Plan for today

1. Recap example showing the integration of backtracking search, E-graph, and matching heuristic
2. Decision procedures

Cross-cutting aspects
- Proof-system search ($\vdash \vdash$)
- Interpretation search ($\models \models$)
- Equality
- Induction
- Quantifiers
- Decision procedures

Plan for rest of the quarter

- Topics still left:
  - Search in the proof domain: 2 lectures
  - Equality part II (rewrite systems): 1 lecture
  - Constructive logic: 1 lecture
  - Automating induction: 1 lecture

A recap example

```define fact hasConstValue(X:Var,C:Const)
with meaning [X == C]

if currStmt == [X = C]
then hasConstValue(X,C)@out

if hasConstValue(X,C)@in
currStmt == [Y = X]
then mustPointTo(Y,C)@out

if hasConstantValue(X,C)@in
currStmt == [X = Y]
then transform to [X = C]
```

VC for the trans rule

- Show that if statements $x := y$ and $x := c$ are executed in a state where $y == c$, then the resulting states are the same.

```if hasConstantValue(X,C)@in A
currStmt == [X = Y]
then transform to [X = C]
```

VC for the trans rule

- Show that if statements $x := y$ and $x := c$ are executed in a state where $y == c$, then the resulting states are the same.

```
\forall x,y,c,\sigma. \quad \sigma[y] = c \Rightarrow
\forall v. \quad \text{step}(x := y, \sigma)[v] = \text{step}(x := c, \sigma)[v]
```

Background axioms

- If $a := k$ gets stepped in store $\sigma$, the resulting store is $\sigma$ with “a” updated to “k”.
- If $a := b$ gets stepped in store $\sigma$, the resulting store is $\sigma$ with “a” updated to the value of “b”.

```
\forall x,y,c,\sigma. \quad \sigma[y] = c \Rightarrow
\forall v. \quad \text{step}(x := y, \sigma)[v] = \text{step}(x := c, \sigma)[v]
```
Background axioms

- Axioms:
  \[ \forall a,k,\sigma \cdot \text{step}(a := k, \sigma) = \text{store}(\sigma, a, k) \]
  \[ \forall a,b,\sigma \cdot \text{step}(a := b, \sigma) = \text{store}(\sigma, a, \sigma[b]) \]
- Show:
  \[ \forall x,y,c,\sigma. \quad \sigma[y] = c \Rightarrow \]
  \[ \forall v. \text{step}(x := y, \sigma)[v] = \text{step}(x := c, \sigma)[v] \]

Expand

- Axioms:
  \[ \forall a,k,\sigma \cdot \text{step}(a := k, \sigma) = \text{store}(\sigma, a, k) \]
  \[ \forall a,b,\sigma \cdot \text{step}(a := b, \sigma) = \text{store}(\sigma, a, \sigma[b]) \]
- Show:
  \[ \forall x,y,c,\sigma. \quad \sigma[y] = c \Rightarrow \]
  \[ \forall v. \text{step}(x := y, \sigma)[v] = \text{step}(x := c, \sigma)[v] \]

Expand

\[ \forall x,y,c,\sigma. \quad \sigma[y] = c \Rightarrow \]
\[ \forall v. \text{step}(x := y, \sigma)[v] = \text{step}(x := c, \sigma)[v] \]

Skolemize

\[ \neg \sigma[y] = c \lor \]
\[ \forall v. \text{step}(x := y, \sigma)[v] = \text{step}(x := c, \sigma)[v] \]

Refutation

\[ \neg \sigma[y] = c \lor \]
\[ \{ \text{step}(x := y, \sigma)[v] = \text{step}(x := c, \sigma)[v] \} \]

Negate formula and show that the negation is unsatisfiable
Refutation

Negate formula and show that the negation is unsatisfiable

$$\sigma[y] = c \land \text{step}(x := y, \sigma)[v] \neq \text{step}(x := c, \sigma)[v]$$

Exhaustive interpretation search

$$L_1 \land L_2$$

Two ways to refute:
* Formula becomes trivially false
* Set of assumed literals is inconsistent

Equality using E-graph

$$L_1 \triangleq \sigma[y] = c$$
$$L_2 \triangleq \text{step}(x := y, \sigma)[v] \neq \text{step}(x := c, \sigma)[v]$$
Equality using E-graph

\[ \sigma[y] = c \]
\[ \text{step}(x \leftarrow y, \sigma)[v] \neq \text{step}(x \leftarrow c, \sigma)[v] \]
\[ \sigma[y] \quad \vdash \quad c \]

Matching

\[ \forall a, k, \sigma. \text{step}(a \leftarrow k, \sigma) = \text{store}(\sigma, a, k) \]

- Pick a trigger
- If trigger appears in E-graph, instantiate quantifier body

\[ \sigma[y] \quad \vdash \quad c \]
Matching

\[ \forall a, b, \sigma \cdot \text{step}(a := b, \sigma) = \text{store}(\sigma, a, \sigma[b]) \]

- Pick a trigger
- If trigger appears in E-graph, instantiate quantifier body

\[
\text{step}(x := y, \sigma) = \text{store}(\sigma, x, \sigma[y])
\]

\[ \sigma[y] \rightarrow c \]

In summary, add:

\[
\text{step}(x := c, \sigma) = \text{store}(\sigma, x, c)
\]

\[
\text{step}(x := y, \sigma) = \text{store}(\sigma, x, \sigma[y])
\]

\[ \sigma[y] \rightarrow c \]

In summary, add:

\[
\text{step}(x := c, \sigma) = \text{store}(\sigma, x, c)
\]

\[ \sigma \rightarrow x \rightarrow \sigma[y] \rightarrow c \]

In summary, add:

\[
\text{step}(x := c, \sigma) = \text{store}(\sigma, x, c)
\]

\[
\text{step}(x := y, \sigma) = \text{store}(\sigma, x, \sigma[y])
\]

\[ \sigma \rightarrow x \rightarrow \sigma[y] \rightarrow c \]

In summary, add:

\[
\text{step}(x := c, \sigma) = \text{store}(\sigma, x, c)
\]

\[
\text{step}(x := y, \sigma) = \text{store}(\sigma, x, \sigma[y])
\]

\[ \sigma \rightarrow x \rightarrow \sigma[y] \rightarrow c \]
In summary, add:

\[ \text{step}(x := c, \sigma) = \text{store}(\sigma, x, c) \]
\[ \text{step}(x := y, \sigma) = \text{store}(\sigma, x, \sigma[y]) \]

\[ L_1 \land L_2 \]
\[ L_1 \triangleq \sigma[y] = c \]
\[ L_2 \triangleq \text{step}(x := y, \sigma)[v] \neq \text{step}(x := c, \sigma)[v] \]

Decision procedures

- Decision procedures are complete algorithms for determining the validity of a formula in a given logic
- Decision procedures exist for many logics
  - EUF
  - Theory of lists
  - Theory of arrays
  - Theory of linear arithmetic over reals or integers
  - Theory of bit-vectors
  - ...

But we are more concerned with how decision procedures can be used within the context of a “heuristic” theorem prover

- A heuristic theorem prover is a theorem prover for an undecidable logic that uses heuristics to guide its search
- We use the term “heuristic” to avoid confusion between the larger heuristic prover and the decision procedures that are being integrated into this larger prover
## Decision procedures

- Why incorporate decision procedures into a heuristic prover?
- Because once the search reaches a formula in a decidable subset of the original logic, the strategies of the heuristic prover may be inefficient and incomplete

## Issues

- There are two issues to consider when incorporating decision procedures into a heuristic prover
  - Communication between decision procedures and the heuristic prover
  - Communication between decision procedures

## In Simplify--

- Communication between decision procedures
  - Don’t have to deal with this, because Simplify-- has only one decision procedure, namely EUF

## Issues again

- Communication between decision procedure and the heuristic prover
  - We’ve seen how this works in Simplify--
- Communication between decision procedures
  - This is what’s next

## In Simplify--

- Communication form heuristic prover to decision procedures

- Communication from decision procedures to the heuristic prover
  - Communication from decision procedures to the heuristic prover
    - Push equalities into the E-graph incrementally
      - Does not require the decision procedure to expose its internal details
  - Matching heuristic looks into E-graph
    - Motivation is to improve the heuristic of the prover
    - For efficiency, expose details of the decision procedure’s data structures
    - Explicating proofs used to guide the backtracking search
      - Motivation is efficiency
Combining decision procedures

- Efficient decision procedures exist for many decidable logics, but some formulas do not belong to any of these logics
- Instead, they belong to a combination of these logics
- For example:

```
if currStmt == [X = Y]
then geq(X,Y)@out
```

Nelson-Oppen example

• $x \leq y \land y \leq x + \text{car}(\text{cons}(0, x)) \land P(h(x) - h(y)) \land \neg P(0)$

```
<table>
<thead>
<tr>
<th>$x \leq y$</th>
<th>$y \leq x + \text{car}(\text{cons}(0, x))$</th>
<th>$P(h(x) - h(y))$</th>
<th>$\neg P(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_R$</td>
<td>$F_E$</td>
<td>$F_E$</td>
<td>$F_E$</td>
</tr>
<tr>
<td>$x \leq y$</td>
<td>$y \leq x + \text{car}(\text{cons}(0, x))$</td>
<td>$P(h(x) - h(y))$</td>
<td>$\neg P(0)$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$x = y$</td>
<td>$x + y$</td>
<td>$x + y$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$x = y$</td>
<td>$x + y$</td>
<td>$x + y$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$x = y$</td>
<td>$x + y$</td>
<td>$x + y$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$x = y$</td>
<td>$x + y$</td>
<td>$x + y$</td>
</tr>
</tbody>
</table>
```

Correctness

- If a contradiction is found, return UNSAT
  - This is clearly sound, if each decision procedure is sound
- If there are no more equalities to be found by any of the decision procedures, return SAT
  - Is this complete? Have the decision procedures exchanged enough info?
  - Each decision procedure has found its own satisfying assignments, but how do we know that these satisfying assignments are compatible (ie: don’t contradict each other)

Convex theories

- A theory is convex if whenever a satisfiable conjunction of literals entails a disjunction of equalities of variables, then it entails one of the equalities
  - Example:
    - Theory of linear arithmetic with equalities
- For convex theories:
  - If no equalities can be found, then it is impossible for there to be a disjunction of equalities that can be found; therefore, no missed equalities

Nonconvex theories

- Example:
  - Reals under multiplication
    - $xy = 0 \land z = 0$ entails $x = 0 \lor y = 0$
  - Integers under $+$ and $\leq$
    - $x + 1 \land y = 2 \land z \leq 2$ entails $x + z = y$
- Theory of sets
- Theory of arrays
- For such theories, must perform a case split when a disjunction of equalities is entailed
  - Try each disjunct recursively.
  - If any one returns SAT, return SAT
  - If all disjuncts return UNSAT, return UNSAT
Algorithm

- Given a formula $F$ that is a conjunction of literals over theories $S$ and $T$, returns whether $F$ is SAT or UNSAT
  1. Assign conjunctions to $F_S$ and $F_T$ so that $F_S$ is a conjunction of $S$-literals and $F_T$ is a conjunction of $T$-literals
  2. If either $F_S$ or $F_T$ is unsatisfiable, return UNSAT
  3. If either $F_S$ or $F_T$ entails some equality between variables not entailed by the other, then add the equality as a new conjunct to the one that does not entail it. Goto step 2.
  4. If either $F_S$ or $F_T$ entails a disjunction $x_1 \lor \ldots x_k$ of equalities between variables, then for each $i$ from 1 to $k$, apply the procedure recursively to $F_S \land F_T \land x_i$. If any recursive call returns SAT, return SAT. Otherwise return UNSAT.
  5. Return SAT

Adding Nelson-Oppen to Simplify--

- Each decision procedure keeps track of its own information
- Decision procedure for theory $T$ exports a function $\text{assert}(F)$, where $F$ is a literal in $T$
- While performing the backtracking search, if a literal is asserted, add that literal (using assert) to the decision procedure for the theory the literal belongs to
  - If the literal belongs to a combination of theories, split the literal into a conjunction of literals, each one belonging to only one theory

Example

- $xy = 0 \land z = 0 \land f(f(x) - f(z)) \neq f(z) \land f(f(y) - f(z)) \neq f(z)$

Example

- $xy = 0 \land z = 0 \land f(f(x) - f(z)) \neq f(z) \land f(f(y) - f(z)) \neq f(z)$