Computer-Aided Reasoning for Software

CSE507

Course Introduction

courses.cs.washington.edu/courses/cse507/14au/

Emina Torlak
emina@cs.washington.edu

Today

What is this course about?

Course logistics

Review of basic concepts

about

Tools for building better software, more easily

more reliable, faster,
more energy efficient

Tools for building **better software**, more easily

Tools for building better software, **more easily**

automatic verification, debugging & synthesis

Tools for building better software, more easily

```
class List {
  Node head;

  void reverse() {
    Node near = head;
    Node mid = near.next;
    Node far = mid.next;

    near.next = far;
    while (far != null) {
      mid.next = near;
      near = mid;
      mid = far;
      far = far.next;
    }

    mid.next = near;
    head = mid;
  }
}

class Node {
  Node next; String data;
}
```

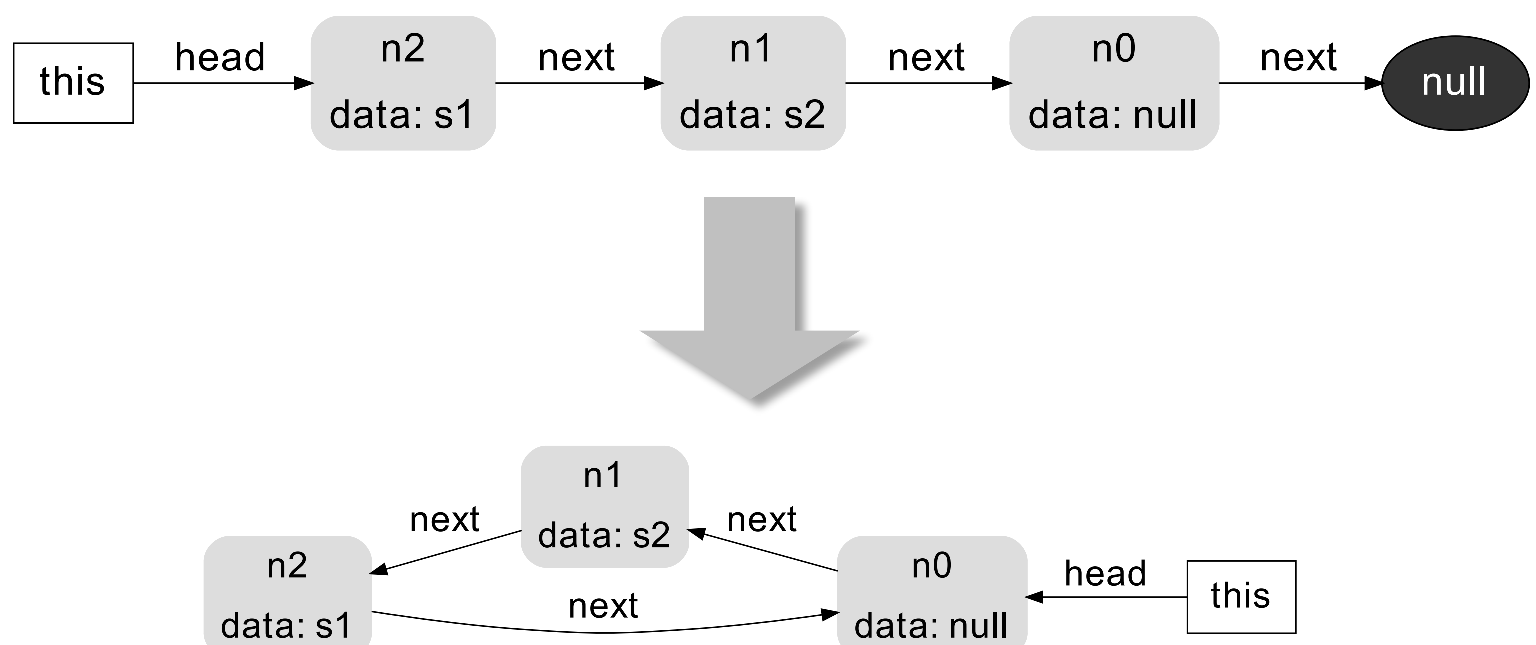
Is this list reversal procedure correct?

Tools for building better software, more easily

```
class List {
  Node head;

  void reverse() {
    Node near = head;
    Node mid = near.next;
    Node far = mid.next;

    near.next = far;
    while (far != null) {
      mid.next = near;
      near = mid;
      mid = far;
      far = far.next;
    }

    mid.next = near;
    head = mid;
  }
}

class Node {
  Node next; String data;
}
```

verification

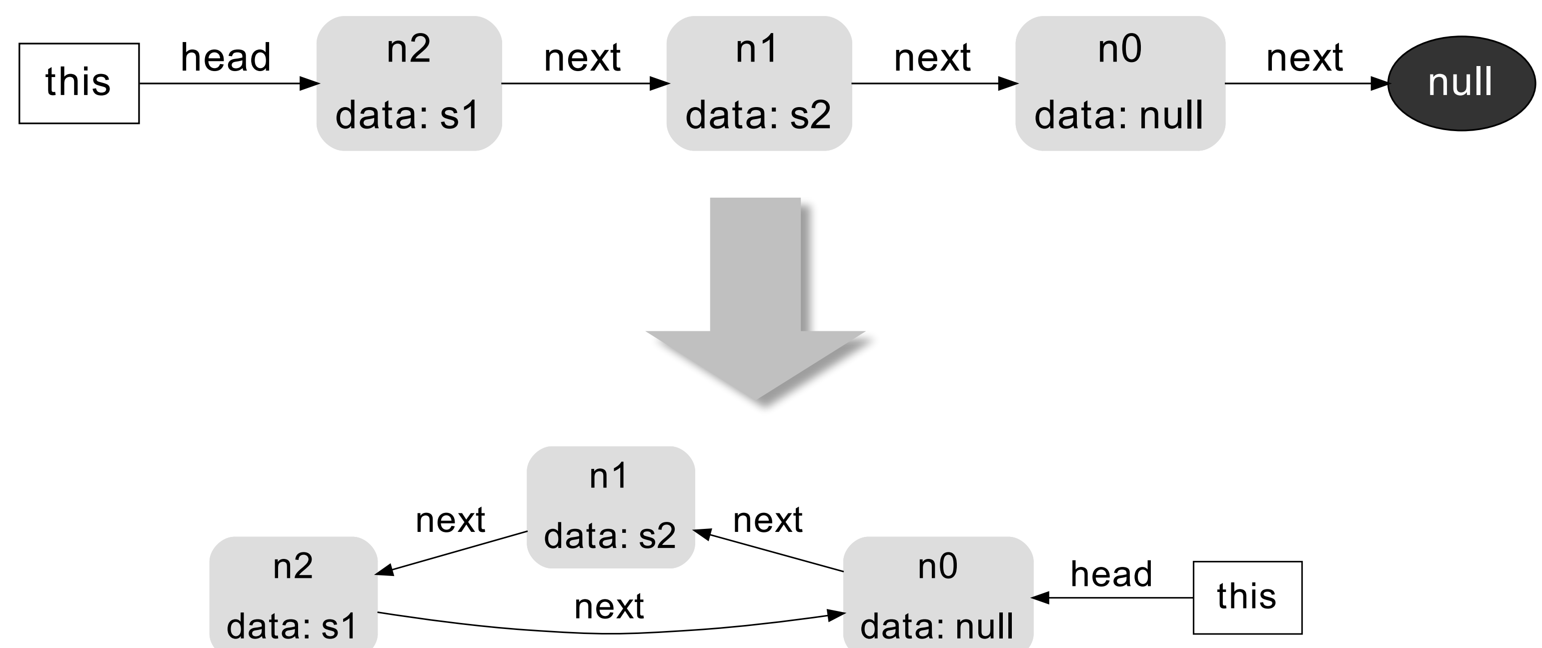
Is this list reversal procedure correct?



Tools for building better software, more easily

```
class List {
  Node head;

  void reverse() {
    Node near = head;
    Node mid = near.next;
    Node far = mid.next;

    near.next = far;
    while (far != null) {
      mid.next = near;
      near = mid;
      mid = far;
      far = far.next;
    }

    mid.next = near;
    head = mid;
  }
}

class Node {
  Node next; String data;
}
```

Which lines of code are responsible for the buggy behavior?

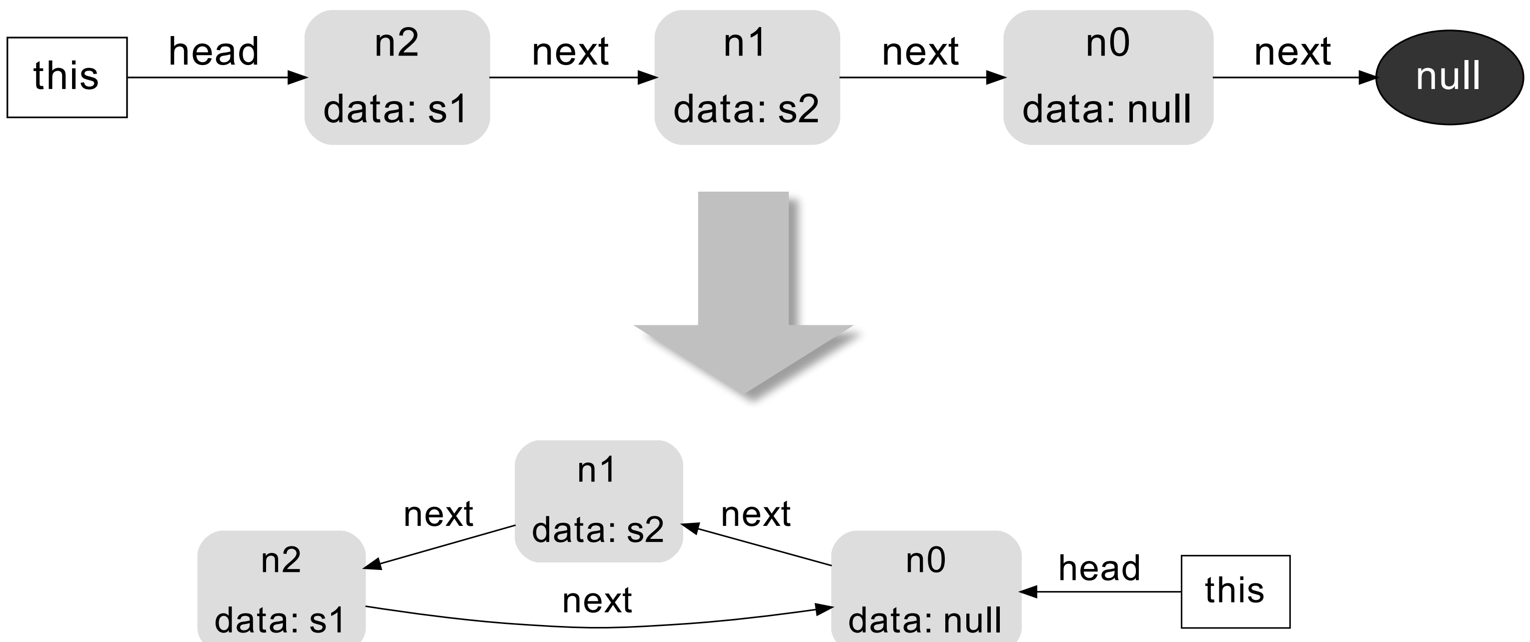


Tools for building better software, more easily

```
class List {
  Node head;

  void reverse() {
    Node near = head;
    Node mid = near.next;
    Node far = mid.next;

    near.next = far;
    while (far != null) {
      mid.next = near;
      near = mid;
      mid = far;
      far = far.next;
    }

    mid.next = near;
    head = mid;
  }
}

class Node {
  Node next; String data;
}
```

debugging

Which lines of code are responsible for the buggy behavior?



Tools for building better software, more easily

```
class List {
  Node head;

  void reverse() {
    Node near = head;
    Node mid = near.next;
    Node far = mid.next;

    near.next = ??;
    while (far != null) {
      mid.next = near;
      near = mid;
      mid = far;
      far = far.next;
    }

    mid.next = near;
    head = mid;
  }
}

class Node {
  Node next; String data;
}
```

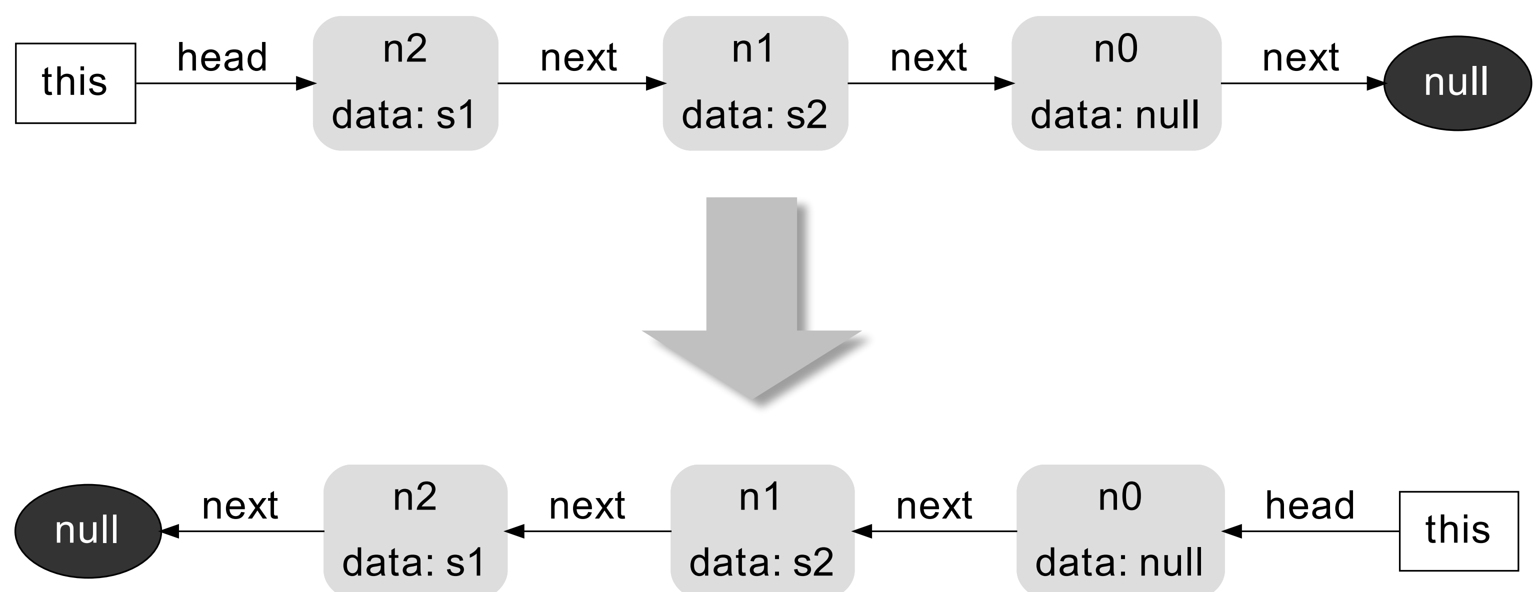
Is there a way to complete this code so that it is correct?

Tools for building better software, more easily

```
class List {
  Node head;

  void reverse() {
    Node near = head;
    Node mid = near.next;
    Node far = mid.next;

    near.next = null;
    while (far != null) {
      mid.next = near;
      near = mid;
      mid = far;
      far = far.next;
    }

    mid.next = near;
    head = mid;
  }
}

class Node {
  Node next; String data;
}
```

synthesis

Is there a way to complete this code so that it is correct?



goal

By the end of this course, you'll be able to build computer-aided tools for any domain!

systems

security

biology

education

goal

By the end of this course, you'll be able to build computer-aided tools for any domain!

networking

hardware

databases

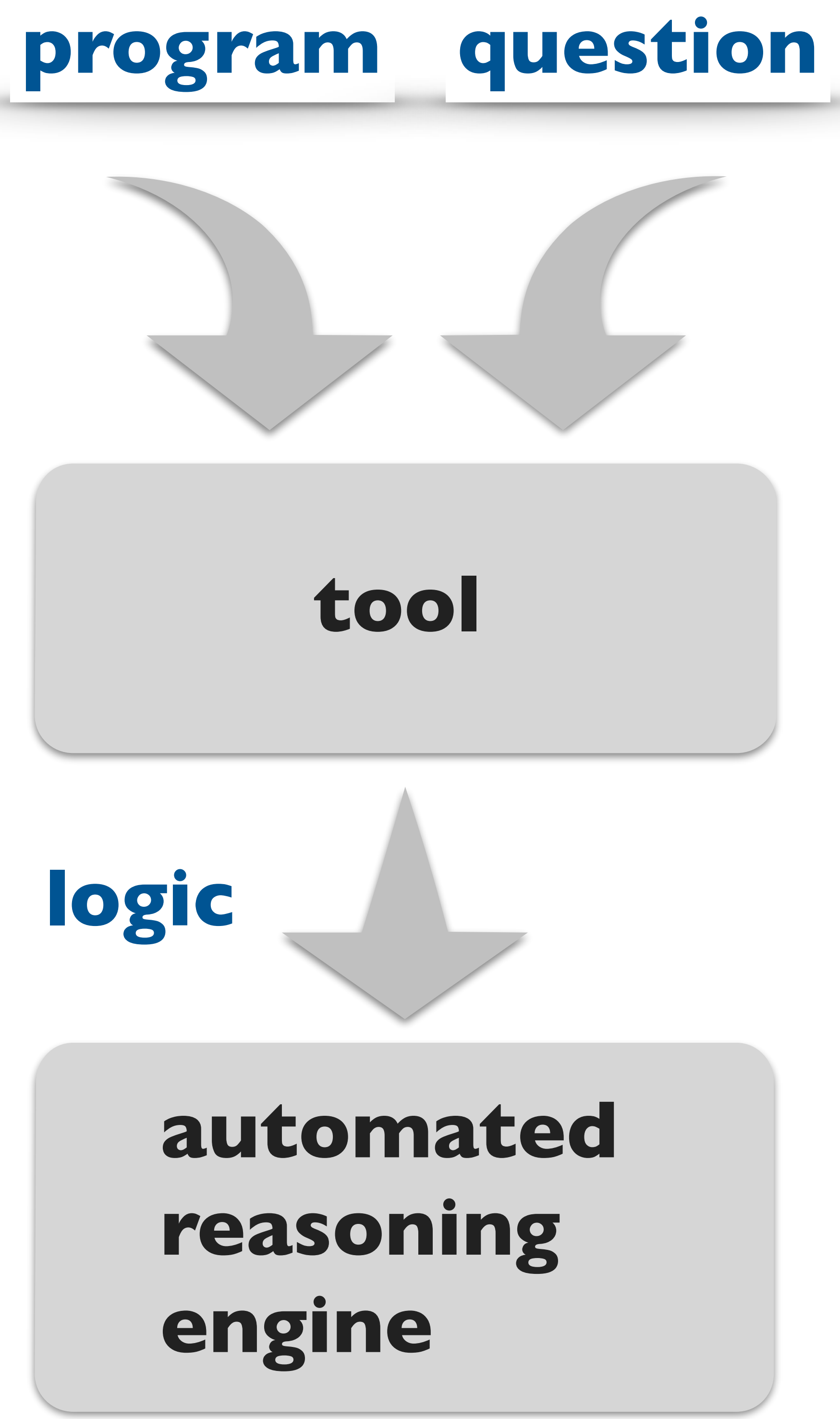
low-power computing

high-performance computing

logistics

Topics, structure, people

Course overview



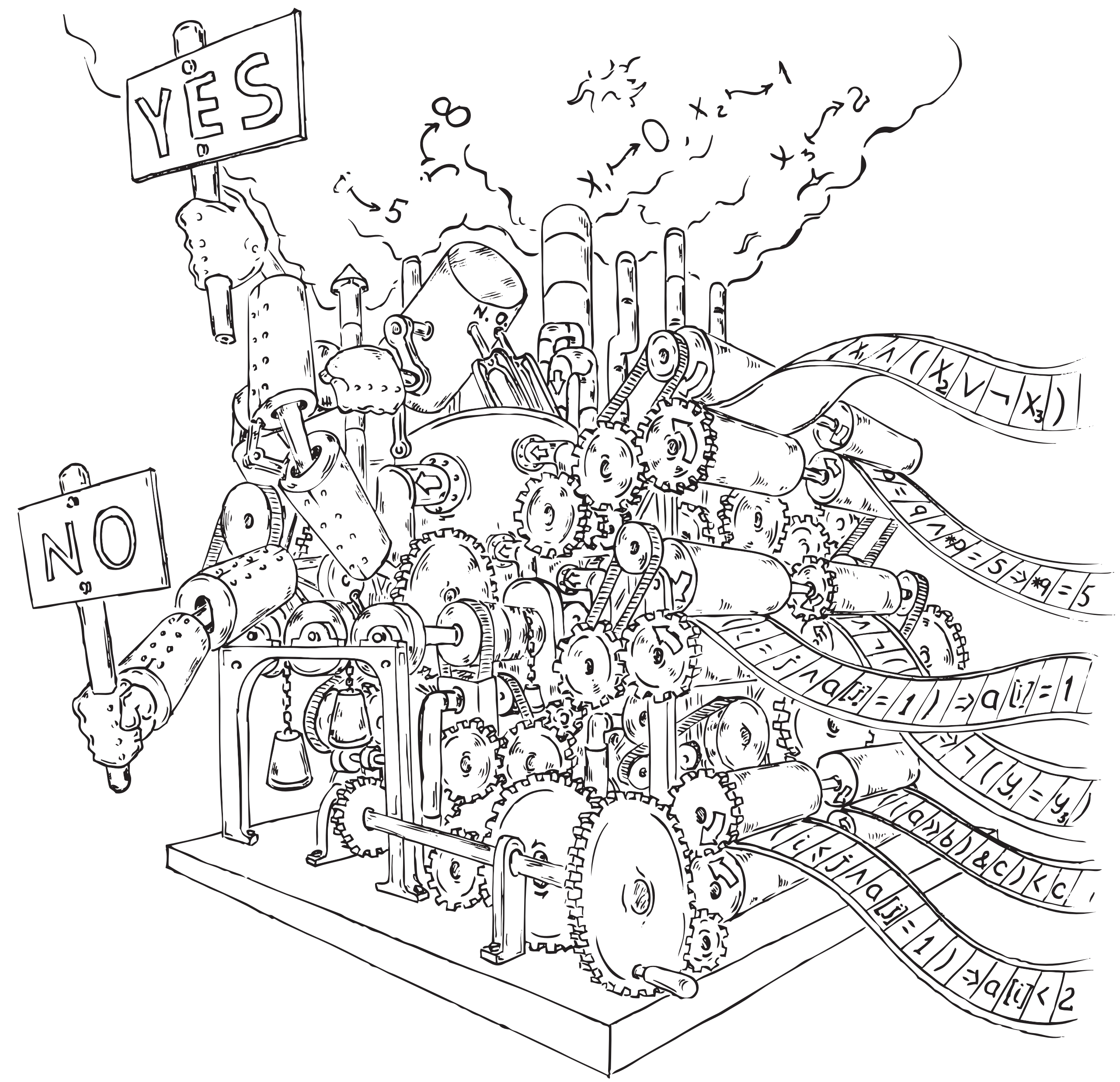
Course overview

program **question**

verifier, synthesizer, fault localizer

logic

SAT, SMT, model finders & checkers



Drawing from “Decision Procedures” by Kroening & Strichman

Course overview

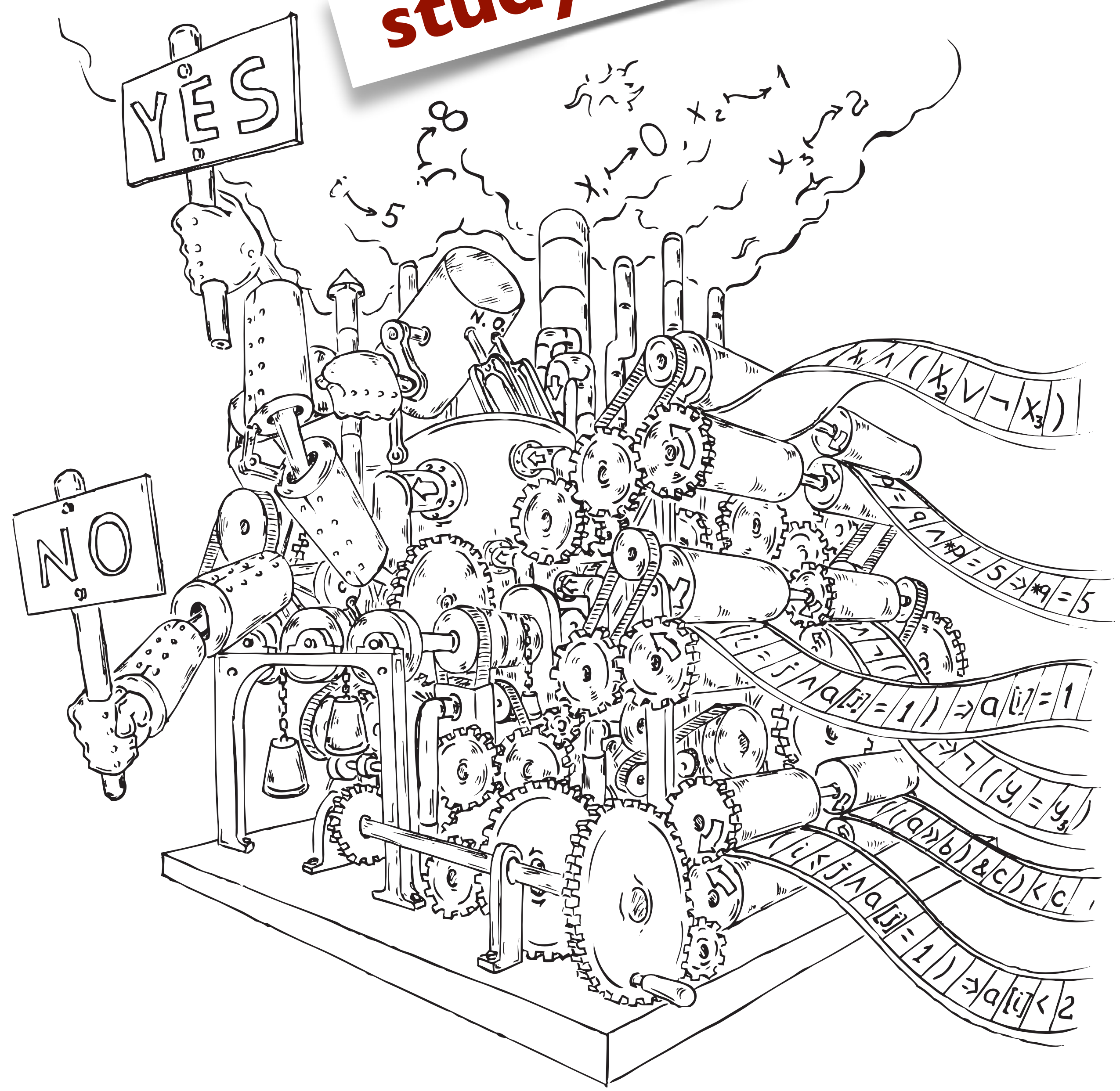
program **question**

verifier, synthesizer, fault localizer

logic

SAT, SMT, model finders & checkers

study (part I)



Drawing from "Decision Procedures" by Kroening & Strichman

Course overview

program **question**

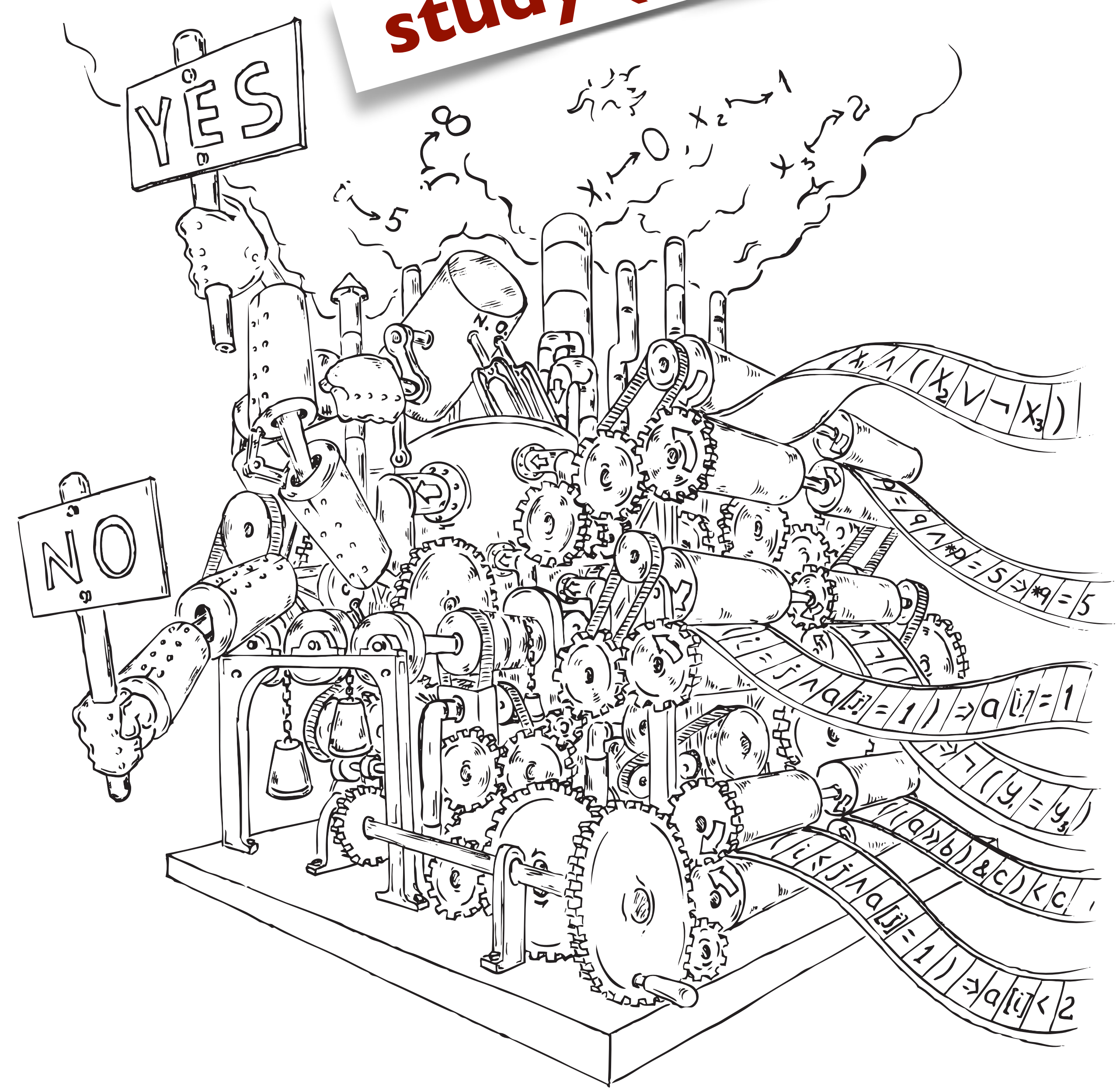
build! (part II)

verifier, synthesizer, fault localizer

logic

SAT, SMT, model finders & checkers

study (part I)



Drawing from “Decision Procedures” by Kroening & Strichman

Grading structure

study (part I)

3 individual homework assignments (50%)

- conceptual problems & proofs (Tex)
- implementations in various programming languages
- may discuss problems with others but solutions must be your own

Course project (50%)

- build a computer-aided reasoning tool for a domain of your choice
- teams of 2-3 people strongly encouraged
- see the course web page for timeline, deliverables and other details

build!
(part II)

Reading and references

Required readings posted on the course web page

- Complete each reading before the lecture for which it is assigned

Recommended text books

- Bradley & Manna, The Calculus of Computation
- Kroening & Strichman, Decision Procedures

Related courses

- Isil Dillig: Automated Logical Reasoning (2013)
- Viktor Kuncak: Synthesis, Analysis, and Verification (2013)
- Sanjit Seshia: Computer-Aided Verification (2012)

Advice for doing well in 507

Come to class (prepared)

- Lecture notes are enough to teach from, but not enough to learn from

Participate

- Ask and answer questions

Meet deadlines

- Turn homework in on time
- Start homework and project sooner than you think you need to
- Follow instructions for submitting code (we have to be able to run it)

People

Emina Torlak
PLSE
CSE 596
Wednesdays 1-2

Mert Saglam
Theory
CSE 618
Thursdays 1-2

People

Emina Torlak
PLSE
CSE 596
Wednesdays 1-2

Mert Saglam
Theory
CSE 618
Thursdays 1-2

students!

Your name
Research area
Survey

review

Propositional logic: syntax, semantics & proof methods

Syntax of propositional logic

$$(\neg p \wedge \top) \vee (q \to \bot)$$

Syntax of propositional logic

$$(\neg \boldsymbol{p} \wedge \top) \vee (\boldsymbol{q} \rightarrow \bot)$$

Atom truth symbols: $\top$ ("true"), $\bot$ ("false")
propositional variables: $p, q, r, \ldots$

Syntax of propositional logic

$$(\neg p \wedge \top) \vee (q \rightarrow \bot)$$

Atom — **truth symbols**: $\top$ (“true”), $\bot$ (“false”)
propositional variables: $p, q, r, \ldots$

Literal — an atom α or its negation $\neg\alpha$

Syntax of propositional logic

$(\neg p \wedge \top) \vee (q \rightarrow \bot)$

Atom	**truth symbols**: $\top$ ("true"), $\bot$ ("false") **propositional variables**: $p, q, r, \ldots$
Literal	an atom α or its negation $\neg\alpha$
Formula	a literal or the application of a **logical connective** to formulas

$\neg F$	"not"	(negation)
$F_1 \wedge F_2$	"and"	(conjunction)
$F_1 \vee F_2$	"or"	(disjunction)
$F_1 \rightarrow F_2$	"implies"	(implication)
$F_1 \longleftrightarrow F_2$	"if and only if"	(iff)

Interpretations of propositional formulas

An **interpretation** *I* for a propositional formula *F* maps every variable in *F* to a truth value:

$$I : \{ p \mapsto \top, q \mapsto \bot, \ldots \}$$

Interpretations of propositional formulas

An **interpretation** I for a propositional formula F maps every variable in F to a truth value:

$$I : \{ p \mapsto \top, q \mapsto \bot, \ldots\}$$

I is a **satisfying interpretation** of F, written as $I \models F$, if F evaluates to $\top$ under I.

I is a **falsifying interpretation** of F, written as $I \not\models F$, if F evaluates to $\bot$ under I.

Semantics of propositional logic

Base cases:

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \top$
- $I \not\models p$ iff $I[p] = \bot$

Semantics of propositional logic

Base cases:

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \top$
- $I \not\models p$ iff $I[p] = \bot$

Inductive cases:

Semantics of propositional logic

Base cases:

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \top$
- $I \not\models p$ iff $I[p] = \bot$

Inductive cases:

- $I \models \neg F$ iff $I \not\models F$

Semantics of propositional logic

Base cases:

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \top$
- $I \not\models p$ iff $I[p] = \bot$

Inductive cases:

- $I \models \neg F$ iff $I \not\models F$
- $I \models F_1 \land F_2$ iff $I \models F_1$ and $I \models F_2$

Semantics of propositional logic

Base cases:

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \top$
- $I \not\models p$ iff $I[p] = \bot$

Inductive cases:

- $I \models \neg F$ iff $I \not\models F$
- $I \models F_1 \wedge F_2$ iff $I \models F_1$ and $I \models F_2$
- $I \models F_1 \vee F_2$ iff $I \models F_1$ or $I \models F_2$
- $I \models F_1 \rightarrow F_2$ iff $I \not\models F_1$ or $I \models F_2$
- $I \models F_1 \longleftrightarrow F_2$ iff $I \models F_1$ and $I \models F_2$, or $I \not\models F_1$ and $I \not\models F_2$

Semantics of propositional logic: example

F: $(p \wedge q) \rightarrow (p \vee \neg q)$

I: $\{p \mapsto \top, q \mapsto \bot\}$

?

Semantics of propositional logic: example

F: $(p \wedge q) \rightarrow (p \vee \neg q)$

I: $\{p \mapsto \top, q \mapsto \bot\}$

$I \models F$ ✓

Satisfiability & validity of propositional formulas

F is **satisfiable** iff $I \models F$ for some I.

F is **valid** iff $I \models F$ for all I.

Satisfiability & validity of propositional formulas

F is **satisfiable** iff $I \models F$ for some I.

F is **valid** iff $I \models F$ for all I.

Duality of satisfiability and validity:

F is valid iff $\neg F$ is unsatisfiable.

Satisfiability & validity of propositional formulas

F is **satisfiable** iff $I \models F$ for some I.

F is **valid** iff $I \models F$ for all I.

Duality of satisfiability and validity:

F is valid iff $\neg F$ is unsatisfiable.

If we have a procedure for checking satisfiability, then we can also check validity of propositional formulas, and vice versa.

Techniques for deciding satisfiability & validity

Search

Deduction

SAT solver

Techniques for deciding satisfiability & validity

Search

Enumerate all interpretations (i.e., build a truth table), and check that they satisfy the formula.

Deduction

SAT solver

Techniques for deciding satisfiability & validity

Search

Enumerate all interpretations (i.e., build a truth table), and check that they satisfy the formula.

Deduction

Assume the formula is invalid, apply proof rules, and check for contradiction in every branch of the proof tree.

SAT solver

Proof by search (truth tables)

F: $(p \wedge q) \rightarrow (p \vee \neg q)$

p	q	$p \wedge q$	$\neg q$	$p \vee \neg q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

Proof by search (truth tables)

$F: \quad (p \wedge q) \rightarrow (p \vee \neg q)$

p	q	$p \wedge q$	$\neg q$	$p \vee \neg q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

Valid.

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \models F_1 \wedge F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}}$$

$$\frac{I \not\models \neg F}{I \models F} \qquad \frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \quad \mid \quad I \not\models F_2}$$

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \models F_1 \land F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}}$$

$$\frac{I \not\models \neg F}{I \models F} \qquad \frac{I \not\models F_1 \land F_2}{I \not\models F_1 \mid I \not\models F_2}$$

F: $p \land \neg q$

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \models F_1 \wedge F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}}$$

$$\frac{I \not\models \neg F}{I \models F} \qquad \frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \quad \mid \quad I \not\models F_2}$$

F: $p \wedge \neg q$

1. $I \not\models p \wedge \neg q$ (assumption)

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \models F_1 \wedge F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}}$$

$$\frac{I \not\models \neg F}{I \models F} \qquad \frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \quad \mid \quad I \not\models F_2}$$

F: $p \wedge \neg q$

1. $I \not\models p \wedge \neg q$ (assumption)
   a. $I \not\models p$ (1, $\wedge$)

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \models F_1 \land F_2}{I \models F_1 \quad I \models F_2}$$

$$\frac{I \not\models \neg F}{I \models F} \qquad \frac{I \not\models F_1 \land F_2}{I \not\models F_1 \mid I \not\models F_2}$$

F: $p \land \neg q$

1. $I \not\models p \land \neg q$ (assumption)
   a. $I \not\models p$ (1, $\land$)
   b. $I \not\models \neg q$ (1, $\land$)

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \models F_1 \wedge F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}}$$

$$\frac{I \not\models \neg F}{I \models F} \qquad \frac{I \not\models F_1 \wedge F_2}{I \not\models F_1 \mid I \not\models F_2}$$

F: $p \wedge \neg q$

1. $I \not\models p \wedge \neg q$	(assumption)
a. $I \not\models p$	($1, \wedge$)
b. $I \not\models \neg q$	($1, \wedge$)
i. $I \models q$	($1b, \neg$)

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \models F_1 \land F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}}$$

$$\frac{I \not\models \neg F}{I \models F} \qquad \frac{I \not\models F_1 \land F_2}{I \not\models F_1 \mid I \not\models F_2}$$

F: $p \land \neg q$

1. $I \not\models p \land \neg q$ (assumption)
   a. $I \not\models p$ $(1, \land)$
   b. $I \not\models \neg q$ $(1, \land)$
      i. $I \models q$ $(1b, \neg)$

Invalid; I is a falsifying interpretation.

Semantic judgements

Formulas F_1 and F_2 are **equivalent**, written $F_1 \iff F_2$, iff $F_1 \longleftrightarrow F_2$ is valid.

Formula F_1 **implies** F_2, written $F_1 \implies F_2$, iff $F_1 \rightarrow F_2$ is valid.

Semantic judgements

Formulas F_1 and F_2 are **equivalent**, written $F_1 \iff F_2$, iff $F_1 \longleftrightarrow F_2$ is valid.

Formula F_1 **implies** F_2, written $F_1 \implies F_2$, iff $F_1 \to F_2$ is valid.

$F_1 \iff F_2$ and $F_1 \implies F_2$ are not propositional formulas (not part of syntax). They are properties of formulas, just like validity or satisfiability.

Semantic judgements

Formulas F_1 and F_2 are **equivalent**, written $F_1 \iff F_2$, iff $F_1 \longleftrightarrow F_2$ is valid.

Formula F_1 **implies** F_2, written $F_1 \implies F_2$, iff $F_1 \rightarrow F_2$ is valid.

If we have a procedure for checking satisfiability, then we can also check for equivalence and implication of propositional formulas.

Summary

Today

- Course overview & logistics
- Review of propositional logic

Next Lecture (by Zach Tatlock)

- Normal forms
- A basic SAT solver

★ Take the **course survey**

★ Read **Chapter 1** of Bradley & Manna