Computer-Aided Reasoning for Software

CSE507

Introduction

courses.cs.washington.edu/courses/cse507/16sp/

Emina Torlak
emina@cs.washington.edu

Today

What is this course about?

Course logistics

Review of propositional logic

A basic SAT solver!

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;
}
```

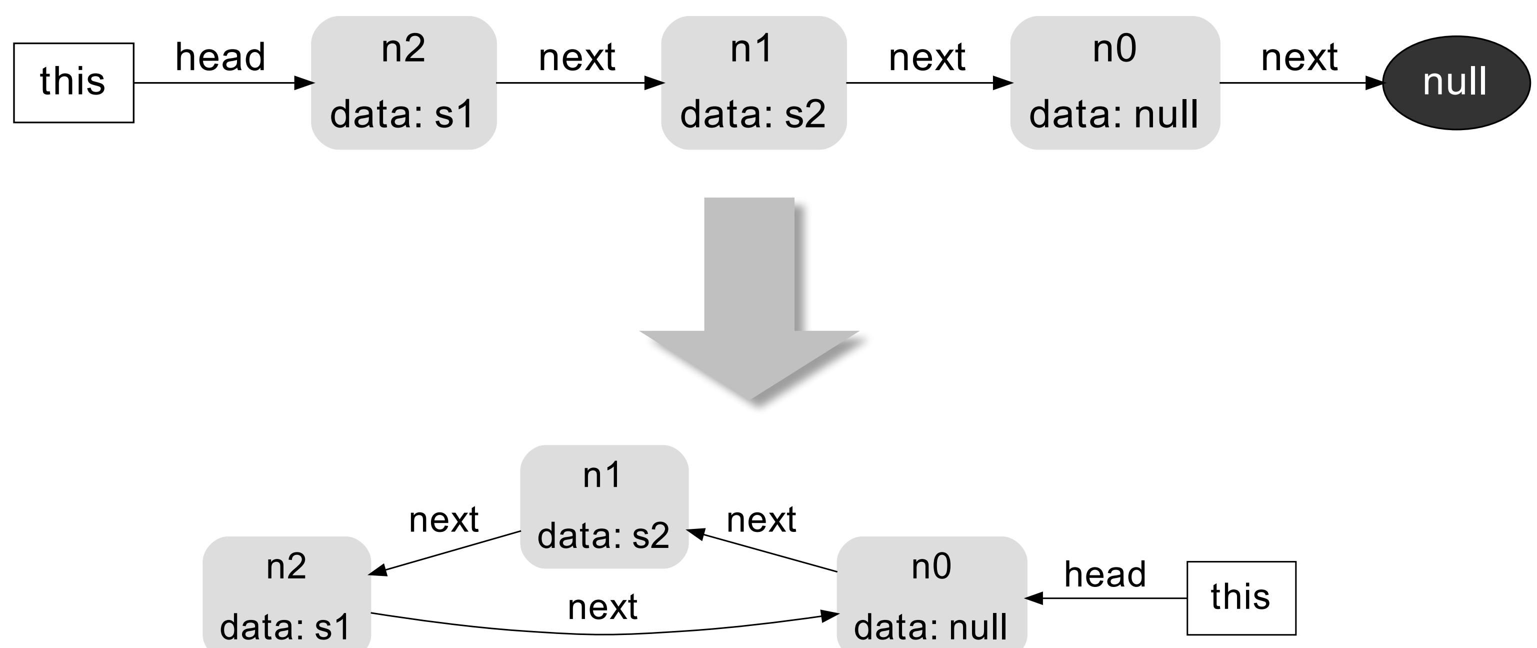
Is this list reversal procedure correct?

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

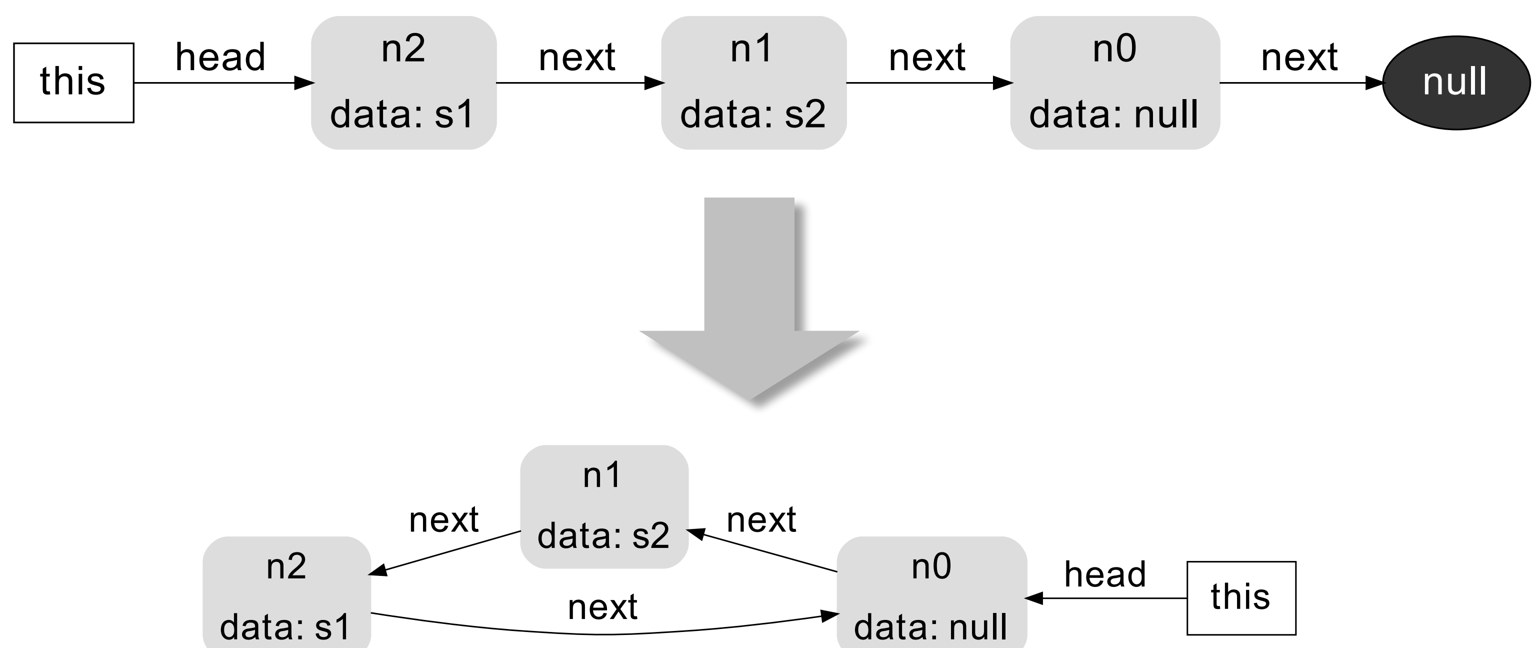
Is this list reversal procedure correct?



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?

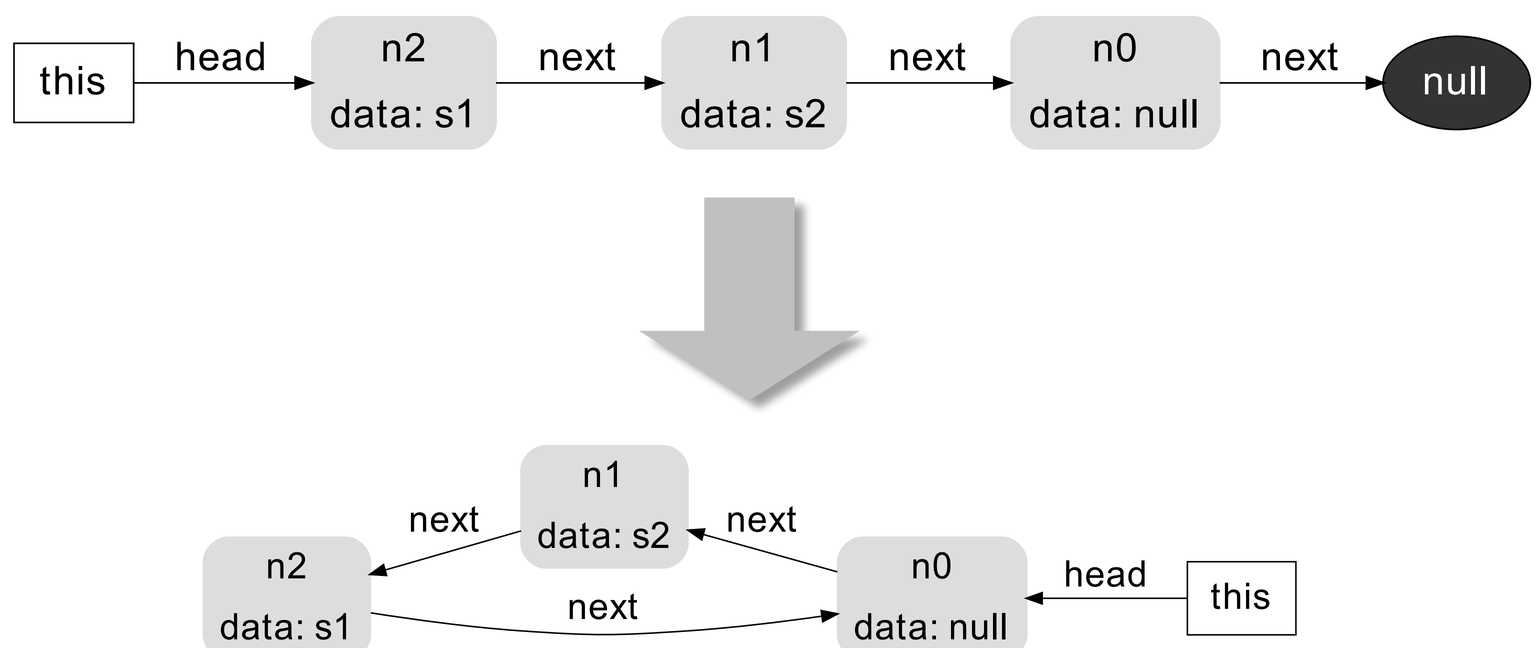


Tools for building better software, more easily

```
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;
}
```

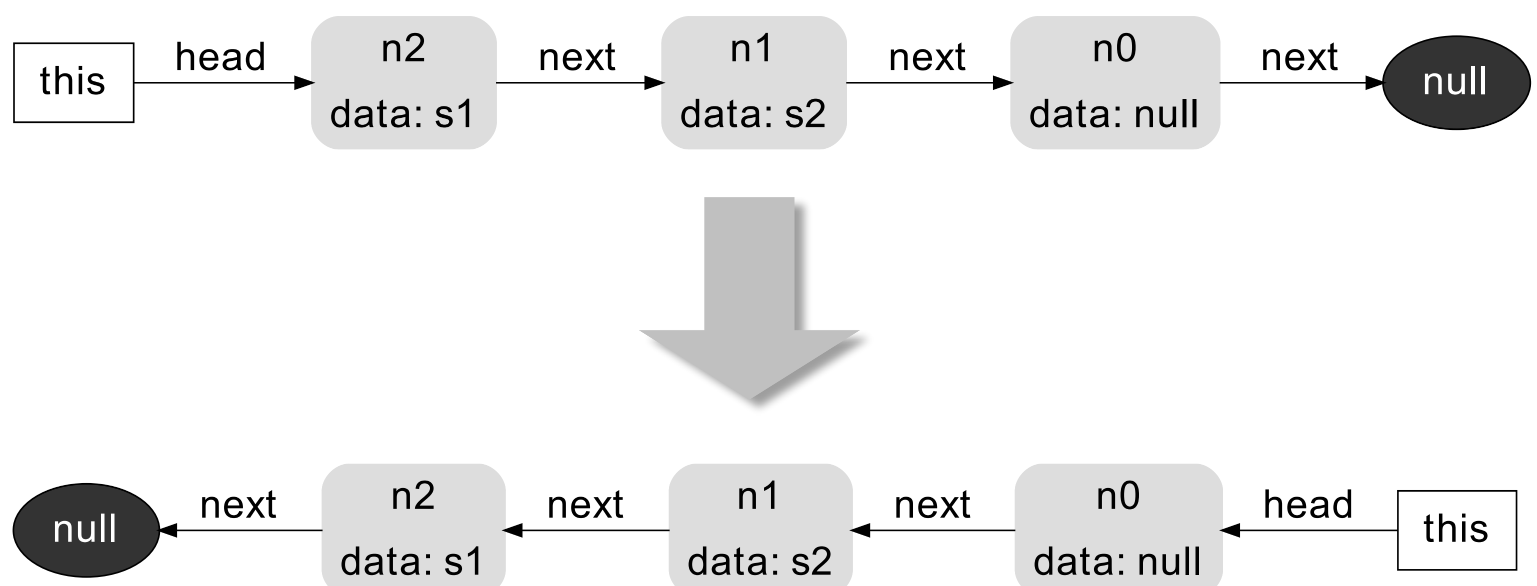
Is there a way to complete this code so that it is correct?

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!

goal

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

systems

security

biology

education

networking

hardware

databases

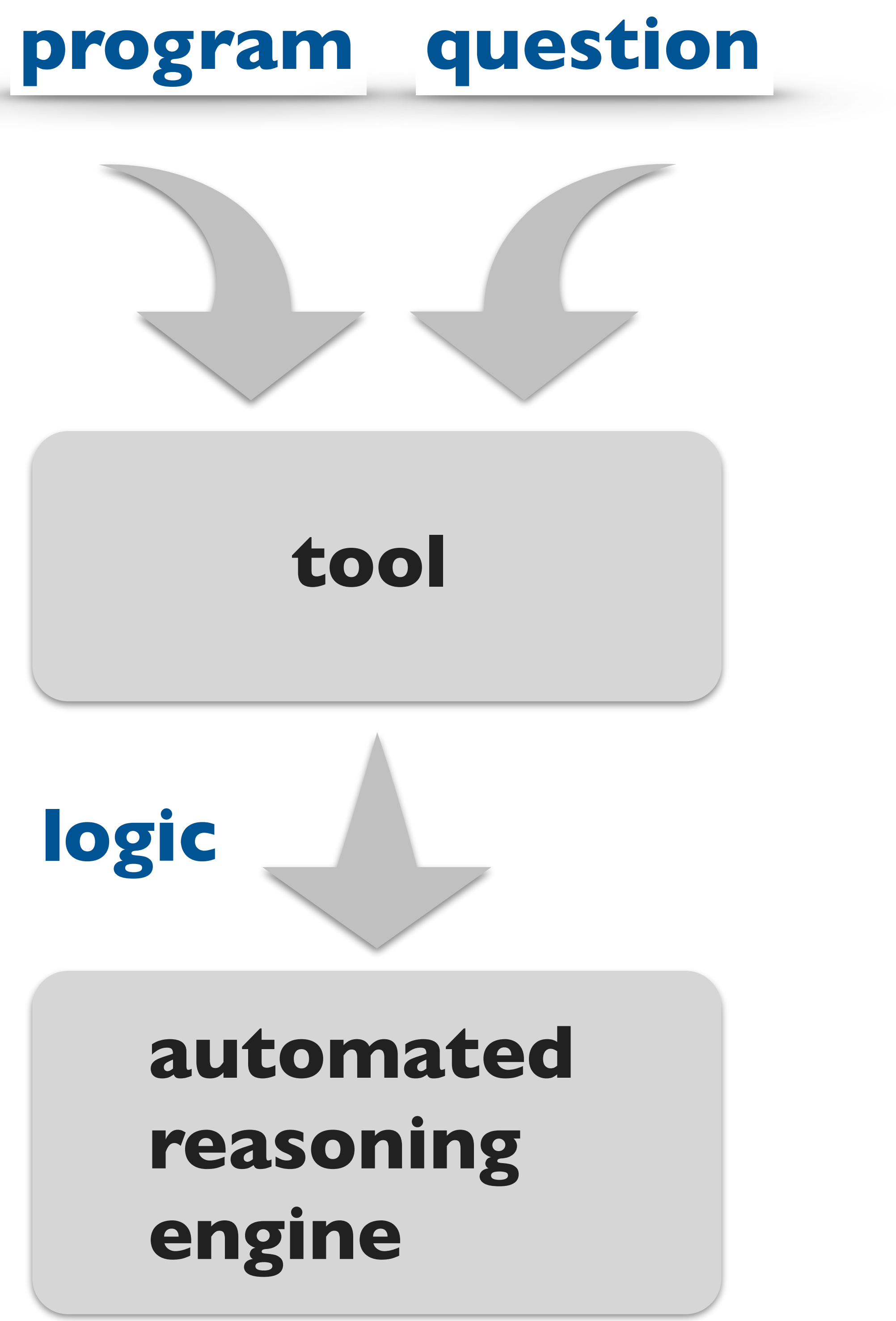
low-power computing

high-performance computing

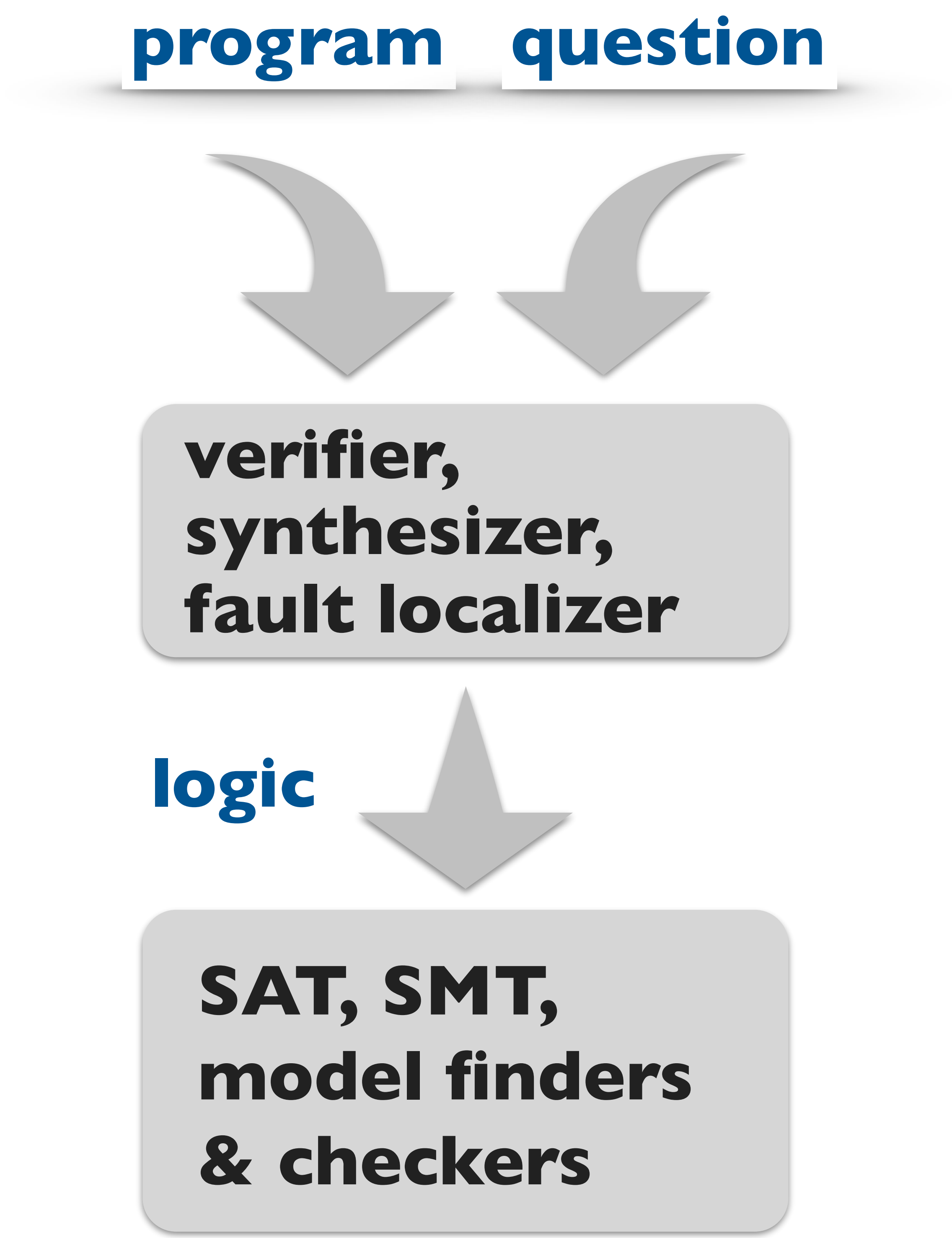
logistics

Topics, structure, people

Course overview



Course overview



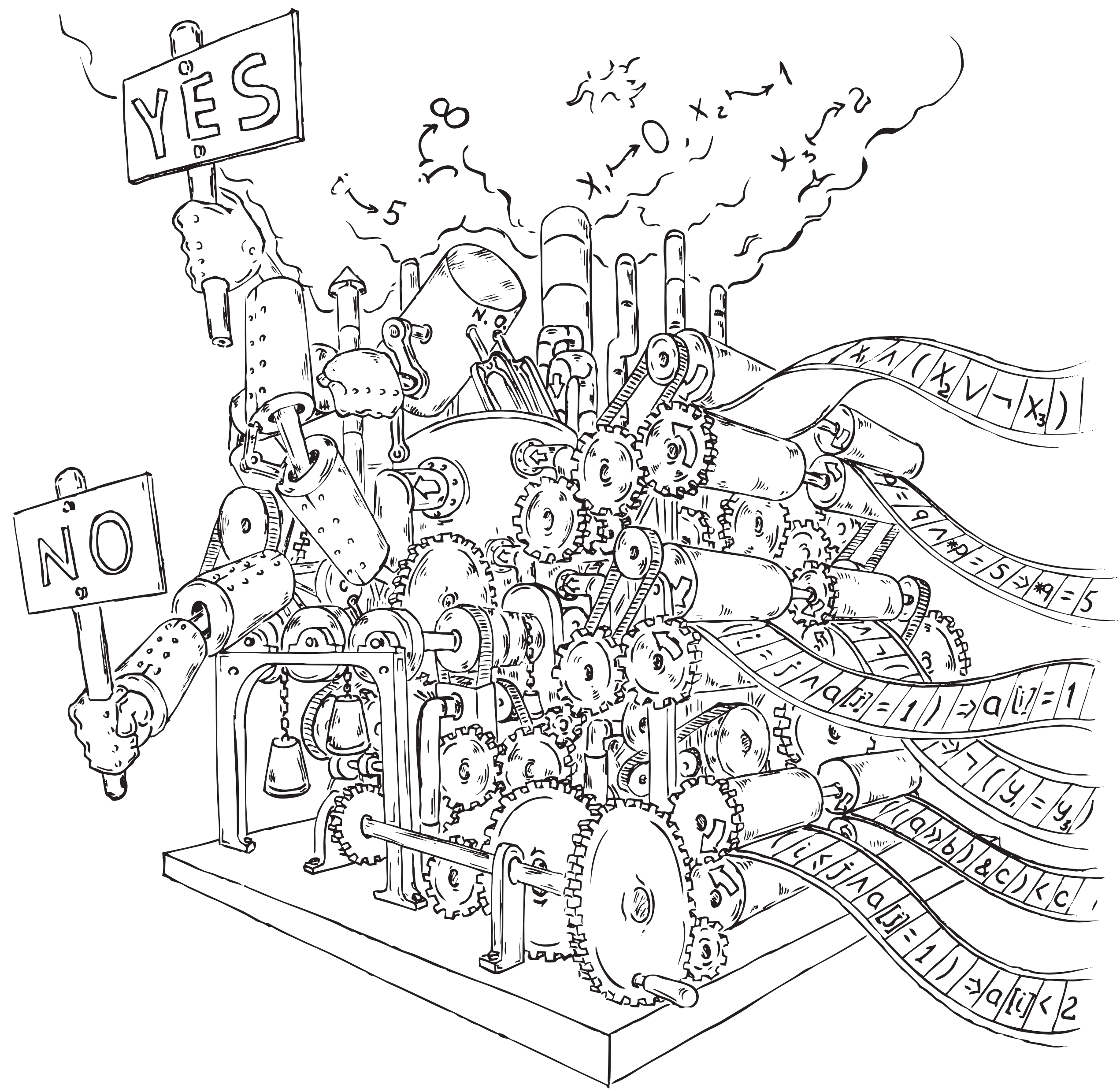
Course overview

program **question**

verifier, synthesizer, fault localizer

logic

SAT, SMT, model finders & checkers



Drawing from “Decision Procedures” by Kroening & Strichman

Course overview

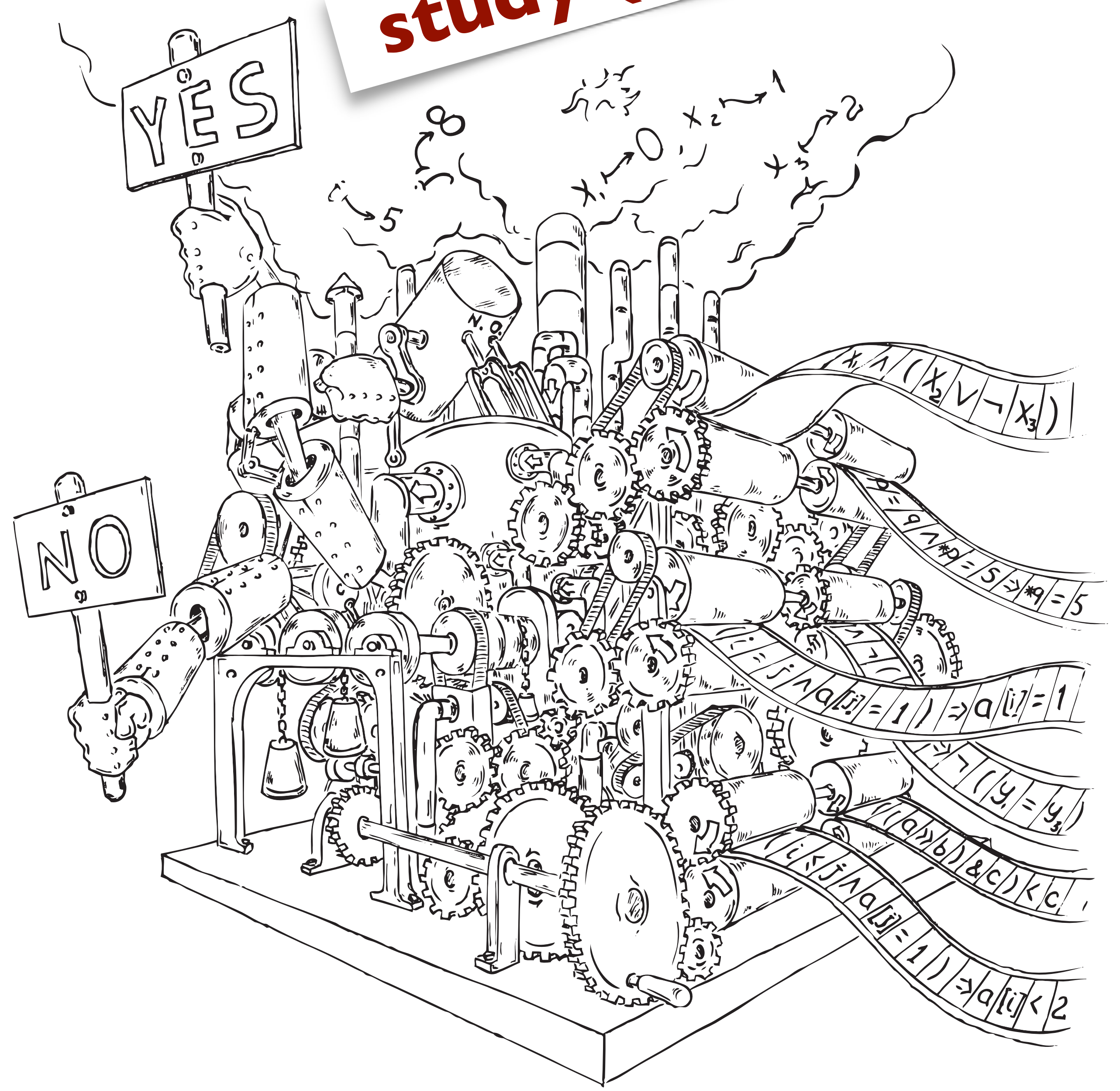
program **question**

verifier, synthesizer, fault localizer

logic

SAT, SMT, model finders & checkers

study (part I)



Drawing from "Decision Procedures" by Kroening & Strichman

Course overview

program **question**

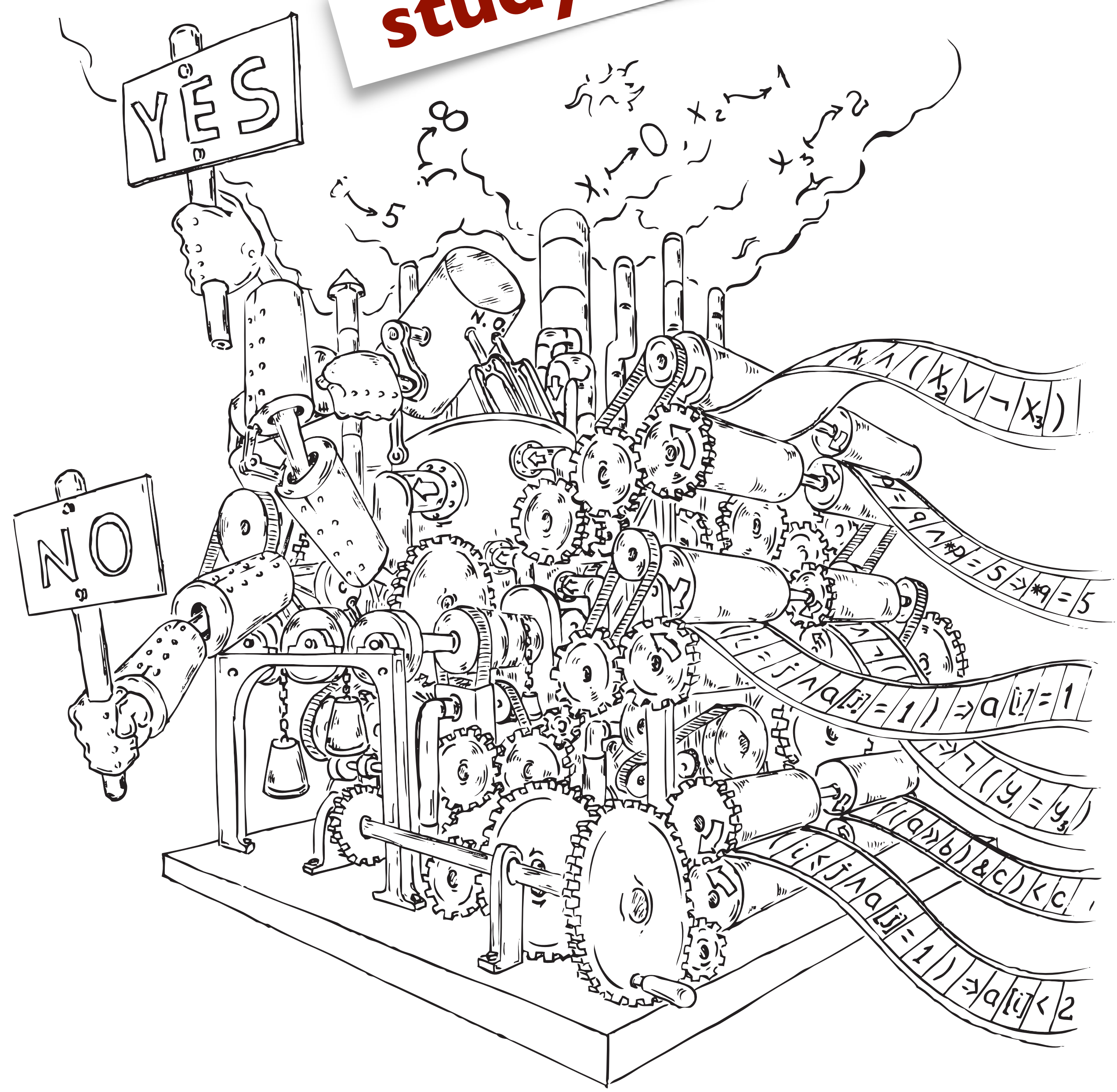
verifier, synthesizer, fault localizer

build! (part II)

logic

SAT, SMT, model finders & checkers

study (part I)



Drawing from “Decision Procedures” by Kroening & Strichman

Grading

4 individual homework assignments (50%)

- conceptual problems & proofs (TeX)
- implementations (Racket)
- 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
- see the **course web page** for timeline, deliverables and other details

study (part I)

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

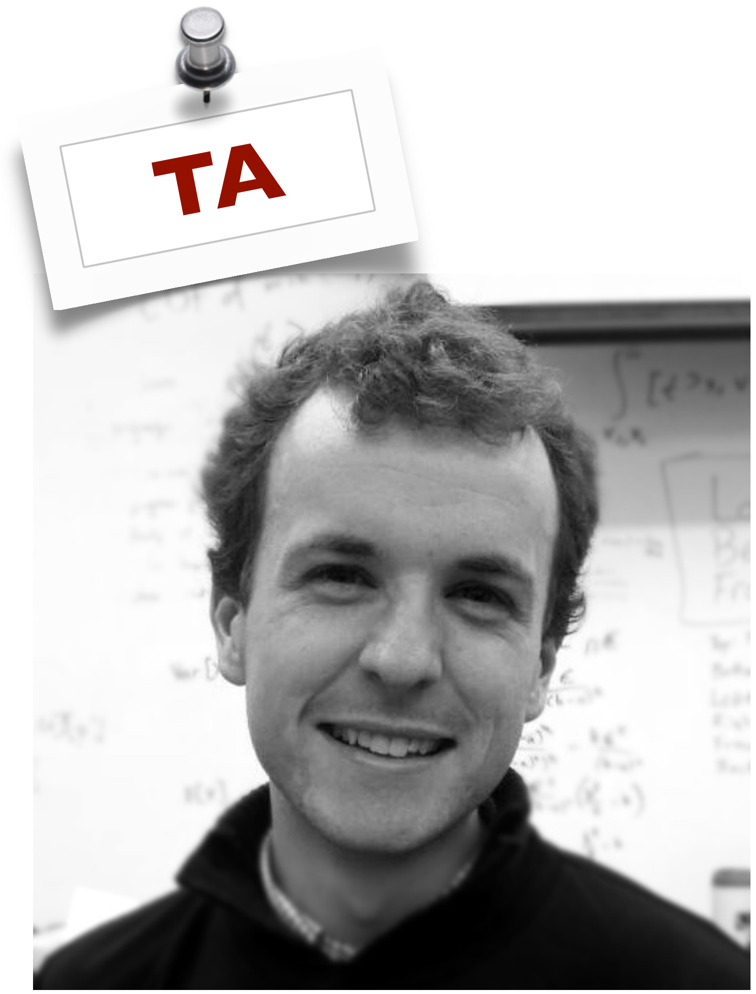
- Ask and answer questions

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

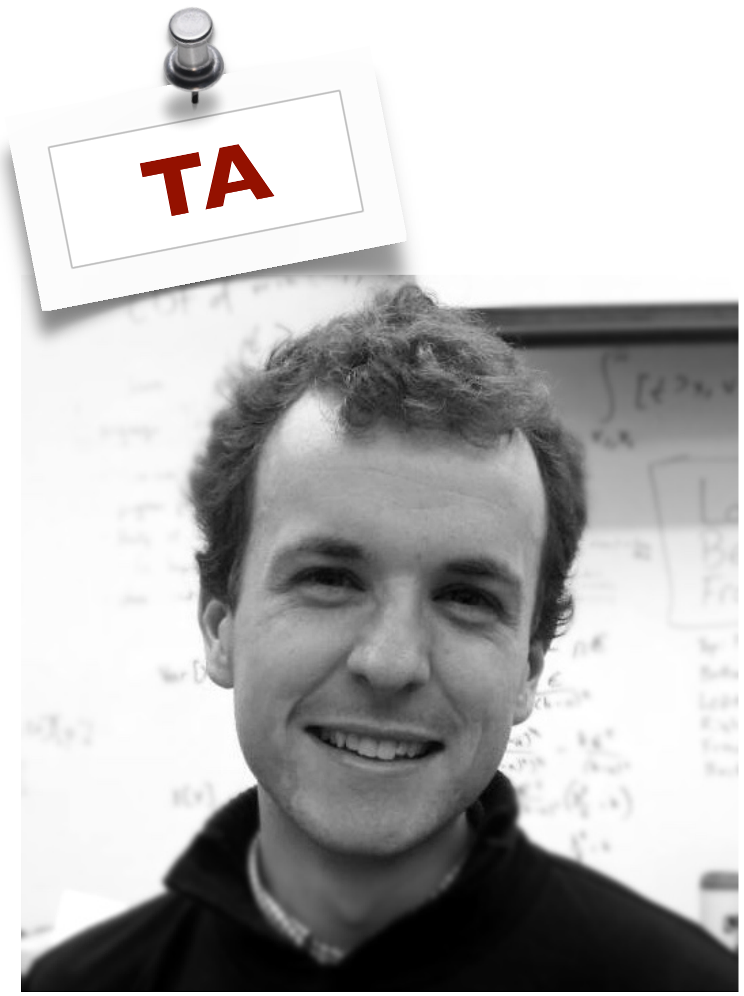
Emina Torlak
PLSE
CSE 596
Fridays 11-12



James Bornholt
PLSE
CSE 218
Wednesdays 11-12

People

Emina Torlak
PLSE
CSE 596
Fridays 11-12



James Bornholt
PLSE
CSE 218
Wednesdays 11-12

students!

Your name
Research area

review

Let's get started! A review of propositional logic

- Syntax
- Semantics
- Satisfiability and validity
- Proof methods
- Semantic judgments
- Normal forms (NNF, DNF, CNF)

Syntax of propositional logic

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

Syntax of propositional logic

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

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

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 F, F_1, F_2:

$\neg F$	"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)

Semantics of propositional logic: interpretations

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

$$I : \{\, p \mapsto \text{true}, q \mapsto \text{false}, \ldots\}$$

Semantics of propositional logic: interpretations

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

$$I : \{ p \mapsto \text{true}, q \mapsto \text{false}, \ldots\}$$

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

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

Semantics of propositional logic: definition

Base cases:

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

Semantics of propositional logic: definition

Base cases:

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

Inductive cases:

Semantics of propositional logic: definition

Base cases:

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \text{true}$
- $I \not\models p$ iff $I[p] = \text{false}$

Inductive cases:

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

Semantics of propositional logic: definition

Base cases:

- $I \models \top$
- $I \not\models \bot$
- $I \models p$ iff $I[p] = \text{true}$
- $I \not\models p$ iff $I[p] = \text{false}$

Inductive cases:

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

Semantics of propositional logic: definition

Base cases:

- $I \vDash \top$
- $I \nvDash \bot$
- $I \vDash p$ iff $I[p] = \text{true}$
- $I \nvDash p$ iff $I[p] = \text{false}$

Inductive cases:

- $I \vDash \neg F$ iff $I \nvDash F$
- $I \vDash F_1 \wedge F_2$ iff $I \vDash F_1$ and $I \vDash F_2$
- $I \vDash F_1 \vee F_2$ iff $I \vDash F_1$ or $I \vDash F_2$
- $I \vDash F_1 \rightarrow F_2$ iff $I \nvDash F_1$ or $I \vDash F_2$
- $I \vDash F_1 \leftrightarrow F_2$ iff $I \vDash F_1$ and $I \vDash F_2$, or $I \nvDash F_1$ and $I \nvDash F_2$

Semantics of propositional logic: example

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

I: $\{p \mapsto \text{true}, q \mapsto \text{false}\}$

?

Semantics of propositional logic: example

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

I: $\{p \mapsto \text{true}, q \mapsto \text{false}\}$

$I \models F$ ✓

Satisfiability & validity of propositional formulas

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

F is **valid** iff $I \vDash F$ 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:

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

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

Proof by deduction (semantic arguments)

Example proof rules:

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

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

F: $p \wedge \neg q$

Proof by deduction (semantic arguments)

Example proof rules:

$$\frac{I \models \neg F}{I \not\models F}$$

$$\frac{I \models F_1 \land F_2}{\begin{array}{l} I \models F_1 \\ I \models F_2 \end{array}}$$

$$\frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \not\models F_1 \land F_2}{I \not\models F_1 \quad \mid \quad I \not\models F_2}$$

F: $p \land \neg q$

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

Proof by deduction (semantic arguments)

Example proof rules:

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

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

F: $p \wedge \neg q$

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

Proof by deduction (semantic arguments)

Example proof rules:

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

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

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:

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

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

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, $\land$)
      i. $I \models q$ (1b, $\neg$)

Proof by deduction (semantic arguments)

Example proof rules:

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

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

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, $\neg$)

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.

$F_1 \iff F_2$ 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 $F_1 \to F_2$ is valid.

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

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.

Why do we care?

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

Getting ready for SAT solving with normal forms

A **normal form** for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.

Getting ready for SAT solving with normal forms

A **normal form** for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.

Assembly language for a logic.

Getting ready for SAT solving with normal forms

A **normal form** for a logic is a syntactic restriction such that every formula in the logic has an equivalent formula in the normal form.

Assembly language for a logic.

Three important normal forms for propositional logic:

- Negation Normal Form (NNF)
- Disjunctive Normal Form (DNF)
- Conjunctive Normal Form (CNF)

Negation Normal Form (NNF)

Atom := Variable | ⊤ | ⊥

Literal := Atom | ¬Atom

Formula := Literal | Formula op Formula

op := ∧ | ∨

Negation Normal Form (NNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | $\neg$Atom

Formula := Literal | Formula op Formula

op := $\wedge$ | $\vee$

- The only allowed connectives are $\wedge$, $\vee$, and $\neg$.
- $\neg$ can appear only in literals.

Negation Normal Form (NNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | $\neg$Atom

Formula := Literal | Formula op Formula

op := $\wedge$ | $\vee$

- The only allowed connectives are $\wedge$, $\vee$, and $\neg$.
- $\neg$ can appear only in literals.

Conversion to NNF performed using **DeMorgan's Laws**:

$\neg(F \wedge G) \iff \neg F \vee \neg G$ $\qquad\qquad$ $\neg(F \vee G) \iff \neg F \wedge \neg G$

Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | $\neg$Atom

Formula := Clause $\vee$ Formula

Clause := Literal | Literal $\wedge$ Clauses

Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | $\neg$Atom

Formula := Clause $\vee$ Formula

Clause := Literal | Literal $\wedge$ Clauses

- Disjunction of conjunction of literals.

Disjunctive Normal Form (DNF)

Atom := Variable | ⊤ | ⊥

Literal := Atom | ¬Atom

Formula := Clause ∨ Formula

Clause := Literal | Literal ∧ Clauses

- Disjunction of conjunction of literals.

To convert to DNF, convert to NNF and distribute ∧ over ∨:

$(F \wedge (G \vee H)) \Longleftrightarrow (F \wedge G) \vee (F \wedge H)$

$((G \vee H) \wedge F) \Longleftrightarrow (G \wedge F) \vee (H \wedge F)$

Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | $\neg$Atom

Formula := Clause $\vee$ Formula

Clause := Literal | Literal $\wedge$ Clauses

- Disjunction of conjunction of literals.
- Deciding satisfiability of a DNF formula is trivial.

To convert to DNF, convert to NNF and distribute $\wedge$ over $\vee$:

$(F \wedge (G \vee H)) \iff (F \wedge G) \vee (F \wedge H)$

$((G \vee H) \wedge F) \iff (G \wedge F) \vee (H \wedge F)$

Disjunctive Normal Form (DNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | $\neg$Atom

Formula := Clause $\vee$ Formula

Clause := Literal | Literal $\wedge$ Clauses

- Disjunction of conjunction of literals.
- Deciding satisfiability of a DNF formula is trivial.
- Why not SAT solve by conversion to DNF?

To convert to DNF, convert to NNF and distribute $\wedge$ over $\vee$:

$(F \wedge (G \vee H)) \iff (F \wedge G) \vee (F \wedge H)$

$((G \vee H) \wedge F) \iff (G \wedge F) \vee (H \wedge F)$

Conjunctive Normal Form (CNF)

Atom := Variable | $\top$ | $\bot$

Literal := Atom | ¬Atom

Formula := Clause $\wedge$ Formula

Clause := Literal | Literal $\vee$ Clauses

- Conjunction of disjunction of literals.
- Deciding the satisfiability of a CNF formula is hard.
- SAT solvers use CNF as their input language.

To convert to CNF, convert to NNF and distribute $\vee$ over $\wedge$

$$(F \vee (G \wedge H)) \iff (F \vee G) \wedge (F \vee H)$$

$$((G \wedge H) \vee F) \iff (G \vee F) \wedge (H \vee F)$$

Conjunctive Normal Form (CNF)

Atom := Variable | ⊤ | ⊥

Literal := Atom | ¬Ator

Formula := Clause ∧ Fo

Clause := Literal | Liter

- Conjunction of disjunction of literals.
- ding the satisfiability of a formula is hard.
- SAT solvers use CNF as their input language.

Why CNF? Doesn't the conversion explode just as badly as DNF?

To convert to CNF, convert to NNF and distribute ∨ over ∧

$(F \lor (G \land H)) \iff (F \lor G) \land (F \lor H)$

$((G \land H) \lor F) \iff (G \lor F) \land (H \lor F)$

Equisatisfiability and Tseitin's transformation

Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Equisatisfiability and Tseitin’s transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Tseitin’s transformation converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Equisatisfiability and Tseitin's transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Tseitin's transformation converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

Equisatisfiability and Tseitin's transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Tseitin's transformation converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

$x \rightarrow (y \land z)$

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

Equisatisfiability and Tseitin's transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Tseitin's transformation converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

$x \rightarrow (y \wedge z)$

$a1$

$a1 \longleftrightarrow (x \rightarrow a2)$

$a2 \longleftrightarrow (y \wedge z)$

Equisatisfiability and Tseitin's transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Tseitin's transformation converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

$x \rightarrow (y \wedge z)$

$a1$
$a1 \rightarrow (x \rightarrow a2)$
$(x \rightarrow a2) \rightarrow a1$
$a2 \longleftrightarrow (y \wedge z)$

Equisatisfiability and Tseitin's transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Tseitin's transformation converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

$x \rightarrow (y \wedge z)$

$a1$

$\neg a1 \vee \neg x \vee a2$

$(x \wedge \neg a2) \vee a1$

$a2 \longleftrightarrow (y \wedge z)$

Equisatisfiability and Tseitin's transformation

Formulas F and G are **equisatisfiable** if they are both satisfiable or they are both unsatisfiable.

Tseitin's transformation converts a propositional formula F into an equisatisfiable CNF formula that is **linear** in the size of F.

Key idea: introduce **auxiliary variables** to represent the output of subformulas, and constrain those variables using CNF clauses.

$x \rightarrow (y \wedge z)$

$a1$
$\neg a1 \vee \neg x \vee a2$
$x \vee a1$
$\neg a2 \vee a1$
$a2 \longleftrightarrow (y \wedge z)$

dpll

A basic SAT solver!

Davis-Putnam-Logemann-Loveland (1962)

```
// Returns true if the CNF formula F is
// satisfiable; otherwise returns false.

DPLL(F)
  G ← BCP(F)
  if G = ⊤ then return true
  if G = ⊥ then return false
  p ← choose(vars(G))
  return DPLL(G{p ↦ ⊤}) ||
         DPLL(G{p ↦ ⊥})
```

Davis-Putnam-Logemann-Loveland (1962)

```
// Returns true if the CNF formula F is
// satisfiable; otherwise returns false.

DPLL(F)
  G ← BCP(F)
  if G = ⊤ then return true
  if G = ⊥ then return false
  p ← choose(vars(G))
  return DPLL(G{p ↦ ⊤}) ||
         DPLL(G{p ↦ ⊥})
```

Boolean constraint propagation applies *unit resolution* until fixed point:

$$\frac{\text{lit} \quad \text{clause[lit]}}{\top}$$

$$\frac{\text{lit} \quad \text{clause}[\neg\text{lit}]}{\text{clause}[\bot]}$$

Summary

Today

- Course overview & logistics
- Review of propositional logic
- A basic SAT solver

Next Lecture

- A modern SAT solver
- Read Chapter 1 of Bradley & Manna