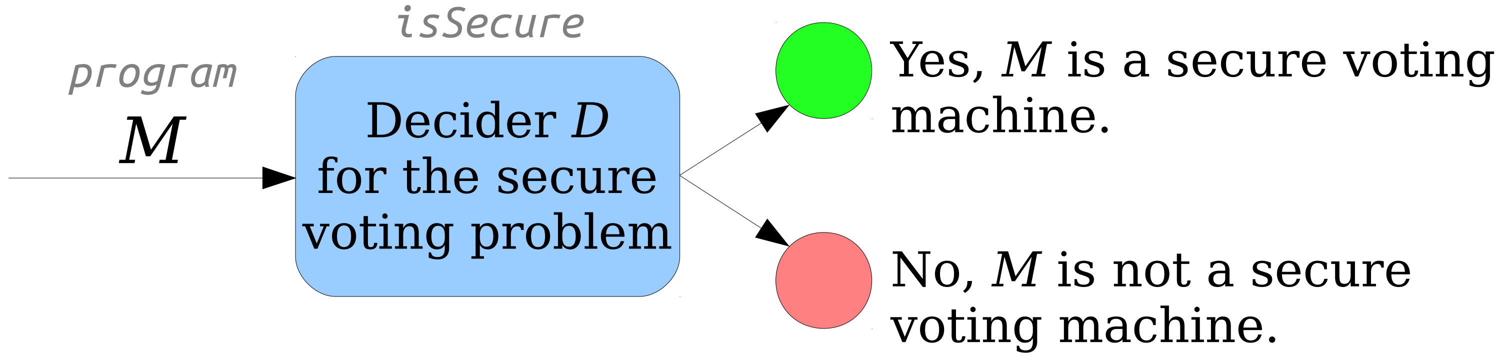
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!



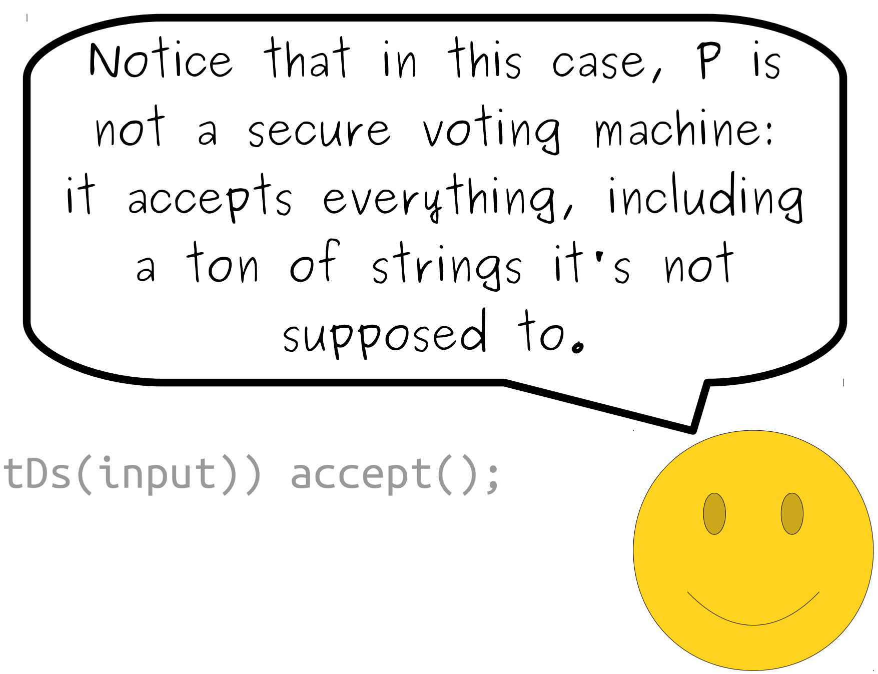

```
bool isSecure(string program)
```

Program *P* design specification:

If *P* is a secure voting machine, then
P is not a secure voting machine.

✓ If *P* is not a secure voting machine, then
P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {
    accept();
  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

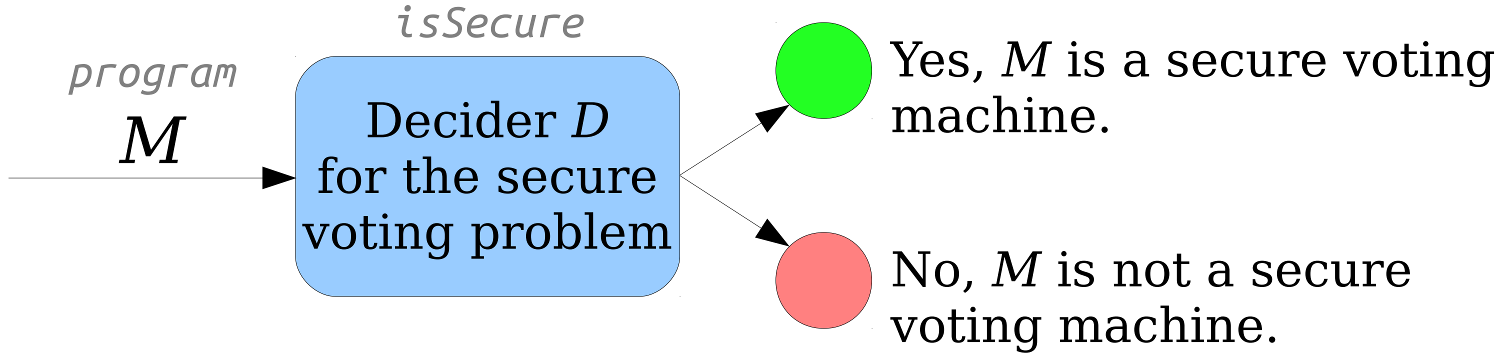
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

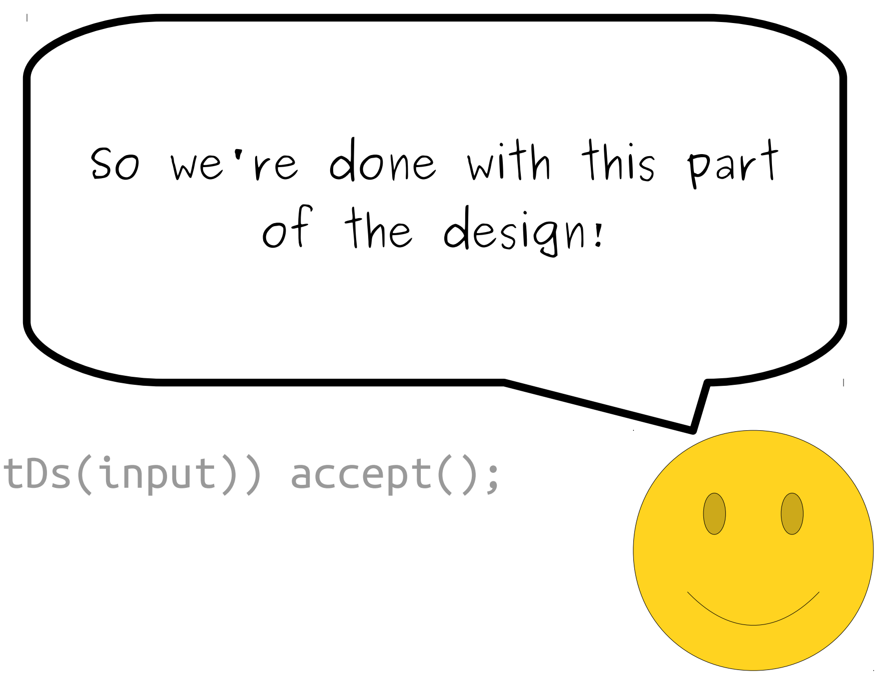



```
bool isSecure(string program)
```

Program *P* design specification:

- ✓ If *P* is a secure voting machine, then *P* is not a secure voting machine.
- ✓ If *P* is not a secure voting machine, then *P* is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {
    accept();
  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

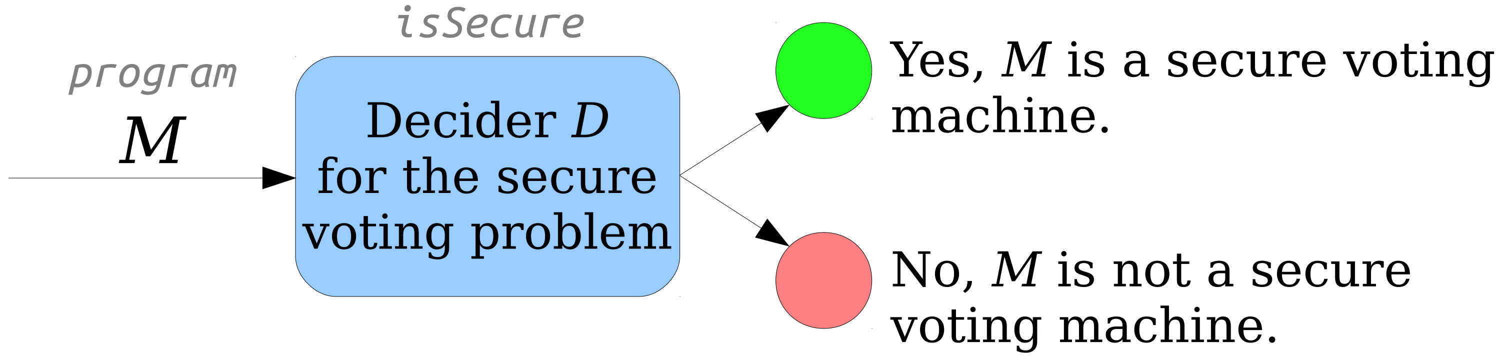
↓

We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

Contradiction!

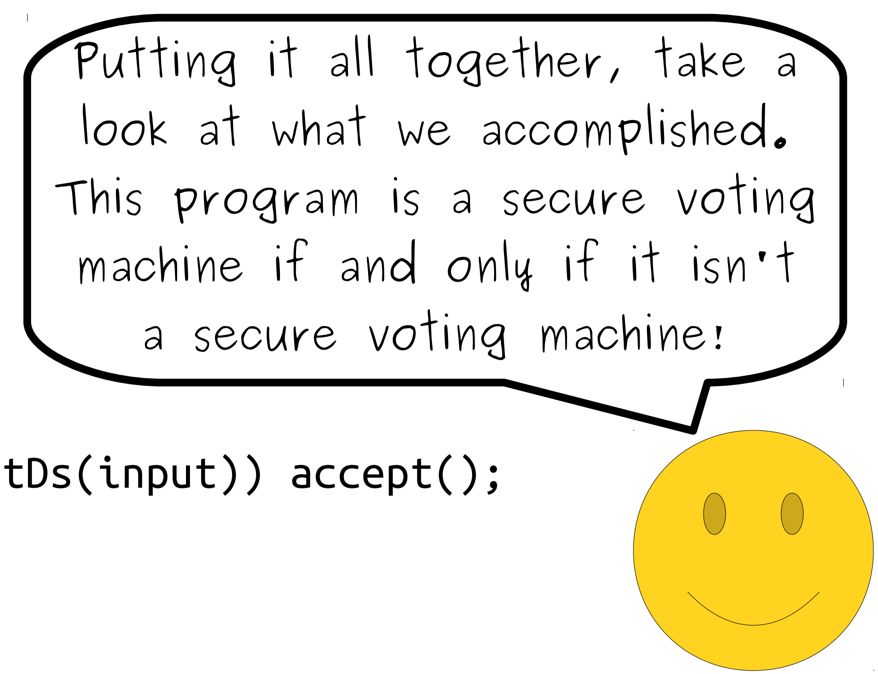



```
bool isSecure(string program)
```

Program *P* design specification:

- ✓ If *P* is a secure voting machine, then
  P is not a secure voting machine.
- ✓ If *P* is not a secure voting machine, then
  P is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {
    accept();
  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

↓

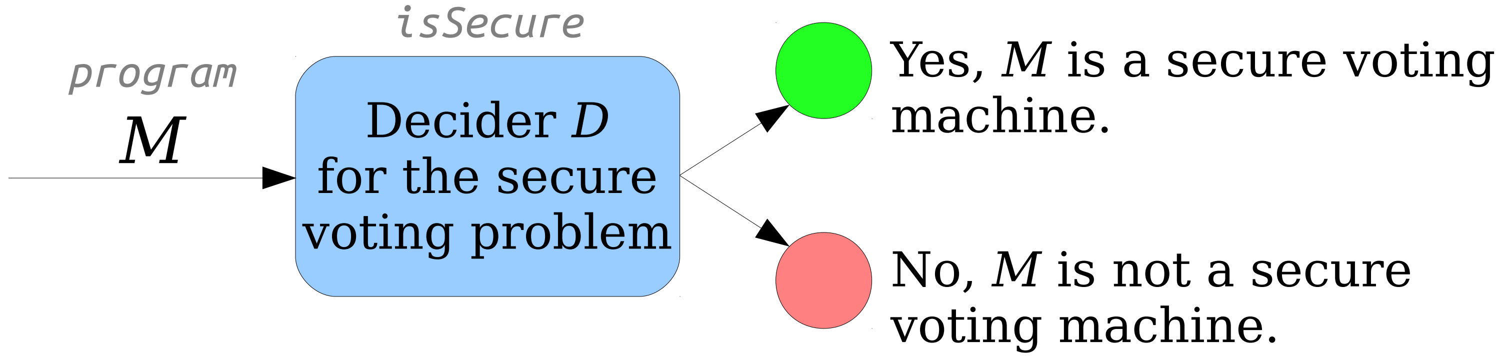
We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

↓

Contradiction!

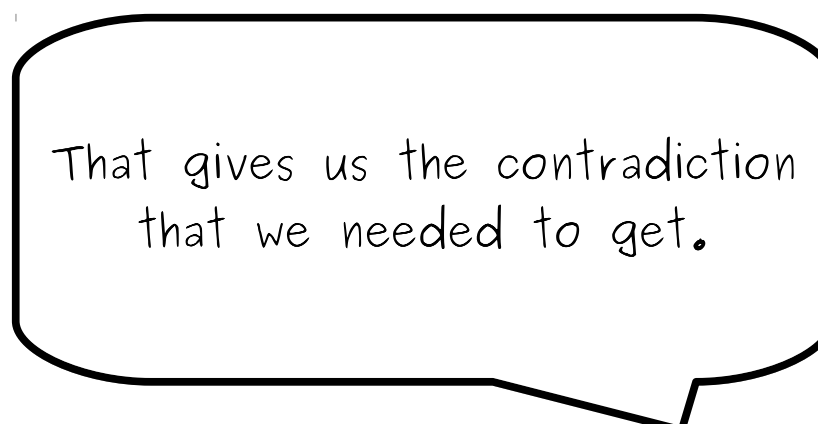


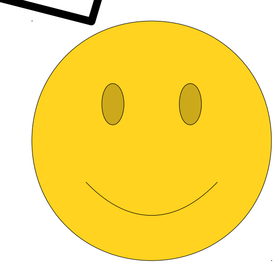

```
bool isSecure(string program)
```

Program *P* design specification:

- ✓ If *P* is a secure voting machine, then *P* is not a secure voting machine.
- ✓ If *P* is not a secure voting machine, then *P* is a secure voting machine.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {
    accept();
  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```





The secure voting problem is decidable.

↓

There is a decider *D* for the secure voting problem

↓

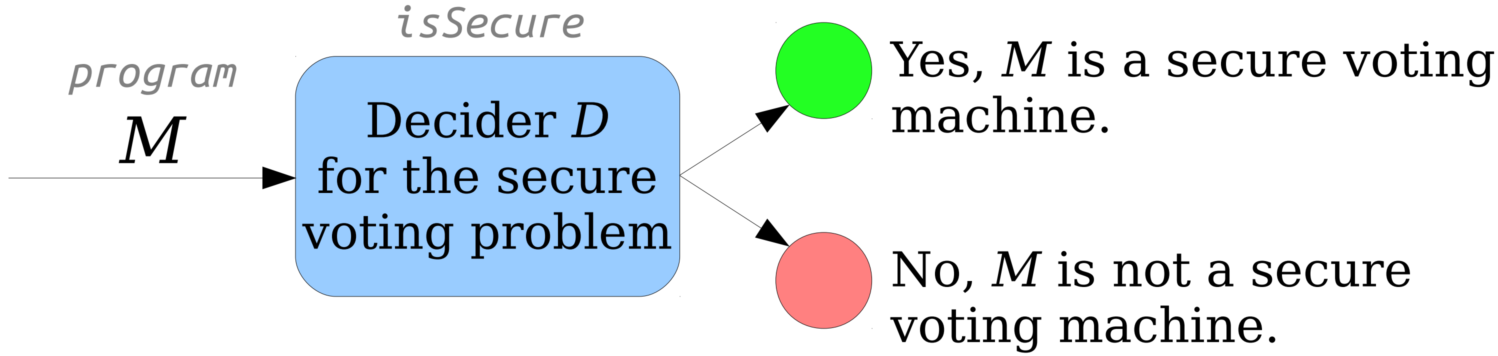
We can write programs that use *D* as a helper method

↓

Program *P* is secure if and only if program *P* is not secure.

↓

Contradiction!



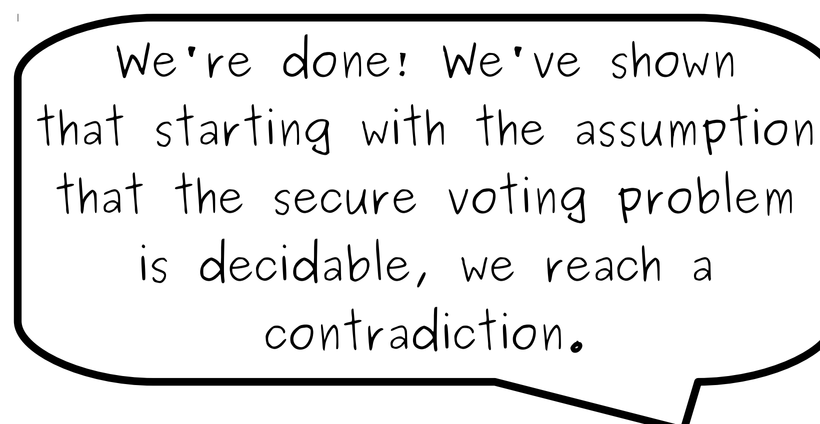

```
bool isSecure(string program)
```

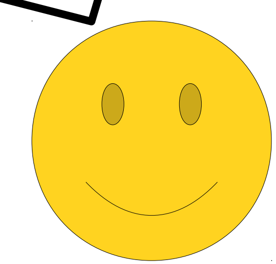
Program *P* design specification:

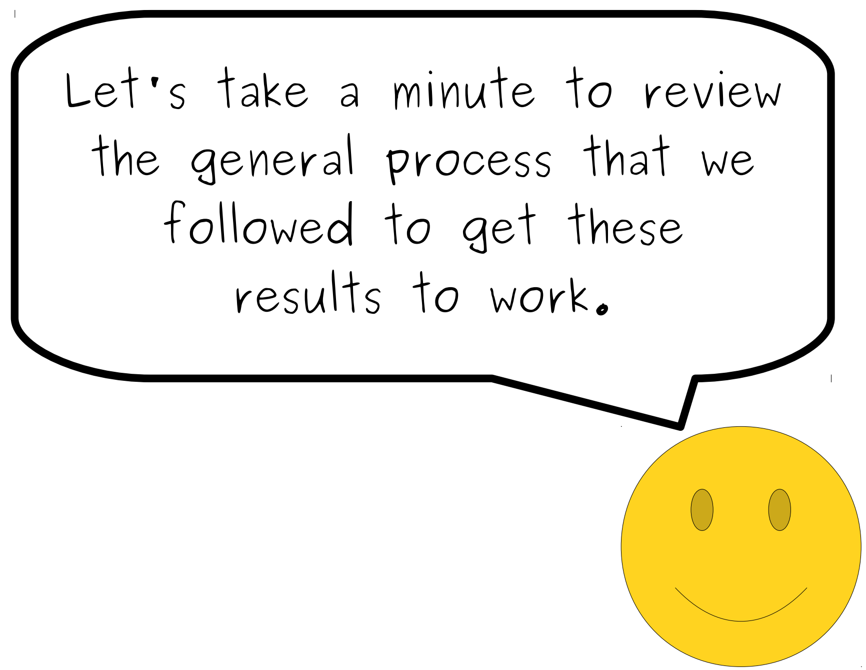
✓ If *P* is a secure voting machine, then
  P is not a secure voting machine.

✓ If *P* is not a secure voting machine, then
  P is a secure voting machine.

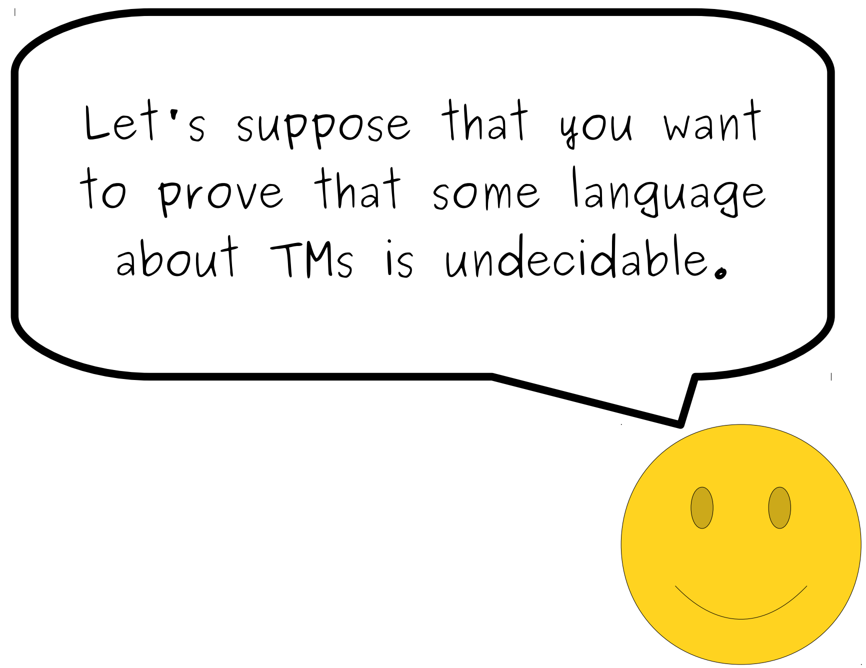
```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (isSecure(me)) {
    accept();
  } else {
    if (countRs(input) > countDs(input)) accept();
    else reject();
  }
}
```



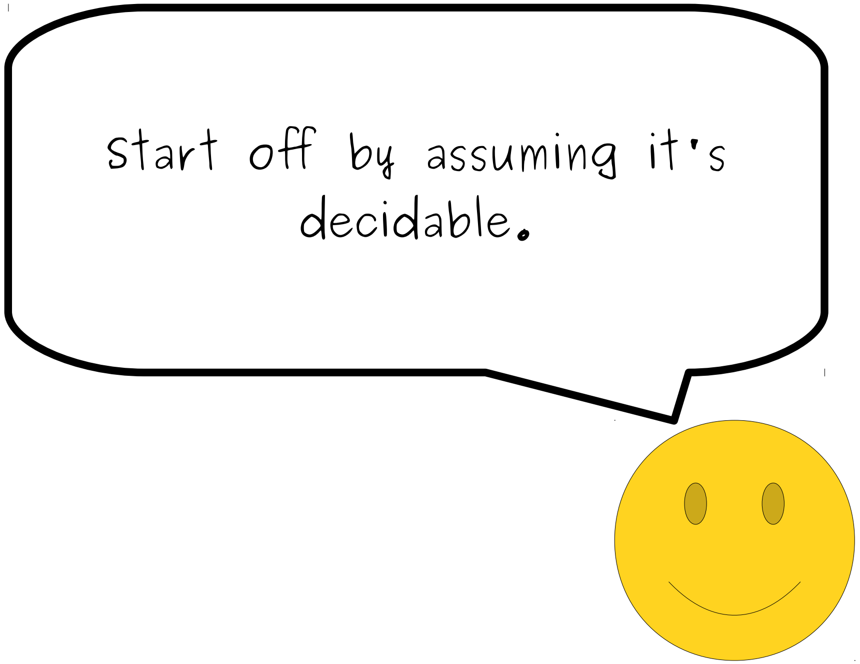



Let's take a minute to review the general process that we followed to get these results to work.


Let's take a minute to review the general process that we followed to get these results to work.
That other guy is going to tell you a general pattern to follow. You might want to take notes.


Let's suppose that you want to prove that some language about TMs is undecidable.

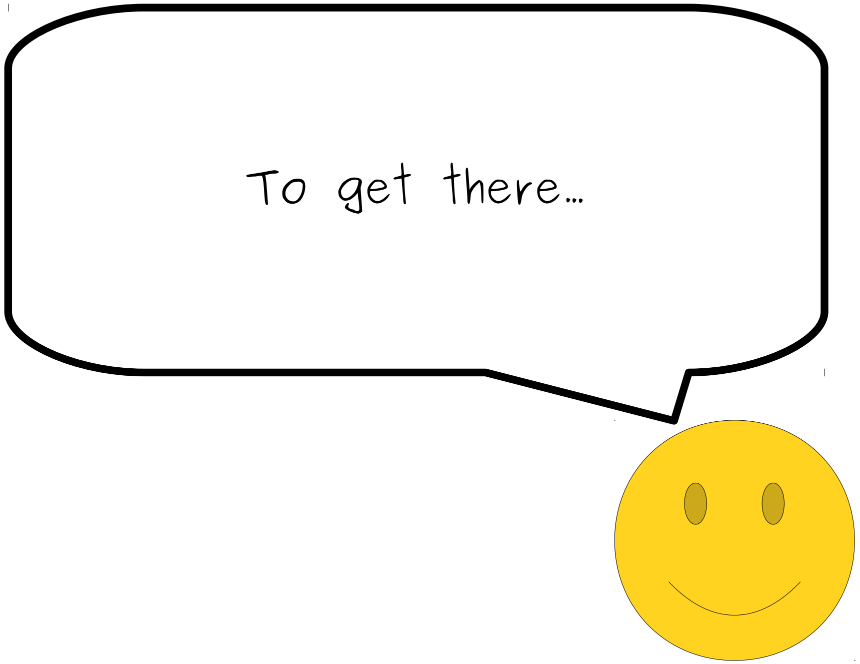
The problem in question is decidable



The problem in question is decidable

Contradiction!

The problem in question is decidable



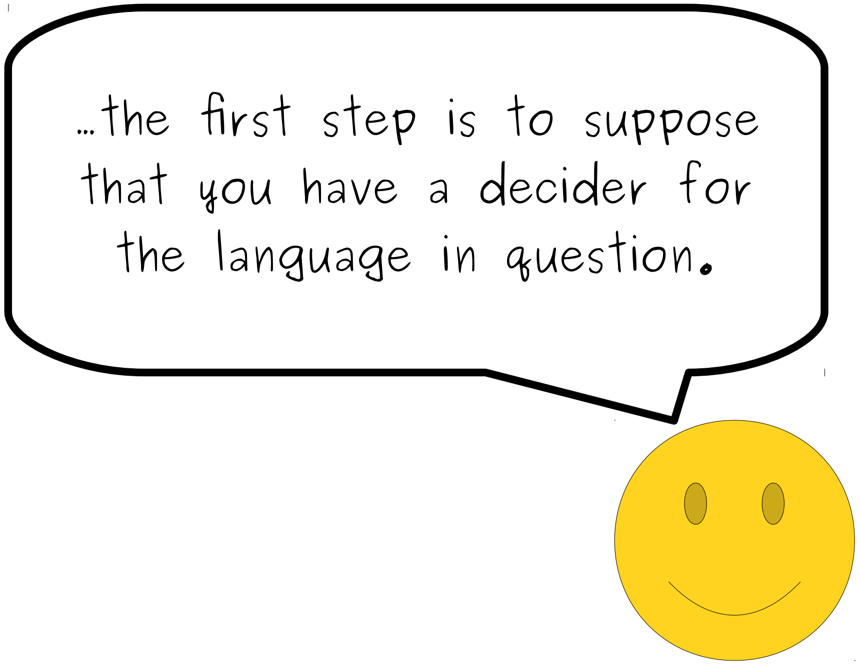
Contradiction!

The problem in question is decidable

↓

There is a decider *D* for that problem.

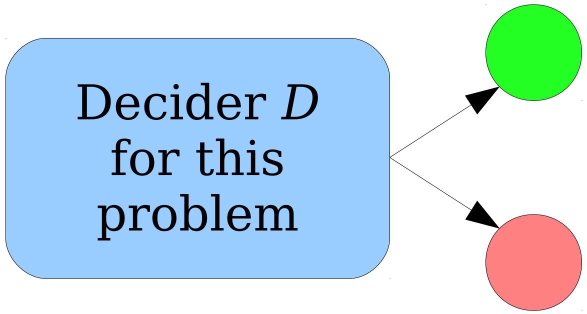
Contradiction!

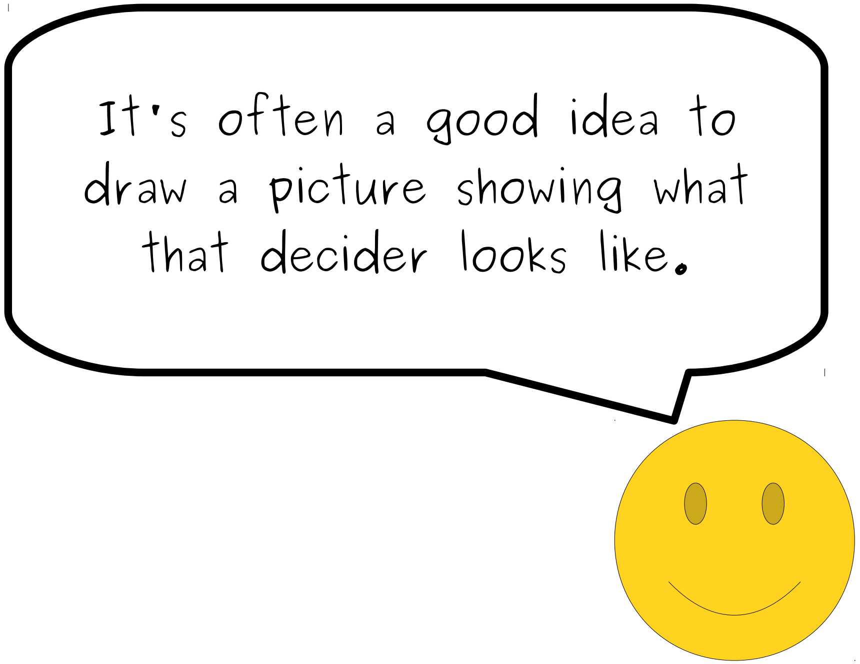


The problem in question is decidable

There is a decider *D* for that problem.

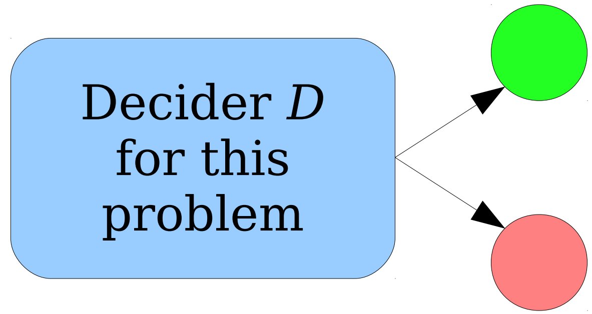
Contradiction!

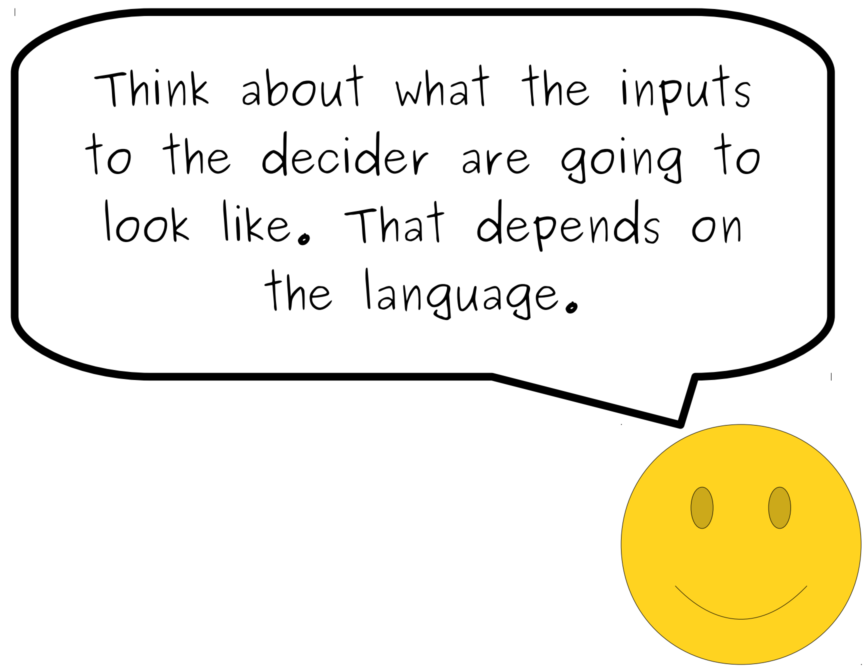




The problem in question is decidable

There is a decider *D* for that problem.



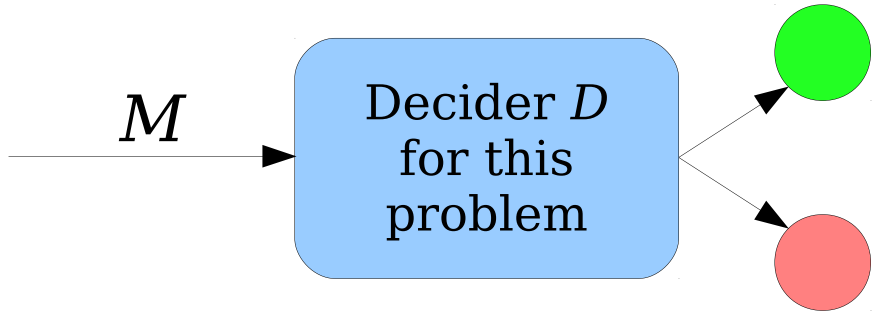


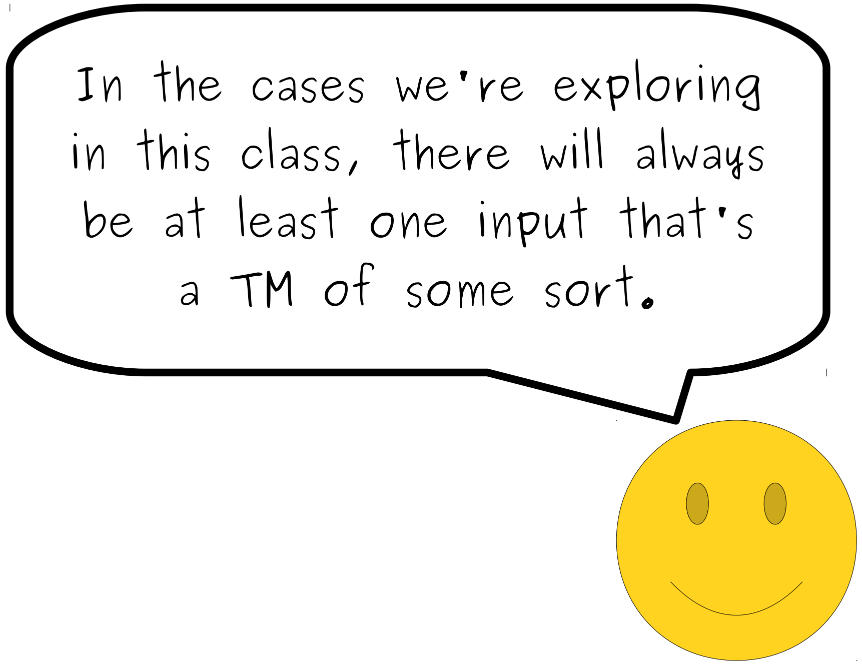
Contradiction!

The problem in question is decidable

↓

There is a decider *D* for that problem.

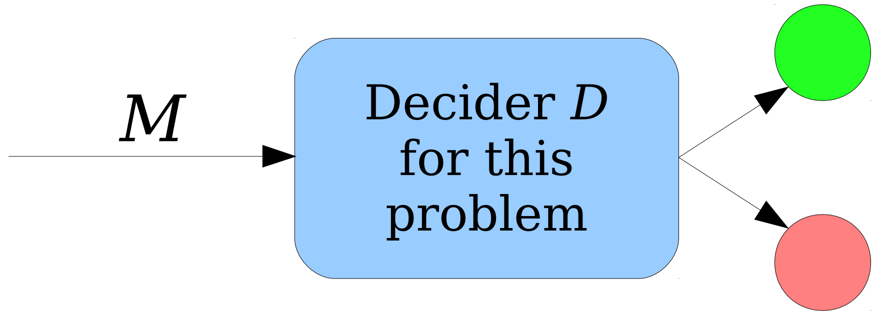


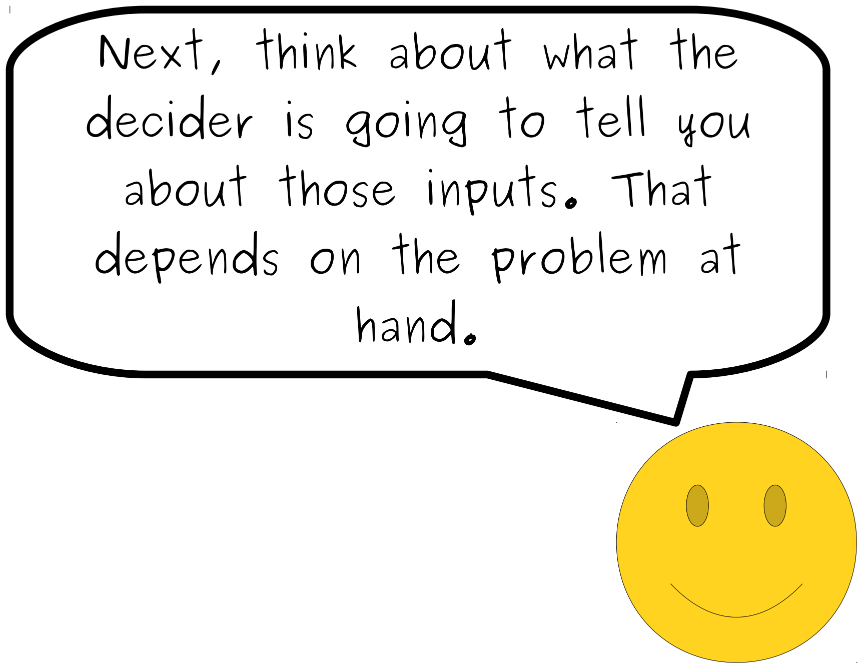


Contradiction!

The problem in question is decidable

There is a decider D for that problem.

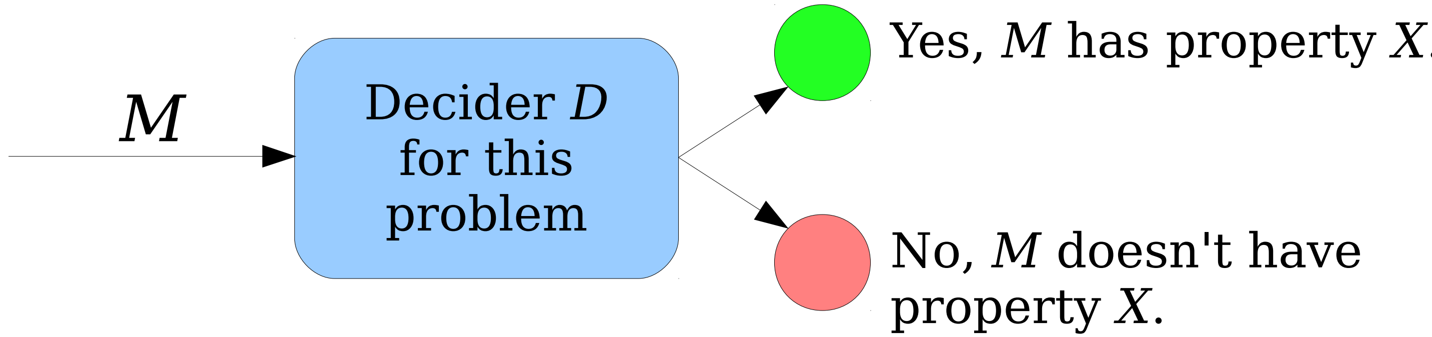


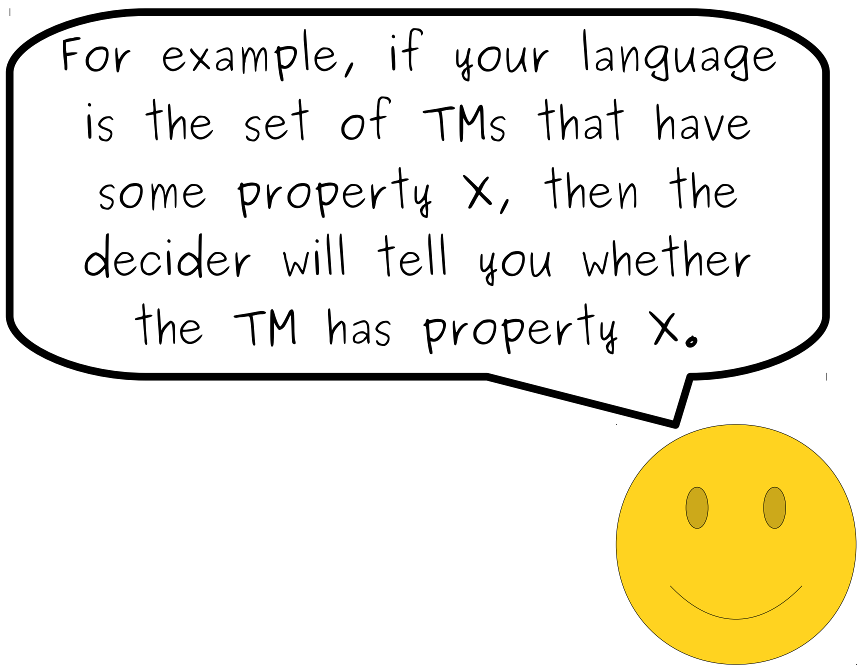


Contradiction!

The problem in question is decidable

There is a decider D for that problem.





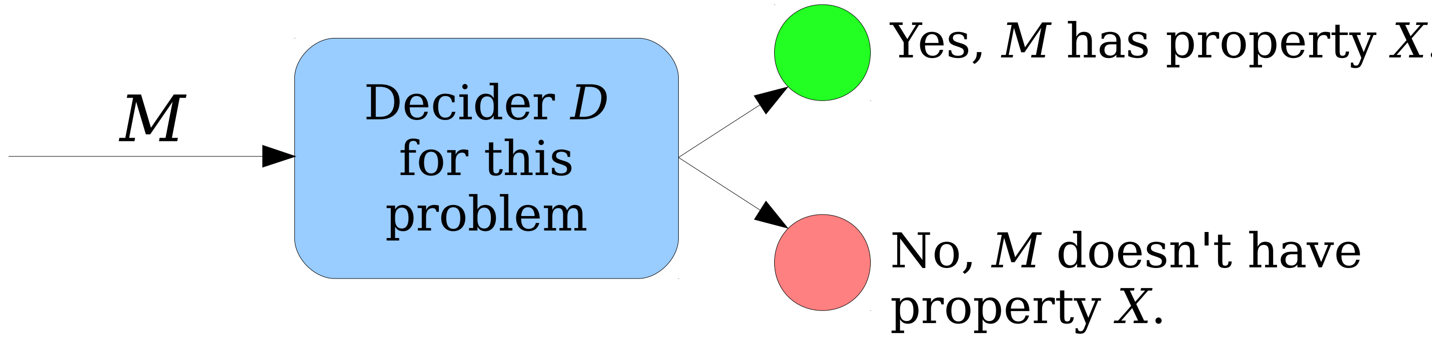
Contradiction!

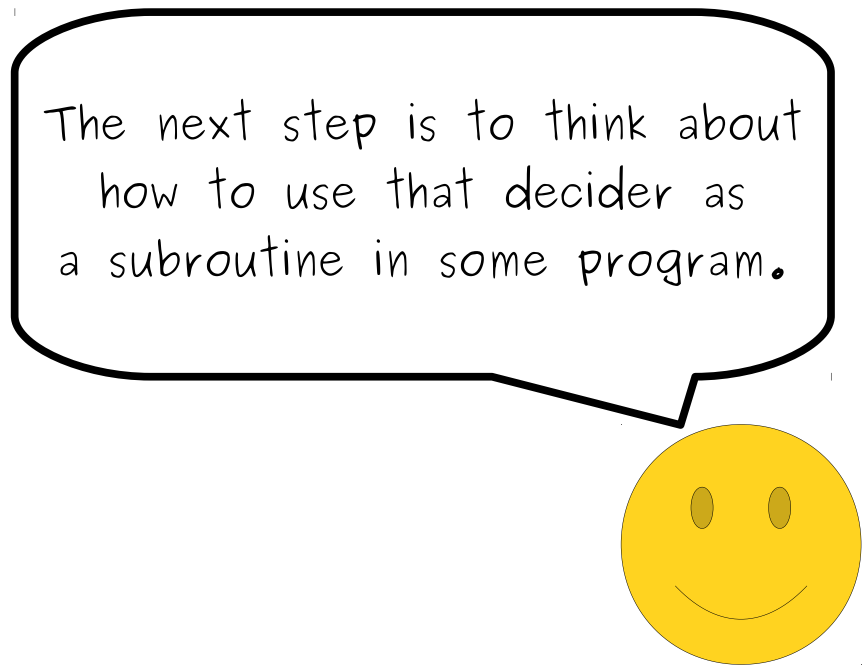
The problem in question is decidable

There is a decider D for that problem.

We can write programs that use D as a helper method

Contradiction!



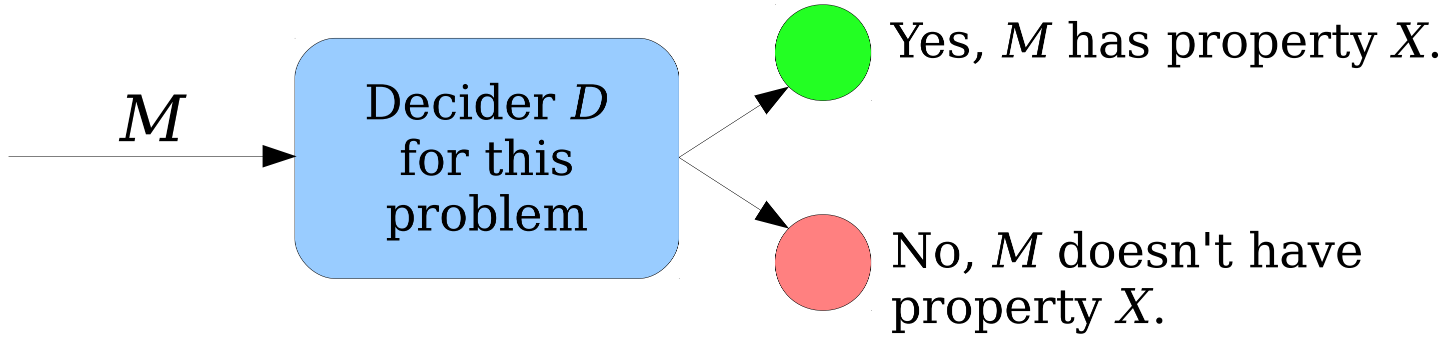


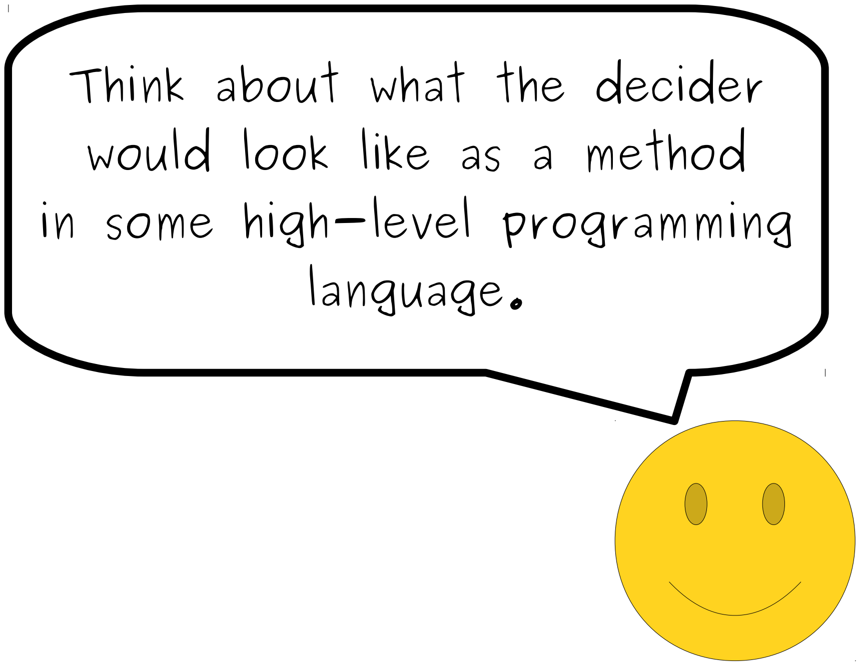
The problem in question is decidable

There is a decider *D* for that problem.

We can write programs that use *D* as a helper method

Contradiction!



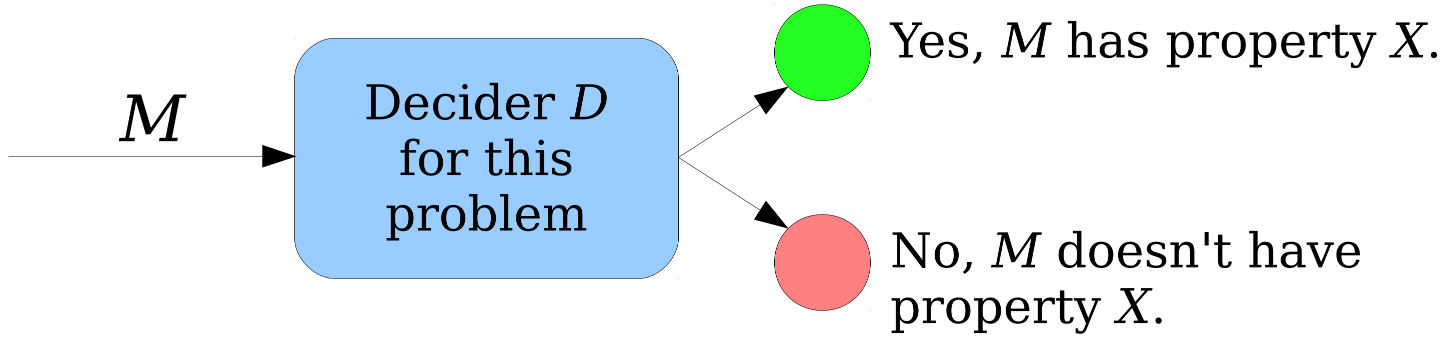


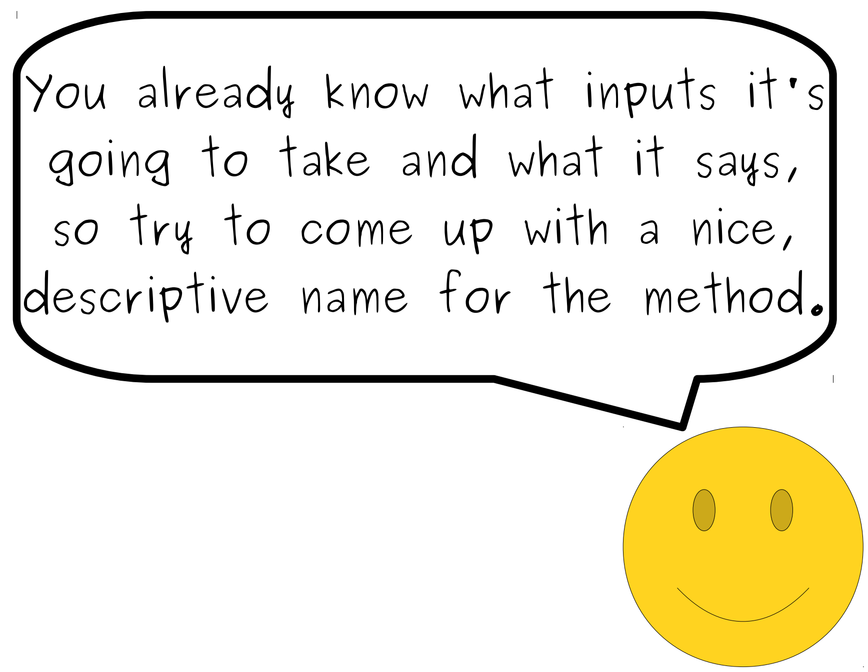
The problem in question is decidable

There is a decider D for that problem.

We can write programs that use D as a helper method

Contradiction!



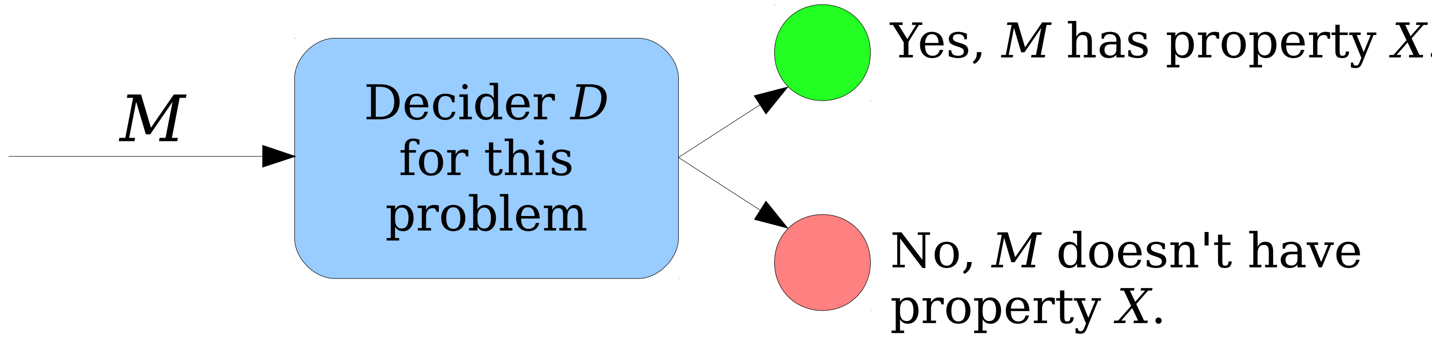


The problem in question is decidable

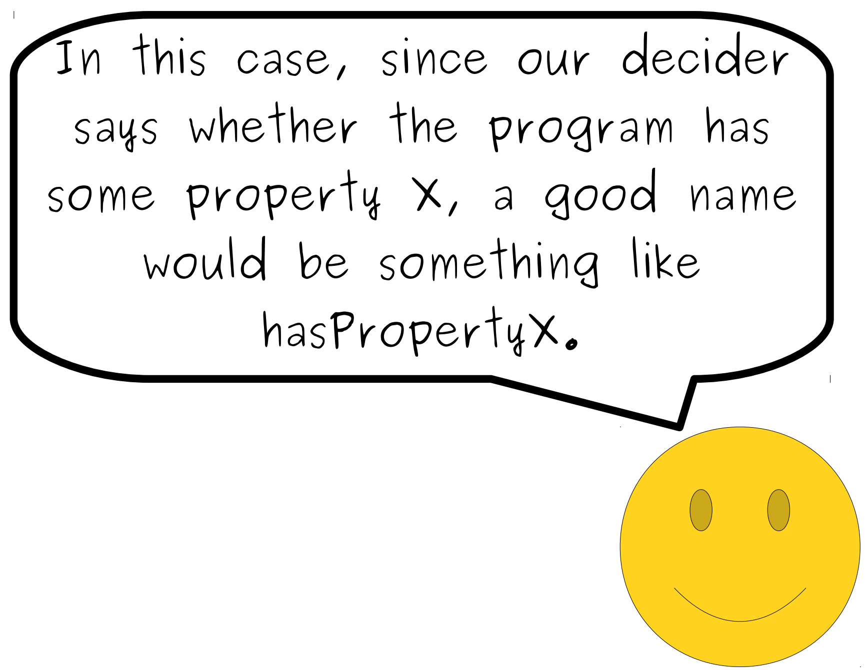
There is a decider D for that problem.

We can write programs that use D as a helper method

Contradiction!




```
bool hasPropertyX(string program)
```



The problem in question is decidable

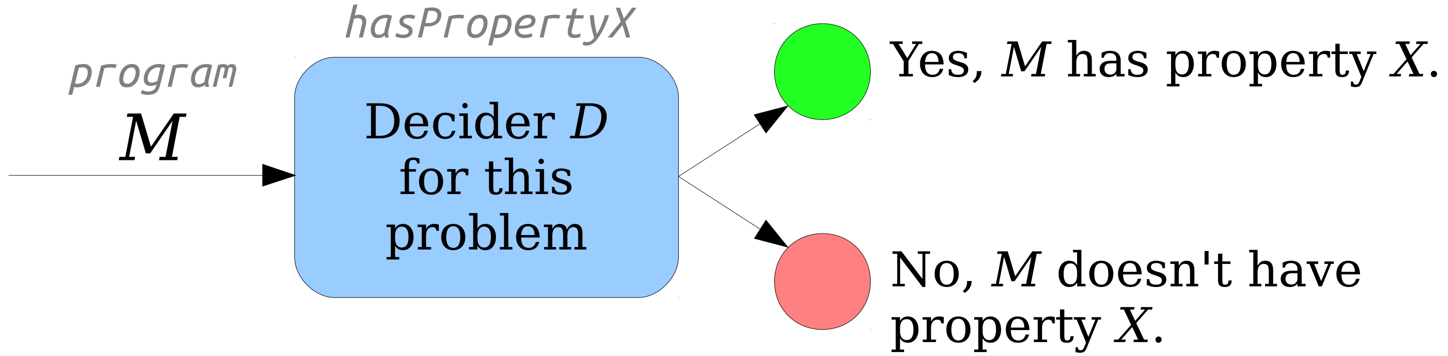
↓

There is a decider D for that problem.

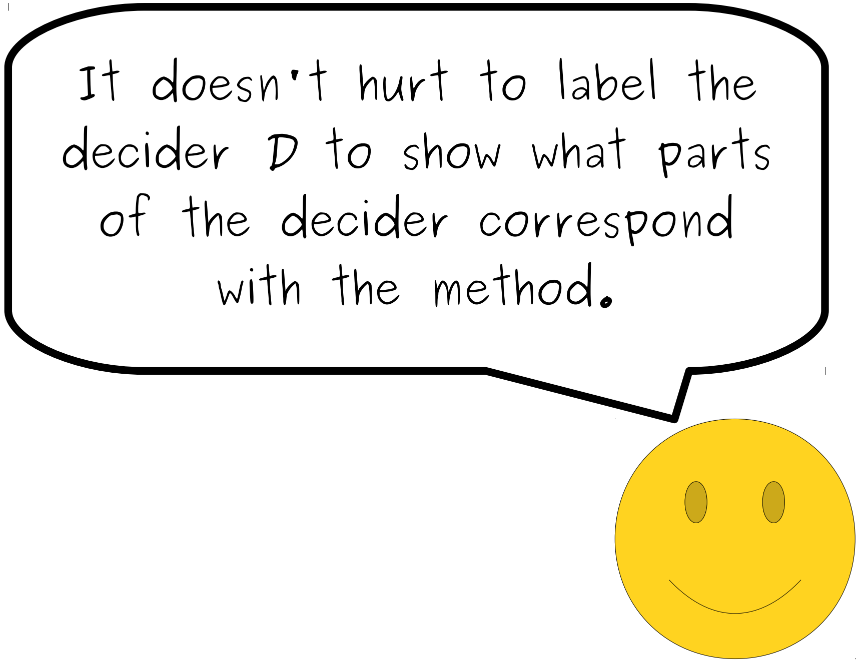
↓

We can write programs that use D as a helper method

Contradiction!




```
bool hasPropertyX(string program)
```

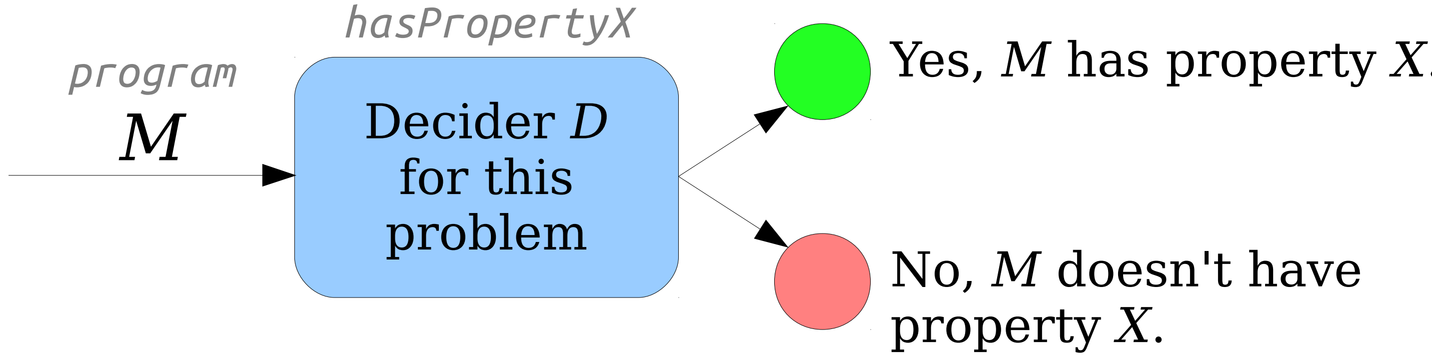


The problem in question is decidable

There is a decider *D* for that problem.

We can write programs that use *D* as a helper method

Contradiction!




```
bool hasPropertyX(string program)
```

The problem in question is decidable

↓

There is a decider *D* for that problem.

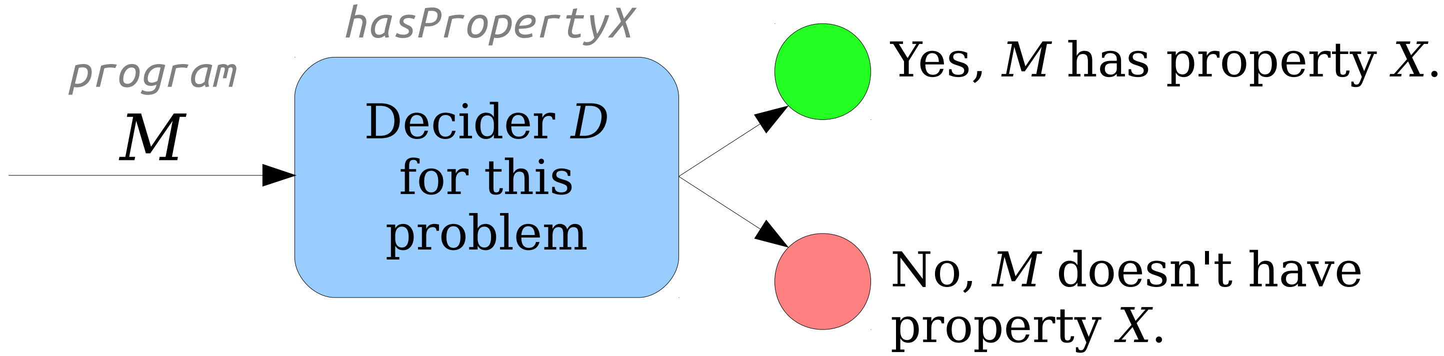
↓

We can write programs that use *D* as a helper method

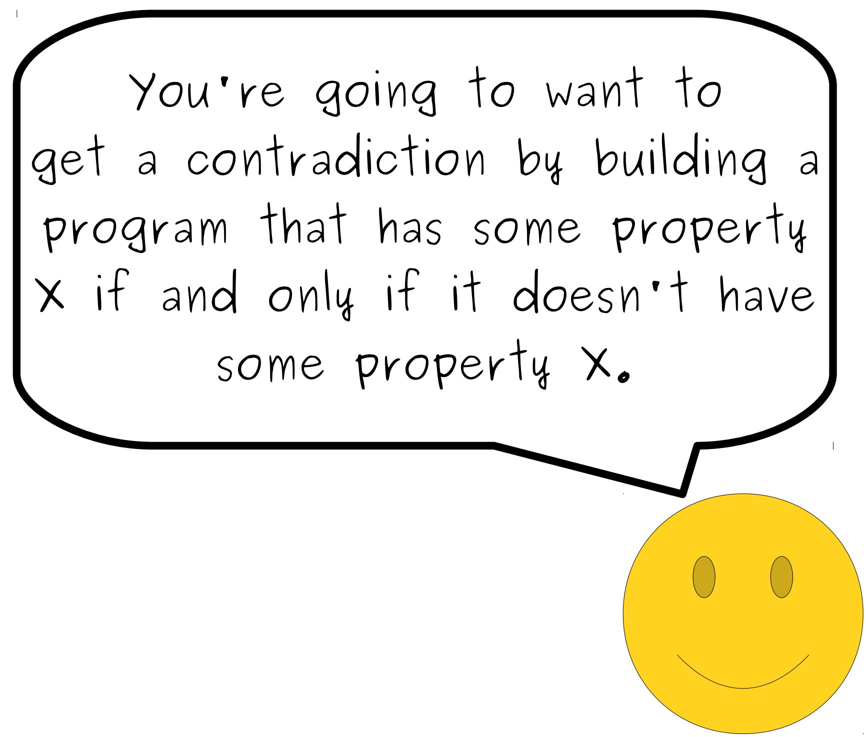
↓

Program *P* has property *X* if and only if *P* doesn't have property *X*

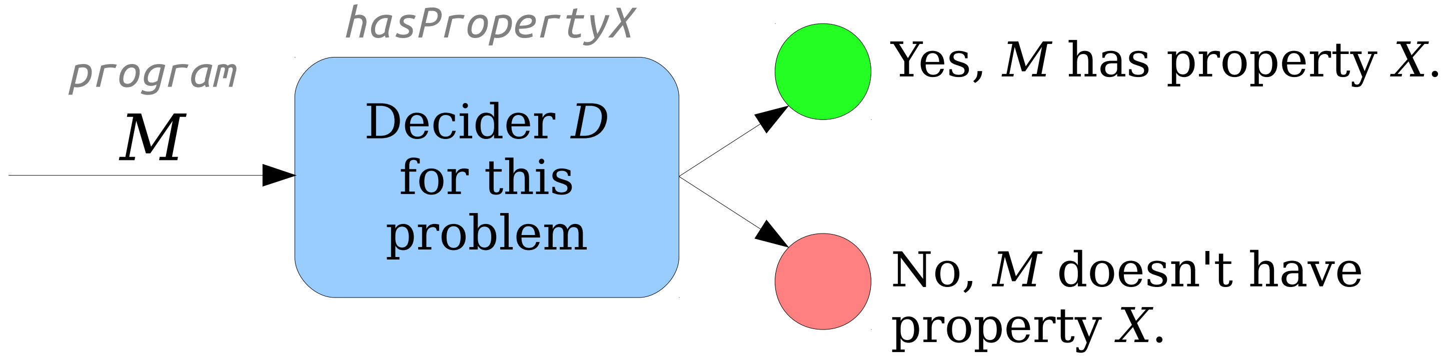
Contradiction!




```
bool hasPropertyX(string program)
```



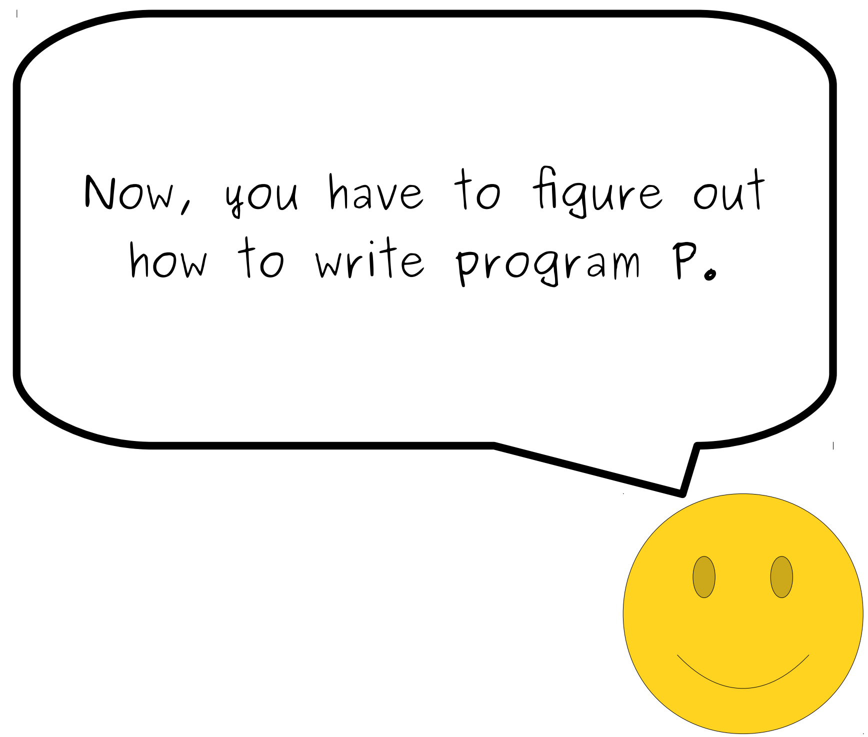
The problem in question is decidable



There is a decider D for that problem.

```
bool hasPropertyX(string program)
```

We can write programs that use D as a helper method



Program P has property X if and only if P doesn't have property X

Contradiction!

The problem in question is decidable

↓

There is a decider *D* for that problem.

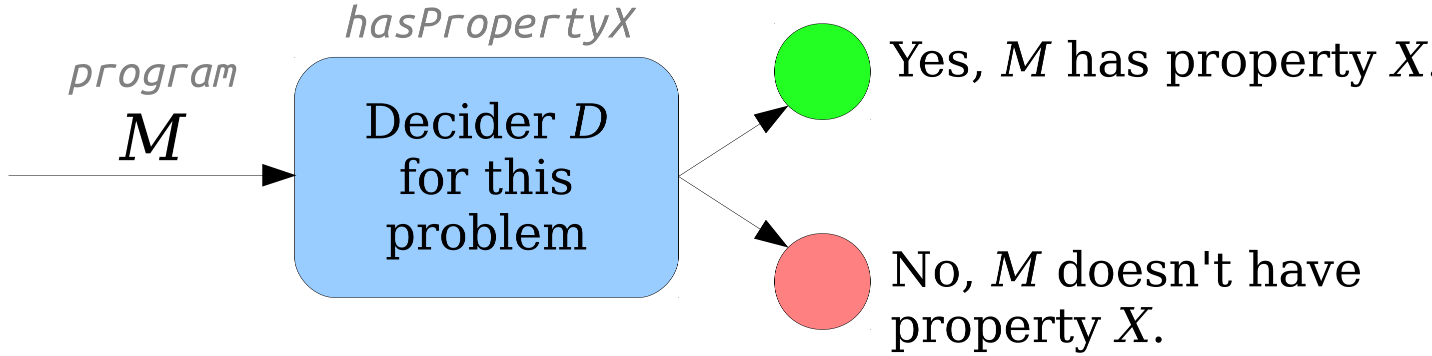
↓

We can write programs that use *D* as a helper method

↓

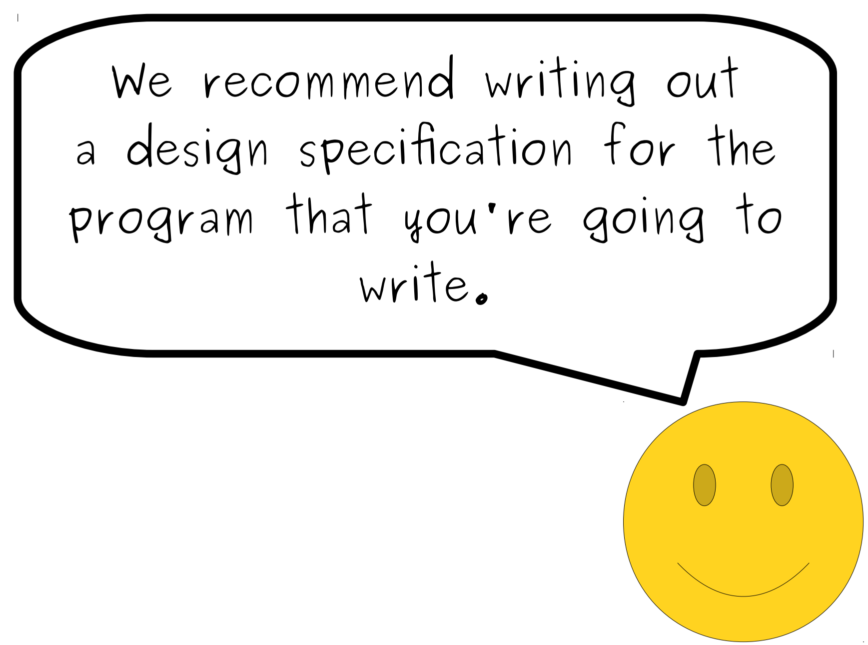
Program *P* has property *X* if and only if *P* doesn't have property *X*

Contradiction!




```
bool hasPropertyX(string program)
```

Program *P* design specification:



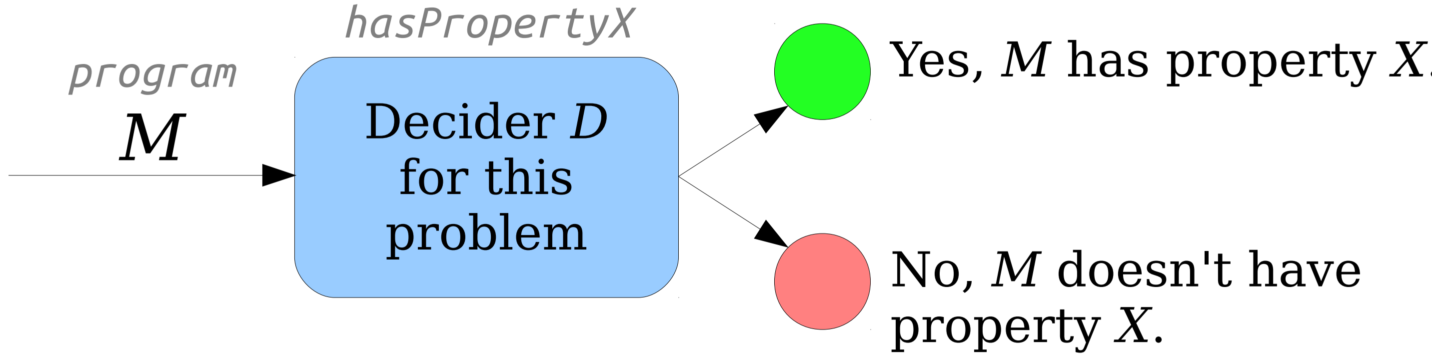
The problem in question is decidable

There is a decider D for that problem.

We can write programs that use D as a helper method

Program P has property X if and only if P doesn't have property X

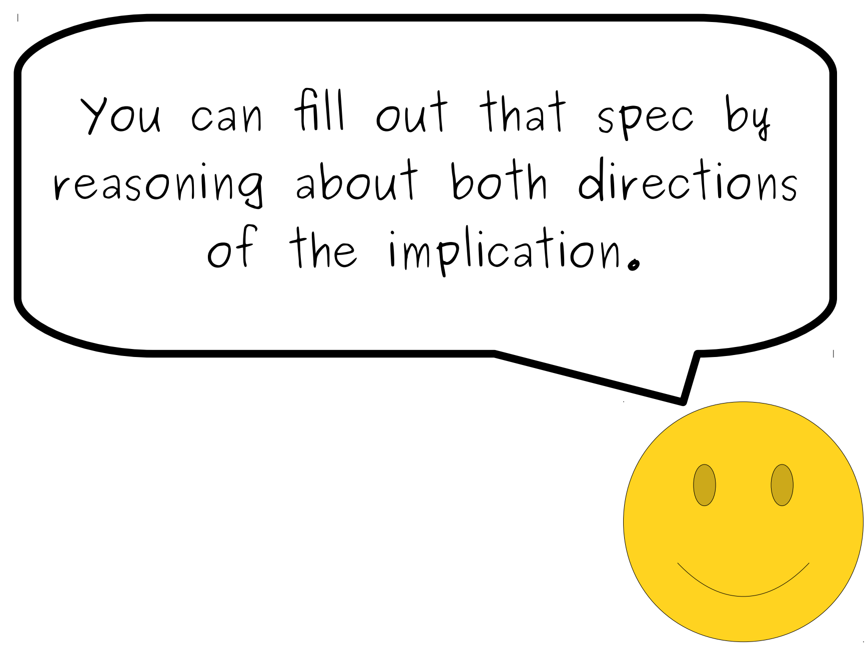
Contradiction!




```
bool hasPropertyX(string program)
```

Program P design specification:

If P has property X, then
P does not have property X.
If P does not have property X, then
P has property X.



The problem in question is decidable

↓

There is a decider *D* for that problem.

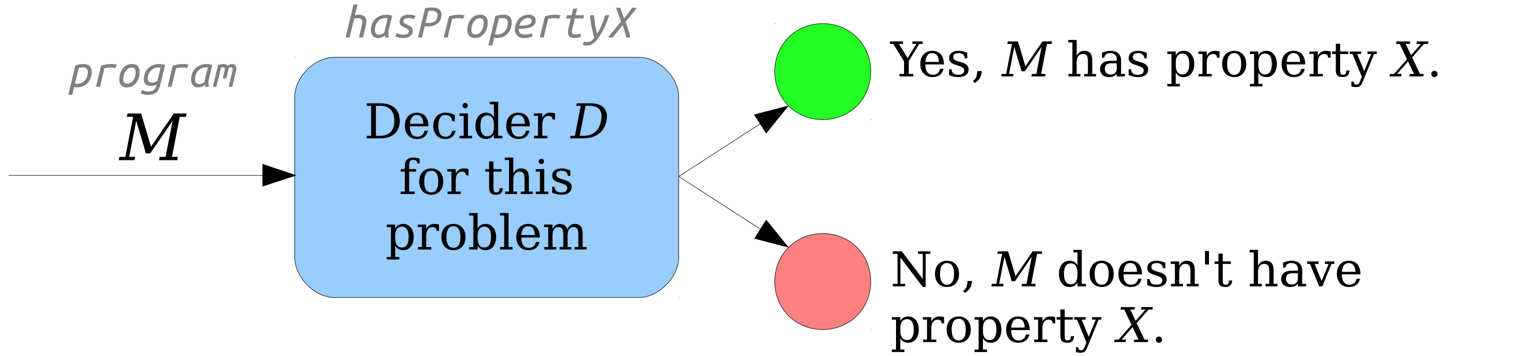
↓

We can write programs that use *D* as a helper method

↓

Program *P* has property *X* if and only if *P* doesn't have property *X*

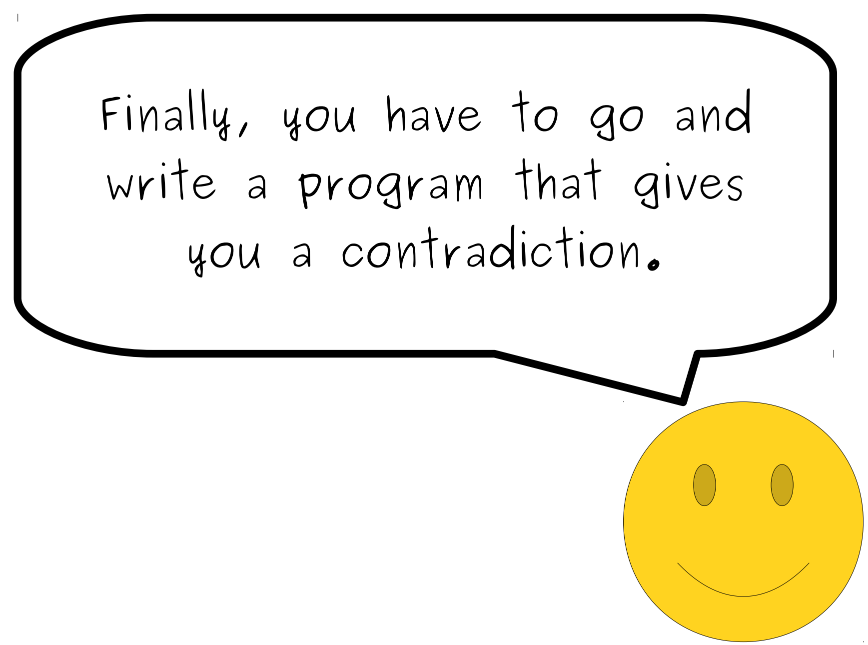
Contradiction!




```
bool hasPropertyX(string program)
```

Program *P* design specification:

If *P* has property *X*, then
P does not have property *X*.
If *P* does not have property *X*, then
P has property *X*.



The problem in question is decidable

↓

There is a decider *D* for that problem.

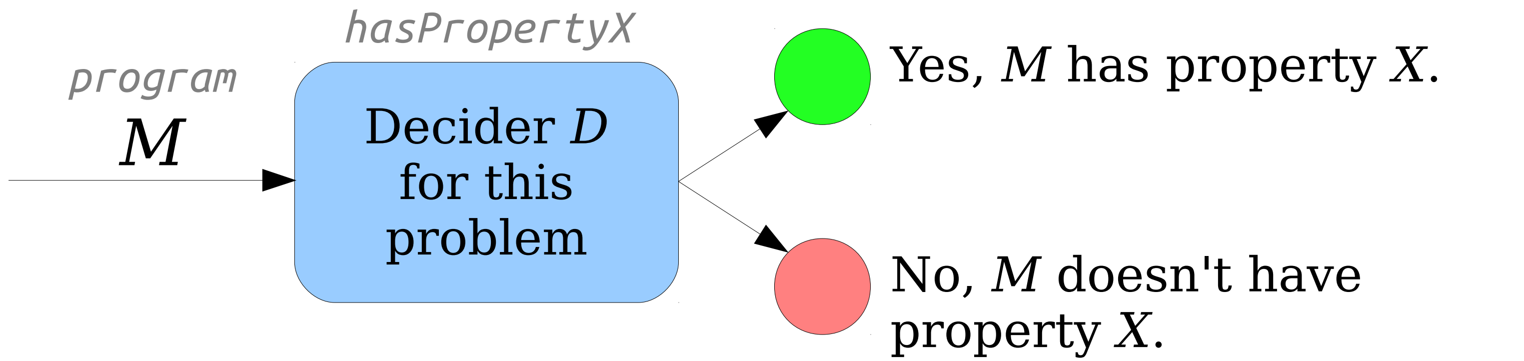
↓

We can write programs that use *D* as a helper method

↓

Program *P* has property *X* if and only if *P* doesn't have property *X*

Contradiction!

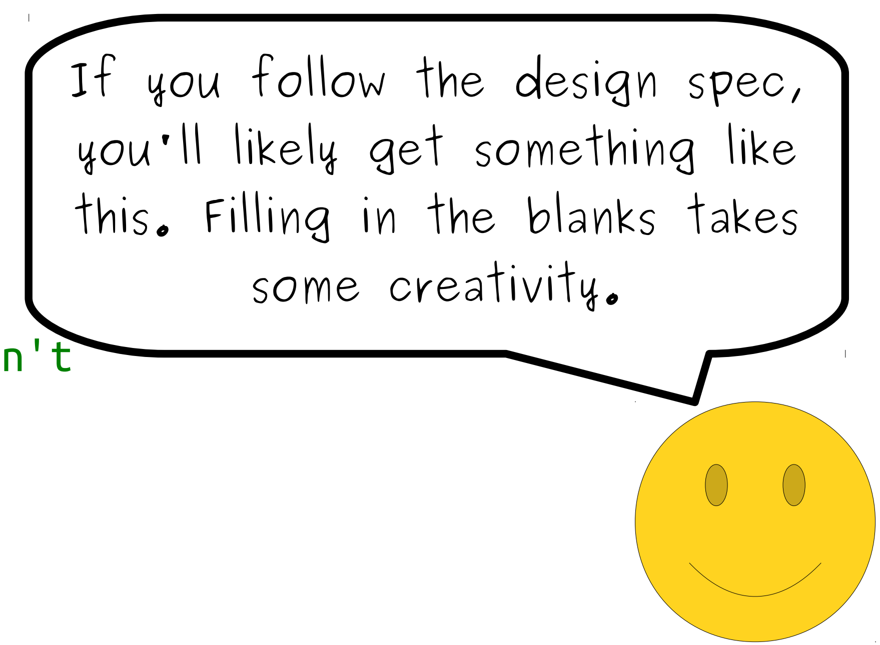



```
bool hasPropertyX(string program)
```

Program *P* design specification:

- If *P* has property *X*, then
  - *P* does not have property *X*.
- If *P* does not have property *X*, then
  - *P* has property *X*.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (hasPropertyX(me)) {
    // do something so you don't
    // have property X.
  } else {
    // Do something so you do
    // have property X.
  }
}
```



The problem in question is decidable

↓

There is a decider *D* for that problem.

↓

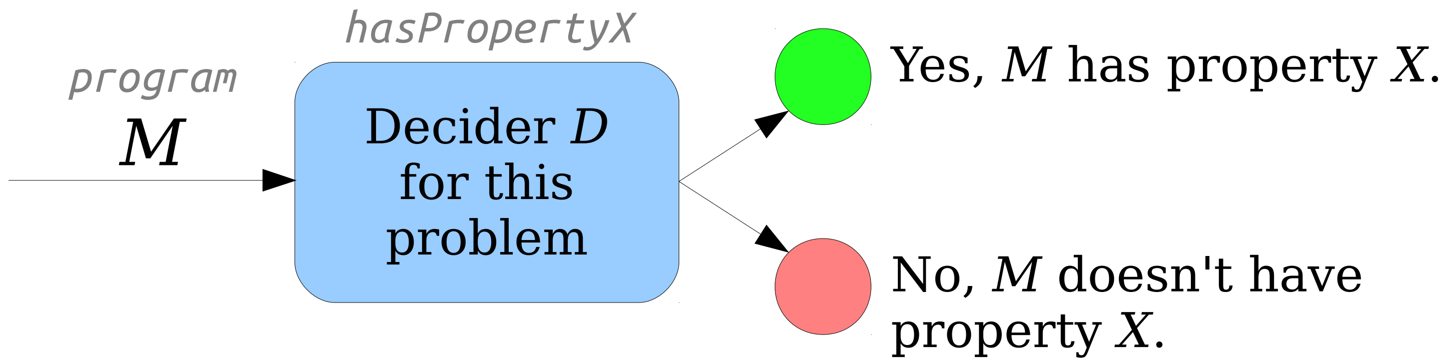
We can write programs that use *D* as a helper method

↓

Program *P* has property *X* if and only if *P* doesn't have property *X*

↓

Contradiction!




```
bool hasPropertyX(string program)
```

Program *P* design specification:

If *P* has property *X*, then
P does not have property *X*.
If *P* does not have property *X*, then
P has property *X*.

```
// Program P

int main() {
  string input = getInput();
  string me = mySource();

  if (hasPropertyX(me)) {
    // do something so you don't
    // have property X.
  } else {
    // Do something so you do
    // have property X.
  }
}
```

Hope this helps!

Please feel free to ask questions if you have them.

Did you find this useful? If so, let us know! We can go and make more guides like these.