Arrays

Specifications section, **Algorithms** topic, **Lecture 9**

Pavel Panchekha

CS 6110, U of Utah

4 February 2020

Substitution

Solve linear equations by **eliminating variables**

$$\Gamma = \boxed{2 \times x + 3 \times y = 5} \wedge x + 4 \times y = 0$$

$$2 \times x = 5 - 3 \times y$$

$$\downarrow$$

$$2 \times x + 8 \times y = 0$$

$$(5 - 3 \times y) + 8 \times y = 0$$

$$5 + 5 \times y = 0$$

$$y = -1$$

Variable elimination is a standard logic technique

Recall **proof by resolution** in boolean logic

Multiple Variables

$$2y - 12 \le 2z \qquad 2z \le 3 - 2x \qquad -y \le 0$$

$$-1 + 3y - x \le 2z \qquad 2z \le 4 - y + x$$

$$2z \le 6 - y$$

Equivalent to one large equation:

$$\textbf{max}(4y - 24, -2 + 6y - 2x) \le 2z \le \textbf{min}(6 - 4y, 8 - 2y + 2x, 6 - y)$$

$$\downarrow$$

$$\textbf{max}(4y - 24, -2 + 6y - 2x) \le \textbf{min}(6 - 4y, 8 - 2y + 2x, 6 - y)$$

SAT in Integers

SAT problems can be **rephrased as integer** problems

$$x, y := v \mid -x \mid x \vee y \mid x \wedge y$$

v

$$\begin{aligned} 0 &\le v \\ v &\le 1 \end{aligned}$$

a

$$\begin{aligned} a &\le 1 - x \\ 1 - x &\le a \end{aligned}$$

a

$$\begin{aligned} x &\le a \\ y &\le a \\ a &\le x + y \end{aligned}$$

a

$$\begin{aligned} a &\le x \\ a &\le y \\ x + y &\le a + 1 \end{aligned}$$

So solving **integer inequalities is as hard as SAT**.

Class Progress

Logical reasoning	Program logics	Static analysis

First-order Logic	Decision Procedures

Mixing Theories	Equality	Integers	Arrays

Arrays

Reasoning about **reads and writes**

Generating new equations for other solvers

Extensional reasoning about arrays

Symbolic reasoning about quantifiers

Limited quantifiers with **array properties**

Reasoning about sorting, partitions, and append

Array Elements

Delegating is the key to good management

The Solver Query

Statement p

$(\neg a_1 \vee a_2) \wedge (\neg b_1 \vee b_2) \wedge \ldots$

Assignment Γ

$a_1 \wedge \neg b_2 \wedge \ldots$

Today: a solver for **array queries**

$p := i = j \mid u = v \mid \neg p \qquad x := a[i] \qquad a := A \mid a[i := x]$

$$\Gamma = \big(a[i] = a[j]\big) \wedge \big(i \neq j\big) \qquad \Gamma = \big(a[i := x][j] = y\big) \wedge \big(x \neq y\big)$$

$$\Gamma = (a[a[i]] = i) \wedge (a[i] \neq i)$$

Whence Queries

```
def append(l, r):
    out = l[:]
    for i in range(len(r)):
        out.append(r[i])
    return out
```

$$o' = o[\mathbf{len}(o) := r[i]]$$

Starting Simple

Consider a query **without writes** to the array:

$$\Gamma = (a[a[i]] = i) \wedge (a[i] \neq i)$$

How is this different from **term equality**?

$$\Gamma = \big(f(f(i)) = i\big) \wedge \big(f(i) \neq i\big)$$

Equivalence classes, …

An array solver will **need** an equality solver.

Array **writes** are the core challenge

Array Writes

The **read-over-write** axioms:

$$i = j \rightarrow a[i := x][j] = x \qquad\qquad i \neq j \rightarrow a[i := x][j] = a[j]$$

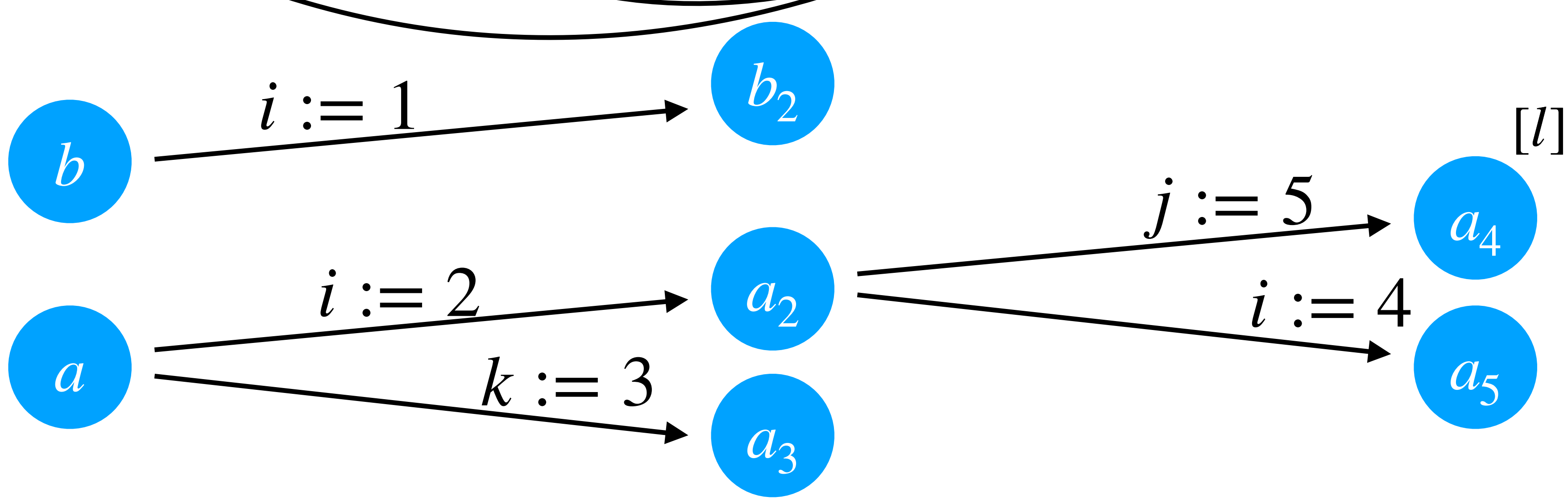
Collect all reads and all writes:

n **writes**

$$a_n = a_{j_n}[i_n := x_n]$$

l **reads**

$$v_l = a_l[i_l]$$



Example

$$\Gamma = \left(a[i := x][j := y][k] = x\right) \wedge \left(x \neq y\right)$$

Writes

$b = a[i := x]$

$c = b[j := y]$

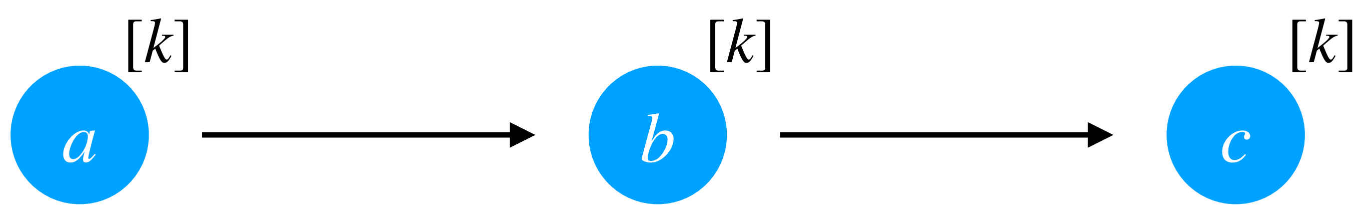
Reads

$u = c[k]$

$v = b[k]$

$w = a[k]$

Generate new formula from the axioms



$i = k \rightarrow v = x$	$j = k \rightarrow u = y$
$i \neq k \rightarrow v = w$	$j \neq k \rightarrow u = v$

No array reasoning!

Exercise

$$\Gamma = \big(a[i := x][j] = b[j := y][i]\big)$$

Writes

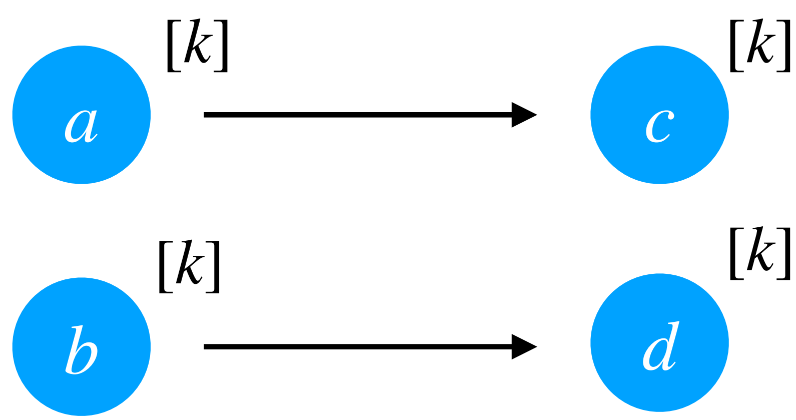
$c = a[i := x]$

$d = b[j := y]$

Reads

$u = c[j]$ $\quad$ $s = a[j]$

$v = d[i]$ $\quad$ $t = b[i]$



$i = j \rightarrow v = y$ $\qquad$ $j = i \rightarrow u = x$

$i \neq j \rightarrow v = t$ $\qquad$ $j \neq i \rightarrow u = w$

Conclusion

Array queries **without writes** are just equality queries:

$$\Gamma = \big(f(f(i)) = i\big) \wedge \big(f(i) \neq i\big)$$

Array writes **translate** to equality queries:

$$\begin{array}{ll} i = k \rightarrow v = x & j = k \rightarrow u = y \\ i \neq k \rightarrow v = w & j \neq k \rightarrow u = v \end{array}$$

Reasoning about **array elements**, not **arrays**

Extensional Arrays

Simple quantified properties

Extensionality

Add a construct for **array equality**:

$$p := i = j \mid u = v \mid \neg p \qquad x := a[i] \qquad a := A \mid a[i := x]$$
$$\mid a = b$$

Define array equality as **quantified equality**:

$$\Big(\forall i, a[i] = b[i]\Big) \rightarrow a = b$$

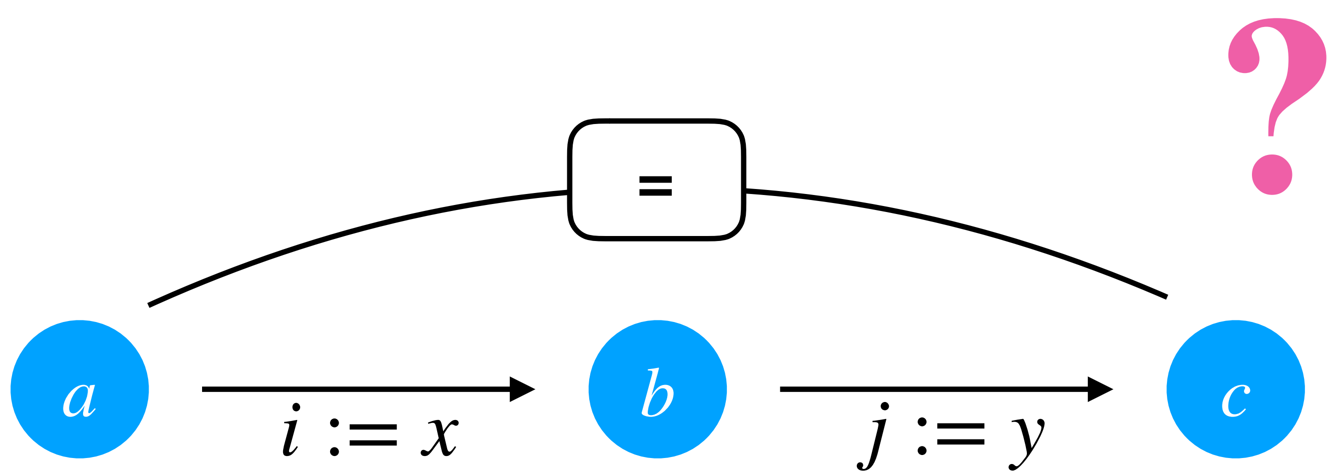
Example

$$i \neq j \wedge a[i] \neq y \wedge a[i := x][j := y] = a$$

$$\exists i, \exists j, \exists a, \exists x, \exists y, \quad \sim\sim\sim\sim\sim\sim\sim \quad \forall k, a[i := x][j := y][k] = a[k]$$

Alternation

So: **how** do we reason about array equality?



Inequalities

$$i \neq j \wedge a[i] \neq y \wedge a[i := x][j := y] \neq a$$

Array **inequalities** do not introduce alternation:

$$\neg \forall k, a[i := x][j := y][k] = a[k]$$

$$\exists i, \exists j, \exists a, \exists x, \exists y, \quad \sim\sim\sim\sim\sim\sim\sim \quad \exists k, a[i := x][j := y][k] \neq a[k]$$

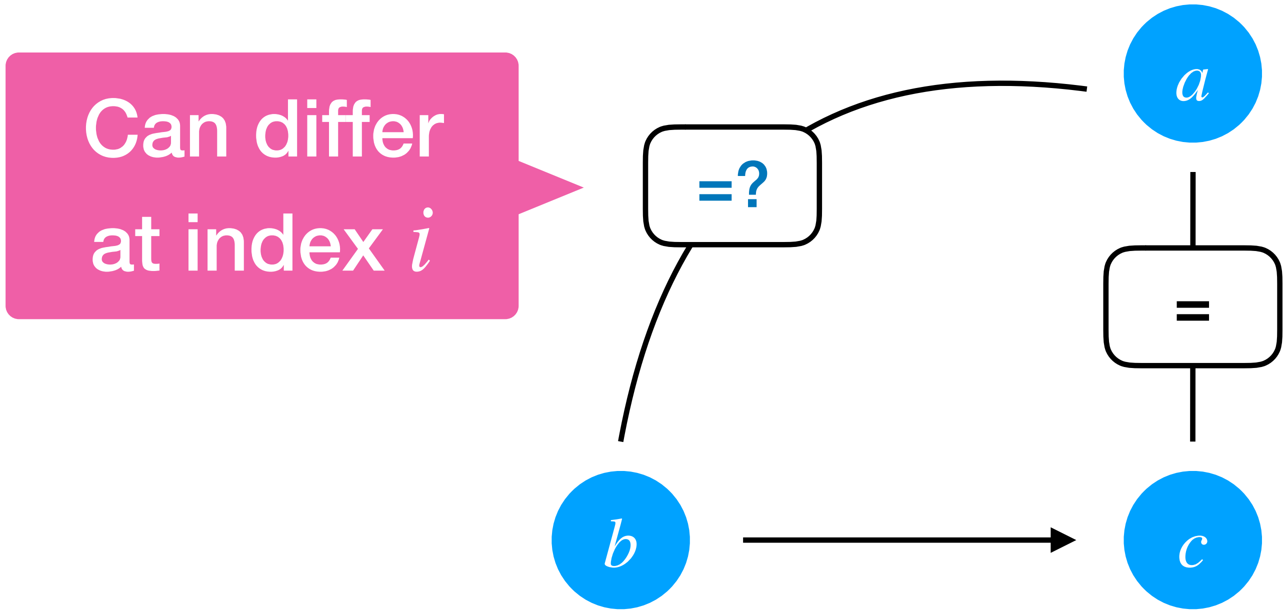
No alternation **New variable**

So **only** need to reason only about true equalities.

Reductions

Equalities are **related to writes**:

$$a = b[i := x]$$



Partial equality

If $a = b[i := x]$, that means:

$$a[i] = x \wedge \forall k, k \neq i \rightarrow a[k] = b[k]$$

$$a' =_{\{i\}} b$$

Partial equality has a **transitivity rule**:

$$a =_I b \wedge b =_J c \rightarrow a =_{(I \cup J)} c$$

It can be **partialized** further with reads:

$$a =_I b \leftrightarrow a =_{i,I} b \wedge a[i] = b[i]$$

Exercise

Rewrite in terms of **partial equality in i, j:**

$$\underline{a[i := 3]} = \underline{b[j := 5]} \wedge a[i] = b[j]$$

$$a'[i] = 3 \qquad\qquad b'[i] = 5$$

$$\underline{a' =_i a} \qquad\qquad \underline{b' =_j b}$$

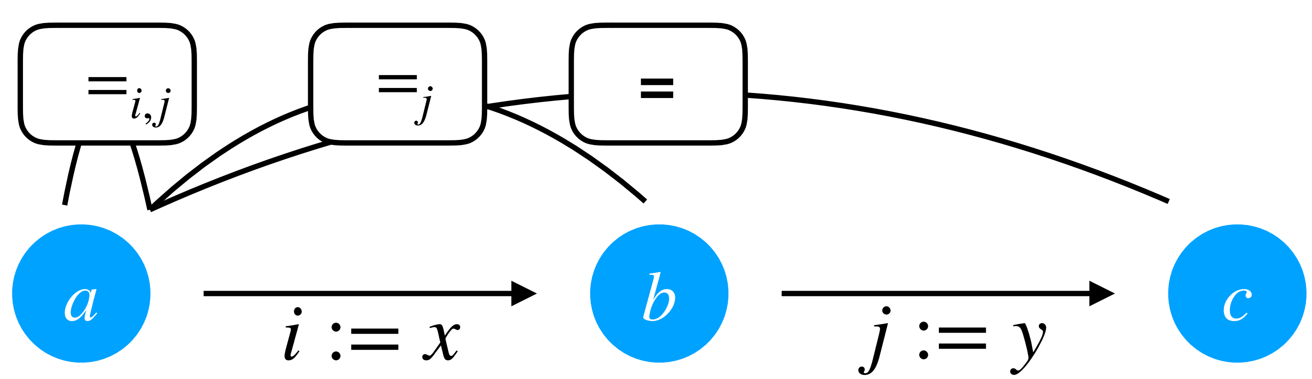
$$a' =_{i,j} a \qquad\qquad b' =_{i,j} b$$

$$a'[j] = a[j] \qquad\qquad b'[i] = b[j]$$

Algorithm

1. Construct graph of array writes
2. Propagate equalities backwards
3. Convert to common partial equality
4. Delete partial equalities (always satisfiable)

Partial equality of array variables



$a[j] = y$ $\quad a =_j b$

$i = j \vee a[i] = x$ $\quad a =_{i,j} a$

Course Updates

Project Proposals

Project Groups

Overall, we have **6 group and 6 solo** projects

Group

Oliver	Abishek
Pranav	Kiranmayee
Skyler	Roxy
Calvin	Saivamshi
Thahnson	Matthew
Kylee	Amit

Solo

Sam
William
Tanmay
Sona
Bradlee
Haochen
Peter

Project Proposals

Get in touch with your project partner

Decide on project idea and start writing proposal

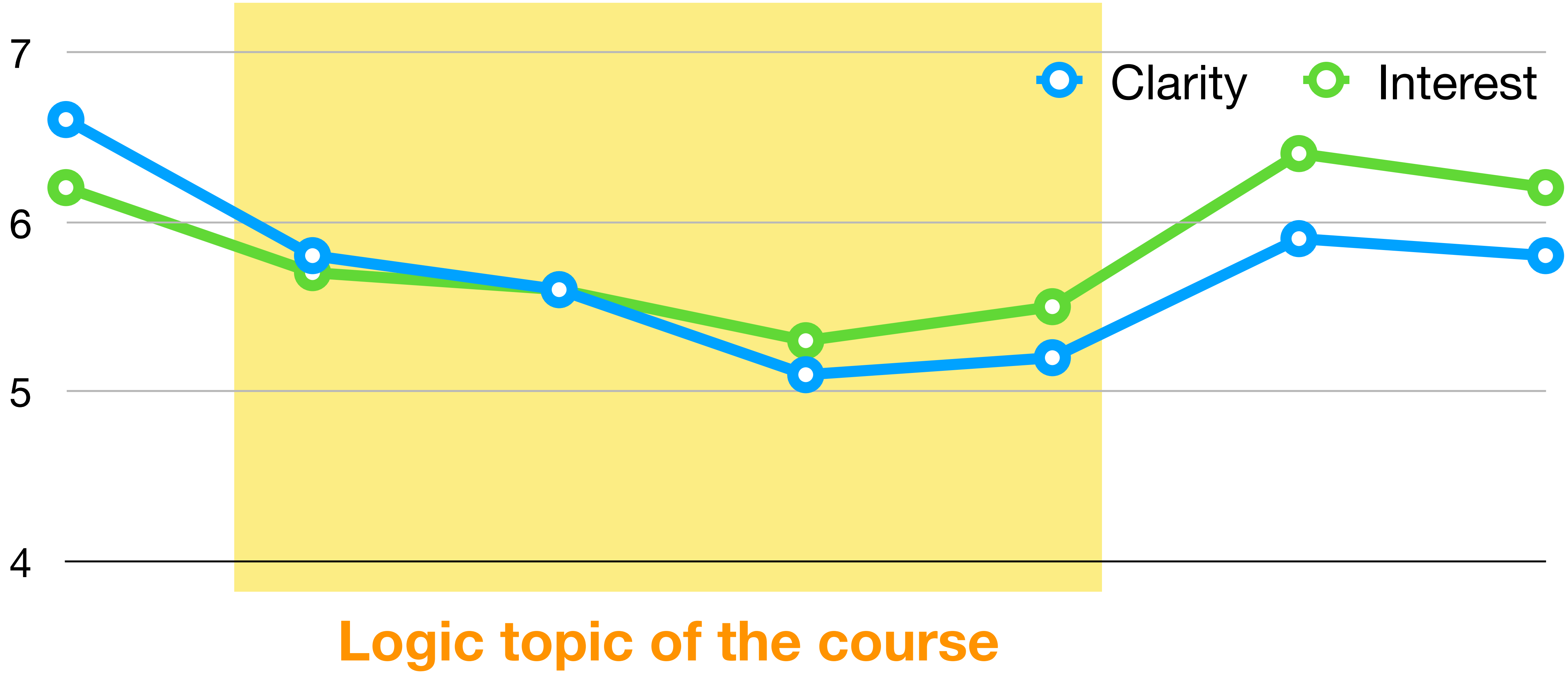
Groups assigned	30 January
Proposals due	6 February
Feedback	11 February
Presentations	18 February

Presentations are **short** (8 minutes group / 4 min solo)

Theory or Practice

Stoked to move onto more algorithmic related material.

Don't listen to theory haters. I wanna freebase the theory stuff.



Array Properties

A quantified fragment for array subsets

Algorithm

Summary of **extensional array** solver:

Negated properties are existential, new variable

Propagate backwards across writes

Read-only formula + **generated domain** formula

Easy

No arrays

What else does this handle?

Array Properties

Array equality is our first **property of arrays**

sorted(A) **partitioned**(A_l, A_r) $x \notin A$

Each of these is defined by **quantification over indices**:

$$\textbf{sorted}(A) := \forall i, \forall j, i < j \rightarrow A[i] < A[j]$$

$$\textbf{partitioned}(A_l, A_r) := \forall i, \forall j, A_l[i] < A_r[j]$$

$$x \notin A := \forall i, A[i] \neq x$$

Array Properties

$$\textbf{sorted}(A) := \underline{\forall i, \forall j,}\ \underline{i < j} \rightarrow \underline{A[i] < A[j]}$$

Common pattern in array predicates:

- **Universal quantifier** over index variables
- Domain formula over **index variables** (guard)
- Domain formula over **array values**

Pattern called the **array property fragment**

- Array equality one example of **an array property**

Exercise

Which of these are in the **array property fragment**:

$$\forall i, a[i] = i \rightarrow \exists j, a[j] = 2$$

$$\forall i, \forall j, A[i] = B[j]$$

$$\forall i, \forall j, a[i] < i \rightarrow a[j] = j$$

$$a \neq b \rightarrow \exists i, a[i] = b[i]$$

$$\forall s, \forall t, \forall r, s < r \wedge r < t \rightarrow B[r] = A[s] + A[t]$$

Solving fragment

Details in textbook

$$\underline{\forall i, \cdots, \forall j,}\ \underline{F(i, \ldots, j)} \rightarrow \underline{G(a[i], \ldots, b[j])}$$

1. Construct graph of array writes

2. Propagate properties backwards

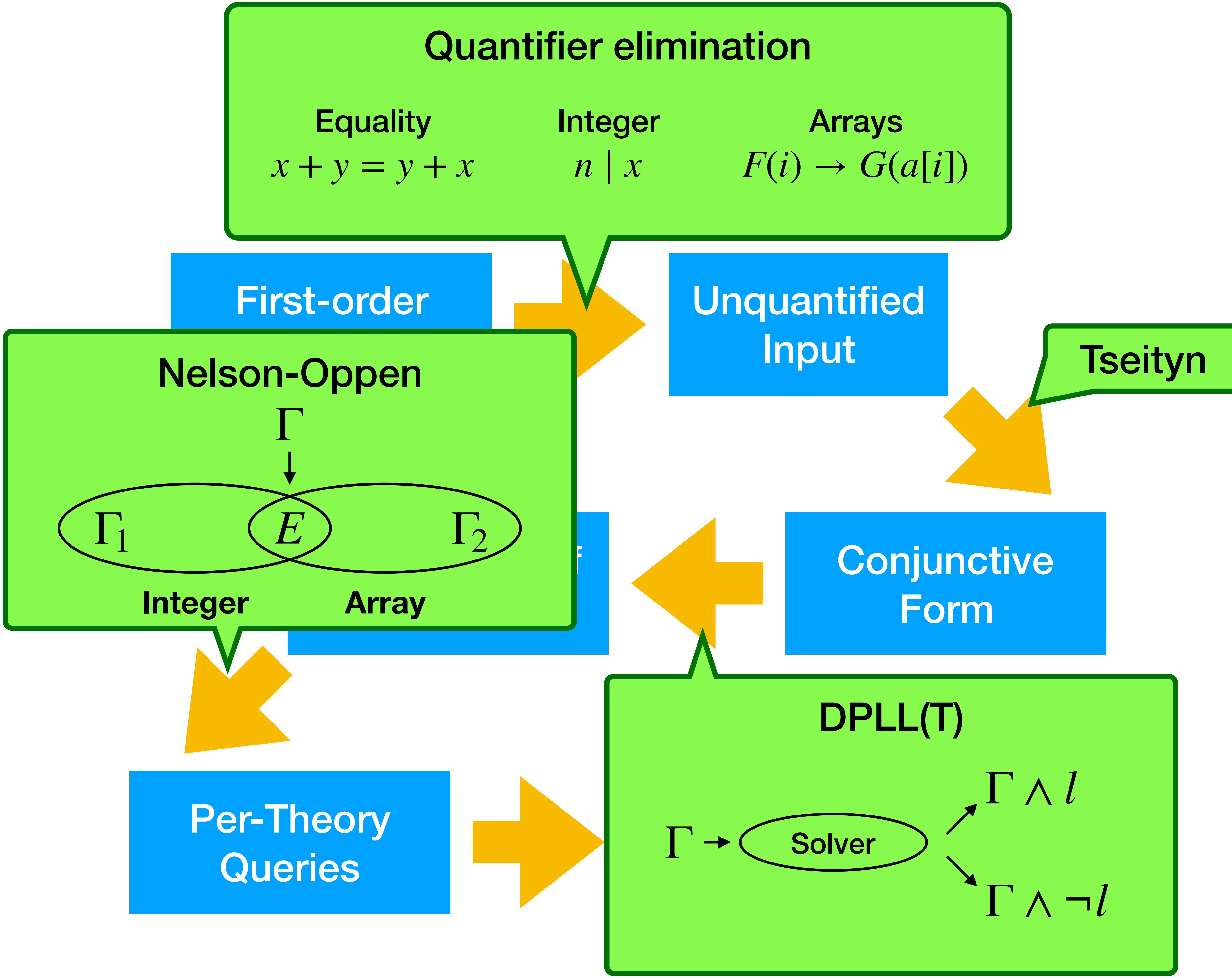
$$G(a[n := x][i]) \leftrightarrow G(v) \wedge \begin{pmatrix} i = n \wedge v = x \vee \\ i \neq n \wedge v = a[i] \end{pmatrix}$$

3. Replace array values with new variables

$$a[i] < a[j] \rightsquigarrow x < y \wedge \left(i = j \rightarrow x = y\right)$$

Summary

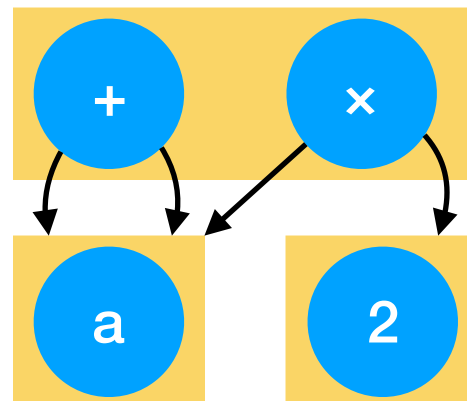
How an SMT solver works



Quantifier elimination
Equality
$x + y = y + x$
Integer
$n \mid x$
Arrays
$F(i) \rightarrow G(a[i])$
First-order
Unquantified Input
Tseityn
Nelson-Oppen
Γ
Γ_1
E
Γ_2
Integer
Array
Conjunctive Form
Per-Theory Queries
DPLL(T)
Γ
Solver
$\Gamma \wedge l$
$\Gamma \wedge \neg l$

Domain Reasoning

Equality



Model building

Term database

Equivalence classes

Integers

$$x + 2y \leq z$$
$$z \leq 2x - y$$
$$\downarrow$$
$$x + 2y \leq 2x - y$$

Matrix form

Variable elimination

Complexity of integers

Arrays

$$a[k := 2] = b$$
$$\downarrow$$
$$b[k] = 2$$
$$a =_k b$$

Mutation graph

Backward propagation

Translation to theory

Next class:

Web Pages

To do:

- ☐ Course feedback
- ☐ Read Chapter 11
- ☐ Project Proposal