Substitution

Solve linear equations by **eliminating variables**

\[
\Gamma = 2 \times x + 3 \times y = 5 \land x + 4 \times y = 0
\]

\[
2 \times x = 5 - 3 \times y \quad \downarrow
\]

\[
2 \times x + 8 \times y = 0
\]

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

\[
5 + 5 \times y = 0
\]

\[
y = -1
\]

**Variable elimination** is a standard logic technique

Recall **proof by resolution** in boolean logic
Multiple Variables

\begin{align*}
2y - 12 & \leq 2z \\
-1 + 3y - x & \leq 2z \\
2z & \leq 3 - 2x \\
2z & \leq 4 - y + x \\
2z & \leq 6 - y \\
- y & \leq 0
\end{align*}

Equivalent to one large equation:

\[
\max(4y - 24, - 2 + 6y - 2x) \leq 2z \leq \min(6 - 4y, 8 - 2y + 2x, 6 - y)
\]
SAT in Integers

SAT problems can be repphrased as integer problems

\[ x, y := v \mid -x \mid x \lor y \mid x \land y \]

\[ \begin{align*}
  v &\leq 0 \\
  v &\leq 1 \\
  a &\leq 1 - x \\
  1 - x &\leq a \\
  x &\leq a \\
  y &\leq a \\
  a &\leq x + y \\
  a &\leq x \\
  a &\leq y \\
  x + y &\leq a + 1
\end{align*} \]

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** \( p \)
\((\neg a_1 \lor a_2) \land (\neg b_1 \lor b_2) \land \ldots\)

**Assignment** \( \Gamma \)
\(a_1 \land \neg b_2 \land \ldots\)

Today: a solver for **array queries**

\( p := i = j \mid u = v \mid \neg p \quad x := a[i] \quad a := A \mid a[i := x] \)

\[ \Gamma = (a[i] = a[j]) \land (i \neq j) \quad \Gamma = (a[i := x][j] = y) \land (x \neq y) \]

\[ \Gamma = (a[a[i]] = i) \land (a[i] \neq i) \]
def append(l, r):
    out = l[:]
    for i in range(len(r)):
        out.append(r[i])
        o' = o[len(o) := r[i]]
    return out
Starting Simple

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

\[ \Gamma = (a[a[i]] = i) \land (a[i] \neq i) \]

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

\[ \Gamma = (f(f(i)) = i) \land (f(i) \neq i) \]

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

Array **writes** are the core challenge
Array Writes

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

\[
i = j \rightarrow a[i := x][j] = x \quad \quad i \neq j \rightarrow a[i := x][j] = a[j]
\]

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

\[ \Gamma = (a[i := x][j := y][k] = x) \land (x \neq y) \]

**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

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

**Writes**
\[
c = a[i := x] \\
d = b[j := y]
\]

**Reads**
\[
u = c[j] \\
v = d[i] \\
s = a[j] \\
t = b[i]
\]

**Diagram**

\[
\begin{array}{ccc}
a[k] & \rightarrow & c[k] \\
b[k] & \rightarrow & d[k]
\end{array}
\]

**Conditions**
\[
i = j \rightarrow v = y \\
i \neq j \rightarrow v = t \\
j = i \rightarrow u = x \\
j \neq i \rightarrow u = w
\]
Conclusion

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

\[ \Gamma = (f(f(i)) = i) \land (f(i) \neq i) \]

Array writes **translate** to equality queries:

\[
\begin{align*}
 i = k & \rightarrow v = x & j = k & \rightarrow u = y \\
 i \neq k & \rightarrow v = w & j \neq k & \rightarrow u = v 
\end{align*}
\]

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

Simple quantified properties
Extensionality

Add a construct for **array equality**:

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

Define array equality as **quantified equality**:

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

\[ i \neq j \land a[i] \neq y \land a[i := x][j := y] = a \]

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

**Alternation**

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

\[ i \neq j \land a[i] \neq y \land a[i := x][j := y] \neq a \]

Array inequalities do not introduce alternation:

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

No alternation \hspace{2cm} New variable

So only need to reason only about true equalities.
Reductions

Equalities are related to writes:

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

Can differ at index \( i \)
Partial equality

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

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

Partial equality has a **transitivity rule**:

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

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

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

Rewrite in terms of \textbf{partial equality in }i, j:

\[
\begin{align*}
\mathit{a}[i := 3] &= \mathit{b}[j := 5] \land \mathit{a}[i] = \mathit{b}[j] \\
\mathit{a}'[i] &= 3 \\
\mathit{a}' &= i \mathit{a} \\
\mathit{a}'[j] &= \mathit{a}[j] \\
\mathit{b}'[i] &= 5 \\
\mathit{b}' &= j \mathit{b} \\
\mathit{b}'[i] &= \mathit{b}[j]
\end{align*}
\]
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

\[ a[j] = y \quad i = j \lor a[i] = x \quad a =_{j} b \quad a =_{i,j} a \]
Course Updates

Project Proposals
# Project Groups

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

<table>
<thead>
<tr>
<th>Group</th>
<th>Solo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oliver</td>
<td>Sam</td>
</tr>
<tr>
<td>Pranav</td>
<td>William</td>
</tr>
<tr>
<td>Skyler</td>
<td>Tanmay</td>
</tr>
<tr>
<td>Calvin</td>
<td>Sona</td>
</tr>
<tr>
<td>Thahnson</td>
<td>Bradlee</td>
</tr>
<tr>
<td>Kylee</td>
<td>Haochen</td>
</tr>
<tr>
<td></td>
<td>Peter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abishek</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kiranmayee</td>
</tr>
<tr>
<td>Roxy</td>
</tr>
<tr>
<td>Saivamshi</td>
</tr>
<tr>
<td>Matthew</td>
</tr>
<tr>
<td>Amit</td>
</tr>
</tbody>
</table>
# Project Proposals

Get in touch with your project partner

Decide on project idea and start writing proposal

<table>
<thead>
<tr>
<th>Event</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups assigned</td>
<td>30 January</td>
</tr>
<tr>
<td>Proposals due</td>
<td>6 February</td>
</tr>
<tr>
<td>Feedback</td>
<td>11 February</td>
</tr>
<tr>
<td>Presentations</td>
<td>18 February</td>
</tr>
</tbody>
</table>

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

\[
\text{sorted}(A) \quad \text{partitioned}(A_l, A_r) \quad x \notin A
\]

Each of these is defined by quantification over indices:

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

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

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

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

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**:

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

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

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

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

\[
\forall s, \forall t, \forall r, s < r \land r < t \rightarrow B[r] = A[s] + A[t]
\]
Solving fragment

\[ \forall i, \ldots, \forall j, F(i, \ldots, j) \rightarrow G(a[i], \ldots, b[j]) \]

1. Construct graph of array writes
2. Propagate properties backwards

\[ G(a[n := x][i]) \leftrightarrow G(v) \land \left( i = n \land v = x \lor \begin{cases} i \neq n \land v = a[i] \end{cases} \right) \]

3. Replace array values with new variables

\[ a[i] < a[j] \iff x < y \land (i = j \rightarrow x = y) \]
Summary

How an SMT solver works
Quantifier elimination

First-order

Nelson-Oppen

Tseityn

Conjunctive Form

DPLL(T)

Per-Theory Queries

Unquantified Input

Integer

Arrays

Equality

Integer

Arrays

$\Gamma \rightarrow \Gamma \wedge l$

$\Gamma \wedge \neg l$

$x + y = y + x$

$n \mid x$

$F(i) \rightarrow G(a[i])$

$E$

$\Gamma_1$

$\Gamma_2$

Integer

Array

Solver
Domain Reasoning

Equality

Model building
Term database
Equivalence classes

Integers

Matrix form
Variable elimination
Complexity of integers

Arrays

Mutation graph
Backward propagation
Translation to theory
Next class:

Web Pages

To do:

☐ Course feedback
☐ Read Chapter 11
☐ Project Proposal