Computation and Inference

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

July 13, 2018
You have a sequence of numbers, e.g., 9, 7, 10, 9, 5, 4, 10.
The task is to find the length of the longest increasing subsequence.
Here the longest subsequence is 7, 9, 10, and its length is 3.
Patience solitaire is a card game where cards are placed, one by one, into a sequence of columns.
Each card is placed at the bottom of the leftmost column where it is no bigger than the current bottom card in the column.
If there is no such column, we start a new column at the right.
Show that the number of columns left at the end yields the length of the longest increasing subsequence.
Computing, like mathematics, is the study of reusable abstractions.

Abstractions in computing include numbers, lists, channels, processes, protocols, and programming languages.

Computing is *abstraction engineering* and logic is the calculus of computing.

Inference is a mechanism for generating knowledge about abstractions through abduction, deduction, and induction.

Inference algorithms employ inference steps to perform computation as solvability-preserving problem transformations.

Algorithm = Inference + Strategy + Indexing
Beyond computing, the world itself is increasingly an interplay of abstractions.

Caches, files, IP addresses, avatars, friends, likes, hyperlinks, packets, network protocols, and cyber-physical systems are all examples of abstractions in daily use.

Such abstract entities and the relationships can be expressed clearly and precisely in logic.

In computing, and elsewhere, we are increasingly dependent on formalization as a way of managing the abstract universe.

Logic has been *unreasonably* effective in computing, with an impact that spans theory of computation, hardware design, software verification, databases, programming languages, artificial intelligence, and systems biology.
We give a brief introduction to the mechanisms of deductive inference based on an article in the recent *Handbook of Model Checking* (eds. Clarke, Henzinger, Veith, and Bloem).

- These deduction techniques are used heavily in hardware and software specification and verification.
- But they have broad applicability beyond verification.
Logic studies the *trinity* between *language*, *interpretation*, and *proof*.

**Language**: What can be expressed?

**Interpretation**: What is the intended meaning?
- Meaning is usually *compositional*: Follows the syntax
- Some symbols have fixed interpretation: *connectives*, *equality*, *quantifiers*
- The interpretation of other symbols is allowed to vary
  - *variables*, *functions*, and *predicates*
- *Assertions* either hold or fail to hold in a given interpretation
- A *valid* assertion holds in every interpretation
- If an assertion is not valid, then its negation is *satisfiable*

**Proofs**: How do we constructively demonstrate the validity of a statement?
Language: The syntactic representation of conditions is using propositional formulas:

\[ \phi ::= P \mid \neg \phi \mid \phi_1 \lor \phi_2 \mid \phi_1 \land \phi_2 \mid \phi_1 \Rightarrow \phi_2 \]

Examples of formulas are \( p, p \land \neg p, p \lor \neg p, (p \land \neg q) \lor \neg p \).

Interpretation: \( M[\phi] \) is the meaning of \( \phi \) in interpretation \( M \) and is computed using truth tables:

<table>
<thead>
<tr>
<th>( \phi )</th>
<th>( p )</th>
<th>( q )</th>
<th>( \neg p )</th>
<th>( p \lor q )</th>
<th>( p \land q )</th>
<th>( p \Rightarrow q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_1(\phi) )</td>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \top )</td>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \top )</td>
</tr>
<tr>
<td>( M_2(\phi) )</td>
<td>( \bot )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \bot )</td>
<td>( \top )</td>
</tr>
<tr>
<td>( M_3(\phi) )</td>
<td>( \top )</td>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \top )</td>
<td>( \bot )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( M_4(\phi) )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \bot )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \top )</td>
</tr>
</tbody>
</table>

A formula \( \phi \) is *satisfiable* if \( M \models \phi \) for some \( M \), and is *unsatisfiable*, otherwise.
The basic judgement is $\Gamma \vdash \Delta$ asserting the validity of $\land \Gamma \Rightarrow \lor \Delta$, where $\Gamma$ and $\Delta$ are sets (or bags) of formulas.

<table>
<thead>
<tr>
<th></th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ax</strong></td>
<td>$\Gamma, A \vdash A, \Delta$</td>
<td></td>
</tr>
<tr>
<td>$\neg$</td>
<td>$\Gamma \vdash A, \Delta$</td>
<td>$\Gamma, A \vdash \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \neg A \vdash \Delta$</td>
<td>$\Gamma \vdash \neg A, \Delta$</td>
</tr>
<tr>
<td>$\lor$</td>
<td>$\Gamma, A \vdash \Delta$</td>
<td>$\Gamma, B \vdash \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \lor B \vdash \Delta$</td>
<td>$\Gamma \vdash A \lor B, \Delta$</td>
</tr>
<tr>
<td>$\land$</td>
<td>$\Gamma, A \vdash \Delta$</td>
<td>$\Gamma, B \vdash \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \land B \vdash \Delta$</td>
<td>$\Gamma \vdash A \land B, \Delta$</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$\Gamma, B \vdash \Delta$</td>
<td>$\Gamma, A \vdash \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, A \Rightarrow B \vdash \Delta$</td>
<td>$\Gamma \vdash A \Rightarrow B, \Delta$</td>
</tr>
<tr>
<td><strong>Cut</strong></td>
<td>$\Gamma, A \vdash \Delta$</td>
<td>$\Gamma, A \vdash \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash A, \Delta$</td>
<td>$\Gamma \vdash \Delta$</td>
</tr>
</tbody>
</table>
What is an Inference Algorithm?

- We want a systematic way to construct and analyze decision procedures for the satisfiability of formulas in a given class.
- An $\Sigma$-inference structure $\langle \Psi, \vdash, \Lambda, M \rangle$ consists of
  - $\Psi$, a set of logical states
  - $\vdash$, the reduction relation between states
  - $\Lambda$, a map from states to $\Sigma$-formulas
  - $M$, which extracts models from canonical states
- An inference system is an inference structure that is
  - Conservative: If $\psi \vdash \psi'$, then $\Lambda(\psi)$ and $\Lambda(\psi')$ are equisatisfiable.
  - Progressive: $\vdash$ is well-founded.
  - Canonizing: If $\psi \nvdash \psi'$ for any $\psi'$, then either $\psi$ is $\bot$ (i.e., unsatisfiable) or $\psi$ is in a canonical form so that $M(\psi)$ is a model for $\Lambda(\psi)$.
- An inference algorithm is a collection of effective inference rules defining the reduction relation.
Ordered Resolution as an Inference Algorithm

- A *clause* is a disjunction $l_1 \lor \ldots \lor l_n$ of literals, where each *literal* $l$ is a propositional variable $p$ or its negation $\neg p$.
- Any propositional formula has an equisatisfiable representation in *conjunctive normal form* as a conjunction of clauses.
- Ordered resolution is an algorithm for CNF satisfiability.
- Input $K$ is a set of ordered clauses (disjunctions of literals $p$ or $\neg p$), where the literals appear in decreasing order w.r.t. some order e.g., $q \prec \neg q \prec p \prec \neg p$.
- Tautologies, i.e., clauses containing both $p$ and $\neg p$, are deleted from initial input.

<table>
<thead>
<tr>
<th>Res</th>
<th>$K, p \lor \Gamma_1, \neg p \lor \Gamma_2$</th>
<th>$\Gamma_1 \lor \Gamma_2 \notin K$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$K, p \lor \Gamma_1, \neg p \lor \Gamma_2, \Gamma_1 \lor \Gamma_2$</td>
<td>$\Gamma_1 \lor \Gamma_2$ is not tautological</td>
</tr>
</tbody>
</table>

| Contrad      | $K, p, \neg p$ | $\bot$ |
Ordered Resolution as an Inference Algorithm

\[
\begin{align*}
(K_0 =) \quad & \neg p \lor \neg q \lor r, \quad \neg p \lor q, \quad p \lor r, \quad \neg r \\
(K_1 =) \quad & \neg q \lor r, \quad K_0 \quad \text{Res} \\
(K_2 =) \quad & q \lor r, \quad K_1 \quad \text{Res} \\
(K_3 =) \quad & r, \quad K_2 \quad \text{Res} \\
\bot \quad & \text{Contrad}
\end{align*}
\]

- Drop the clause \( \neg r \), and we reach an irreducible state from which a truth assignment \( \{ r \mapsto \top, q \mapsto \bot, p \mapsto \bot \} \) can be constructed.
Ordered Resolution as an Inference Algorithm

- The resolution inference system is strongly conservative (i.e., model-preserving, not just satisfiability-preserving): \( \Gamma_1 \lor \Gamma_2 \) is satisfiable if \( p \lor \Gamma_1 \) and \( \neg p \lor \Gamma_2 \) are.

- It is progressive: Bounded number of new clauses in the input variables.

- It is canonizing: Build a model \( M \) by assigning to atoms \( p_1 \) to \( p_n \) within a series of partial assignments \( M_0, \ldots, M_n \):
  - \( M_0 \) is the empty truth assignment.
  - \( M_{i+1} = M_i[p_{i+1} \leftrightarrow v] \), where \( v = \top \) iff there is some clause \( p_{i+1} \lor \Gamma \) in the irreducible state \( K \) such that \( M_i \models \neg \Gamma \).

- If \( M_i \models \neg \Gamma \), then for any clause \( \neg p_i \lor \Delta \), \( M_i \models \Delta \) since \( \Gamma \lor \Delta \in K \).

- Invariant: \( M_i \models \Gamma \) for all clauses \( \Gamma \) in \( K \) in the atoms \( p_1, \ldots, p_i \).
Boolean Satisfiability (SAT) using CDCL

Conflict-Driven Clause Learning (CDCL) builds a partial assignment through guessing (Select), propagation (Propagate), and conflict-triggered retraction (Backjump).

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propagate</td>
<td>$h, \langle M \rangle, K, C$</td>
<td>$\Gamma \equiv l \lor \Gamma' \in K \cup C$</td>
</tr>
<tr>
<td></td>
<td>$h, \langle M, l[\Gamma] \rangle, K, C$</td>
<td>$M</td>
</tr>
<tr>
<td>Select</td>
<td>$h, \langle M \rangle, K, C$</td>
<td>$M \not</td>
</tr>
<tr>
<td></td>
<td>$h + 1, \langle M; l[] \rangle, K, C$</td>
<td>$M \not</td>
</tr>
<tr>
<td>Conflict</td>
<td>$0, \langle M \rangle, K, C$</td>
<td>$M</td>
</tr>
<tr>
<td>Backjump</td>
<td>$h + 1, \langle M \rangle, K, C$</td>
<td>$M</td>
</tr>
<tr>
<td></td>
<td>$h', \langle M_{\leq h'}, l[\Gamma'] \rangle, K, C \cup {\Gamma'}$</td>
<td>$\langle h', \Gamma' \rangle = analyze(\psi)(\Gamma)$ for $\psi = h, \langle M \rangle, K, C$</td>
</tr>
</tbody>
</table>

CDCL state consists of a decision level $h$, a partial assignment $\langle M \rangle$ (a list of decision literals and derived literals with justification clauses), the input clause set $K$, and the learned clause set $C$. 
Let $K$ be
\[
\{p \lor q, \neg p \lor q, p \lor \neg q, s \lor \neg p \lor q, \neg s \lor p \lor \neg q, \neg p \lor r, \neg q \lor \neg r\}.
\]

The partial assignment is extended through selection and propagation until a $K \cup C$ contains a conflict, a clause that is falsified by the current partial assignment.

<table>
<thead>
<tr>
<th>step</th>
<th>$h$</th>
<th>$M$</th>
<th>$K$</th>
<th>$C$</th>
<th>$\Gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>select $s$</td>
<td>1</td>
<td>$; s$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>select $r$</td>
<td>2</td>
<td>$; s; r$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>propagate</td>
<td>2</td>
<td>$; s; r, \neg q[\neg q \lor \neg r]$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>propagate</td>
<td>2</td>
<td>$; s; r, \neg q, p[p \lor q]$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>conflict</td>
<td>2</td>
<td>$; s; r, \neg q, p$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>$\neg p \lor q$</td>
</tr>
</tbody>
</table>
The conflict clause $p \lor q$ is *analyzed* by resolving it against the justification clauses for assignments at the current level (i.e., level 2) until there is a unique literal at the current level.

<table>
<thead>
<tr>
<th>step</th>
<th>$h$</th>
<th>$M$</th>
<th>$K$</th>
<th>$C$</th>
<th>$\Gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>conflict</td>
<td>2</td>
<td>$s; r, \neg q, p$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>$\neg p \lor q$</td>
</tr>
<tr>
<td>backjump</td>
<td>0</td>
<td>$\emptyset$</td>
<td>$K$</td>
<td>$q$</td>
<td>_</td>
</tr>
<tr>
<td>propagate</td>
<td>0</td>
<td>$q[q]$</td>
<td>$K$</td>
<td>$q$</td>
<td>_</td>
</tr>
<tr>
<td>propagate</td>
<td>0</td>
<td>$q, p[p \lor \neg q]$</td>
<td>$K$</td>
<td>$q$</td>
<td>_</td>
</tr>
<tr>
<td>propagate</td>
<td>0</td>
<td>$q, p, r[\neg p \lor r]$</td>
<td>$K$</td>
<td>$q$</td>
<td>_</td>
</tr>
<tr>
<td>conflict</td>
<td>0</td>
<td>$q, p, r$</td>
<td>$K$</td>
<td>$q$</td>
<td>$\neg q \lor \neg r$</td>
</tr>
</tbody>
</table>

A conflict at level 0 implies that the input clause set is unsatisfiable since there are no decision literals.
Language: In addition to propositional atoms, we add a set of constants \( \kappa \) given by \( c_0, c_1, \ldots \) and equalities \( c = d \) for constants \( c \) and \( d \).

\[
\phi ::= P \mid \neg \phi \mid \phi_1 \lor \phi_2 \mid \phi_1 \land \phi_2 \mid \kappa_1 = \kappa_2
\]

Interpretation: The structure \( M \) now has a domain \( |M| \) and maps propositional variables to \( \{\top, \bot\} \) and constants to \( |M| \).

\[
M[c = d] = \begin{cases} 
\top, & \text{if } M[c] = M[d] \\
\bot, & \text{otherwise}
\end{cases}
\]

Proof:

<table>
<thead>
<tr>
<th>Reflexivity</th>
<th>( \Gamma \vdash a = a, \Delta )</th>
</tr>
</thead>
</table>
| Symmetry    | \( \Gamma \vdash a = b, \Delta \)  \\
|             | \( \Gamma \vdash b = a, \Delta \)  |
| Transitivity| \( \Gamma \vdash a = b, \Delta \)  \\
|             | \( \Gamma \vdash b = c, \Delta \)  \\
|             | \( \Gamma \vdash a = c, \Delta \)  |
The logical state is a triple $\langle G; F; D \rangle$ with the input equalities and disequalities $G$, the processed disequalities $D$, and the find structure $F$ which is a set of oriented equalities, i.e., orient $y = x$ as $x = y$ if $y \prec x$.

<table>
<thead>
<tr>
<th></th>
<th>$x = y, G; F; D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>$\frac{G; F; D}{\text{if } F(x) \equiv F(y)}$</td>
</tr>
<tr>
<td>Merge</td>
<td>$x = y, G; F; D$</td>
</tr>
<tr>
<td></td>
<td>$\frac{G; F' \circ F; D}{\text{if } F(x) \not\equiv F(y)}$</td>
</tr>
<tr>
<td></td>
<td>$F' = {\text{orient}(F(x) = F(y))}$</td>
</tr>
<tr>
<td>Diseq</td>
<td>$x \not= y, G; F; D$</td>
</tr>
<tr>
<td></td>
<td>$\frac{G; F; x \not= y, D}{\text{if } F(x) = F(y)}$</td>
</tr>
<tr>
<td>Contrad</td>
<td>$G; F; x \not= y, D$</td>
</tr>
<tr>
<td></td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

- The above inference system is (strongly) conservative, progressive, and canonizing.
- Example: $x = y, y \not= z x = z, u = v; \emptyset; \emptyset$ reduces to $\bot$ through $\emptyset; x = z, y = z, u = v; y \not= z$. 

N. Shankar
Computation and Inference
First-Order Logic

**Language:** Signature $\Sigma$ lists allowable function and predicate symbols with their arities.

$$\tau := X \mid f(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0$$

$$\phi := \neg \phi \mid \phi_1 \lor \phi_2 \mid \phi_1 \land \phi_2 \mid \tau_1 = \tau_2 \mid \forall x.\phi \mid \exists x.\phi \mid q(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0$$

**Meaning:**

$$M[a = b] = M[a] = M[b]$$

$$M[f(a_1, \ldots, a_n)] = (M[f])(M[a_1], \ldots, M[a_n])$$

$$M[x]_\rho = \rho(x)$$

$$M[q(a_1, \ldots, a_n)]_\rho = M[q](M[a_1]_\rho, \ldots, M[a_n]_\rho)$$

$$M[\forall x. A]_\rho = \begin{cases} \top, & \text{if } M[A]_\rho[x := d] \text{ for all } d \in D \\ \bot, & \text{otherwise} \end{cases}$$

$$M[\exists x. A]_\rho = \begin{cases} \top, & \text{if } M[A]_\rho[x := d] \text{ for some } d \in D \\ \bot, & \text{otherwise} \end{cases}$$
A theory in a signature $\Sigma$ restricts the meaning of the function and predicate symbols.

For example, the array theory is given by operations $\text{select}$ and $\text{store}$ such that

1. $\text{select}(\text{store}(A, i, v), i) = v$
2. $i \neq j \Rightarrow \text{select}(\text{store}(A, i, v), j) = A(j)$

SMT deals with formulas with theory atoms like $x = y$, $x \neq y$, $x - y \leq 3$, and $\text{select}(\text{store}(A, i, v), j) = w$.

For SMT, the CDCL search state is augmented with a theory state $S$ in addition to the partial assignment.

When a literal is added to $M$ by unit propagation, it is also asserted to $S$.

When a literal is implied by $S$, it is propagated to $M$.

When backjumping, any literals deleted from $M$ are also retracted from $S$. 
The state extends CDCL with a find structure $F$ and disquality set $D$.

Input is $y = z$, $x = y \lor x = z$, $x \neq y \lor x \neq z$

<table>
<thead>
<tr>
<th>Step</th>
<th>$M$</th>
<th>$F$</th>
<th>$D$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assert</td>
<td>$y = z$</td>
<td>${y \leftrightarrow z}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Select</td>
<td>$y = z; x \neq y$</td>
<td>${y \leftrightarrow z}$</td>
<td>${x \neq y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Prop</td>
<td>$\ldots, x \neq z$</td>
<td>${y \leftrightarrow z}$</td>
<td>${x \neq y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td></td>
<td>$[x \neq z \lor y \neq z \lor x = y]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict</td>
<td>$\ldots$</td>
<td>${y \leftrightarrow z}$</td>
<td>${x \neq y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Analyze</td>
<td>$\ldots$</td>
<td>${y \leftrightarrow z}$</td>
<td>${x \neq y}$</td>
<td>${y \neq z$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\lor x = y}$</td>
</tr>
<tr>
<td>Bkjump</td>
<td>$y = z, x = y$</td>
<td>${y \leftrightarrow z}$</td>
<td>$\emptyset$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Assert</td>
<td>$y = z, x = y$</td>
<td>${x \leftrightarrow y, y \leftrightarrow z}$</td>
<td>$\emptyset$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Prop</td>
<td>$\ldots, x = z$</td>
<td>${x \leftrightarrow y, y \leftrightarrow z}$</td>
<td>$\emptyset$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td>$[x = z \lor x \neq y \lor y \neq z]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let \( \text{gcd}(x, y, z) \) represent the formula

\[
\begin{align*}
x &\geq 0 \land y \geq 0 \land x + y > 0 \land z > 0 \\
\land & z|x \land z|y \\
\land (\forall z' : z'|x \land z'|y \Rightarrow z' \leq z)
\end{align*}
\]

There is only one inference step:

\[
\frac{\text{gcd}(x, y, z)}{\text{gcd}((x \sqcup y) - (x \sqcap y), x \sqcap y, z)} \quad \text{if } x > 0, y > 0
\]

- **Conservative**, because any divisor of \( x, y \) such that \( x > y > 0 \) is also a divisor of \( x - y \).
- **Progressive**, because \( x + y \) decreases with each inference step.
- **Canonizing**, because if \( x = 0 \) or \( y = 0 \), then \( z = x \sqcup y \).
Given an array $A$ of $k$ integers, such an array is sorted if
$\forall i < j < k : (A(i) \not> A(j))$, i.e., there are no inversions.

The predicate $\text{sort}(A, B)$ holds if $B$ is a sorted, permutation of $A$.

The initial state is just the input array $a$, where $\Lambda(a)$ asserts $\text{sort}(a, B)$ for a result variable $B$.

The inference rule is

$$\frac{A}{\text{swap}(A, i, j) \quad i < j, A(i) > A(j)}$$

- **Conservative**: Each swap step preserves solvability.
- **Progressive**: Each swap is a decreasing step in a lexicographic ordering.
- **Canonizing**: When no swaps are possible, the array is already sorted.
Given a weighted directed graph $G = (V, E, W)$, with non-negative edge weights, find the smallest-weight path from a given source vertex $s$ to each vertex, i.e., a map $P_s$ on $V$:

$P_s(s) = 0$, and for $v \neq s$,

$P_s(v) = \bigwedge \{ P_s(u) + W(u, v) \mid u \in V \}$.

Let

$\text{post}(X)(v) = \begin{cases} 
0, & \text{if } v = s \\
\bigwedge \{ X(u) + W(u, v) \mid u \in \text{dom}(X) \}, & \text{otherwise.}
\end{cases}$

We therefore want to compute $P_s$ such that $P_s = \text{post}(P_s)$. 
The logical state has two partial maps $D$ and $Q$:

1. Each $v \in V$ is either in $\text{dom}(D)$ or $\text{dom}(Q)$, but not both,
2. $D(v) = \text{post}(D)(v)$ for $v \in \text{dom}(D)$,
3. $Q(v) = \text{post}(D)(v)$ for $v \in \text{dom}(Q)$, and
4. $D(u) \leq Q(v)$ for $u \in \text{dom}(D)$ and $v \in \text{dom}(Q)$.

Initially, $D = [s \mapsto 0]$, and

$$Q = \begin{cases} W(s, v), & \text{if } \langle s, v \rangle \in E \\
\infty, & \text{otherwise}\end{cases} \quad \left| \begin{array}{c} \text{if } v \neq s \end{array} \right|.$$

Each inference step has the form

$$\frac{\langle D, Q \rangle}{\langle D', Q' \rangle}$$

where $u = \text{arg } \text{min}_u Q(u)$, $D' = D[u \mapsto Q(u)]$, and $Q' = [v \mapsto Q(v) \cap (Q(u) + W(u, v)) \mid v \in \text{dom}(Q) - \{u\}]$. 
Conclusions

- Computing is abstraction engineering.
- The world is full of useful abstractions.
- Deductive inference is both a foundational medium for understanding computation, and a powerful and practical tool for analyzing them.
- Algorithms operate on these abstractions through inference steps, where

  \[ \text{Algorithm} = \text{Inference} + \text{Strategy} + \text{Indexing} \]

- Tools based on inference algorithms\(^1\) are making a significant impact across a range of fields.
- For students, this is a golden age with a confluence of theory, tools, and applications creating bountiful research opportunities.

\(^1\)Abduction (e.g., interpolation) and induction (e.g., synthesis, machine learning) are also important.