Review

- Finite Models
- Size of Models
- Theories
- Interpretations Between Theories

Outline

- Combining Decision Procedures
- Example Application: Translation Validation

Sources:


Combining Decision Procedures

Often, verification conditions are expressed in a language which mixes several theories.

A natural question is whether one can use decision procedures for individual theories to construct a decision procedure for the union theory.

More precisely, suppose that $\Sigma_1, \ldots, \Sigma_n$ are $n$ signatures, and for $i = 1, \ldots, n$, let $T_i$ be a $\Sigma_i$-theory.

Then, let $\text{Sat}_i$ be a decision procedure for deciding the $T_i$-satisfiability of $\Sigma_i$-formulas.

How can we use these to construct a decision procedure for the $T$-satisfiability of $\Sigma$-formulas, where $T = Cn \bigcup T_i$ and $\Sigma = \bigcup \Sigma_i$.
The Nelson-Oppen Method

A very general method for combining decision procedures is the Nelson-Oppen method.

This method is applicable when
1. The signatures $\Sigma_i$ are disjoint.
2. The theories $T_i$ are stably-infinite.
   A $\Sigma$-theory $T$ is stably-infinite if every $T$-satisfiable quantifier-free $\Sigma$-formula is satisfiable in an infinite model.
3. The formulas to be tested for satisfiability are quantifier-free.

In practice, only the third requirement is a significant restriction.

We may also restrict our attention to conjunctions of literals.

This is because any quantifier-free formula can be put into disjunctive normal form. It then suffices to check the satisfiability of each conjunction.

The Nelson-Oppen Method

Before explaining the procedure in detail, we need the following definitions.
1. For $i = 1, \ldots, n$, a member of $\Sigma_i$ is an $i$-symbol.
2. A $\Sigma$-term $t$ is an $i$-term if it is a variable, a constant $i$-symbol, or the application of a functional $i$-symbol.
3. An $i$-predicate is an application of a predicate $i$-symbol.
4. An atomic $i$-formula is an $i$-predicate or an equation whose left hand side is an $i$-term (for equations whose left-hand-sides are variables, we arbitrarily choose a theory $T_i$ to associate with each variable).
5. An $i$-literal is an atomic $i$-formula or the negation of an atomic $i$-formula.
6. An occurrence of a term $t$ in either a term or a formula is $i$-alien if $t$ is a $j$-term with $i \neq j$ and all of its super-terms (if any) are $i$-terms.
7. An $i$-term or $i$-literal is pure if it contains only $i$-symbols.

The Nelson-Oppen Method

Now we can explain step one of the Nelson-Oppen method:

1. Conversion to Separate Form

Given a conjunction of literals, $\phi$, we desire to convert it into a separate form: a $T$-equisatisfiable conjunction of literals $\phi_1 \land \phi_2 \land \ldots \land \phi_n$, where each $\phi_i$ is a $\Sigma_i$-formula.

The following algorithm accomplishes this.

1. Let $\psi$ be some $i$-literal in $\phi$.
2. If $\psi$ is a pure $i$-literal, for some $i$, remove $\psi$ from $\phi$ and add $\psi$ to $\phi_i$; if $\phi$ is empty then stop; otherwise goto step 1.
3. Let $t$ be an $i$-alien term in $\psi$. Replace $t$ in $\phi$ with a new variable $z$ associated with theory $T_i$, and add $z = t$ to $\phi$. Goto step 1.

The Nelson-Oppen Method

It is easy to see that $\phi$ is $T$-satisfiable iff $\phi_1 \land \ldots \land \phi_n$ is $T$-satisfiable.

Furthermore, because each $\phi_i$ is a $\Sigma_i$-formula, we can run $\text{Sat}_i$ on each $\phi_i$.

Clearly, if $\text{Sat}_i$ reports that any $\phi_i$ is unsatisfiable, then $\phi$ is satisfiable.

But the converse is not true in general.

We need a way for the decision procedures to communicate with each other about shared variables.

First a definition: If $S$ is a set of terms and $\sim$ is an equivalence relation on $S$, then the arrangement of $S$ induced by $\sim$ is $\text{Ar}_\sim = \{ x = y \mid x \sim y \} \cup \{ x \neq y \mid x \not\sim y \}$.
The Nelson-Oppen Method

Suppose that $T_1$ and $T_2$ are theories with disjoint signatures $\Sigma_1$ and $\Sigma_2$ respectively. Let $T = T_1 \cup T_2$ and $\Sigma = \bigcup \Sigma_i$. Given a $\Sigma$-formula $\phi$ and decision procedures $Sat_1$ and $Sat_2$ for $T_1$ and $T_2$ respectively, we wish to determine if $\phi$ is $T$-satisfiable. The non-deterministic Nelson-Oppen algorithm for this is as follows:

1. Convert $\phi$ to its separate form $\phi_1 \land \phi_2$.
2. Let $S$ be the set of variables shared between $\phi_1$ and $\phi_2$. Guess an equivalence relation $\sim$ on $S$.
3. Run $Sat_1$ on $\phi_1 \cup A \sim$. Run $Sat_2$ on $\phi_2 \cup A \sim$.
4. If there exists an equivalence relation $\sim$ such that both $Sat_1$ and $Sat_2$ succeed, then we claim that $\phi$ is $T$-satisfiable.
5. If no such equivalence relation exists, then we claim that $\phi$ is $T$-unsatisfiable.

The generalization to more than two theories is straightforward.

Correctness of Nelson-Oppen

We define an interpretation of a signature $\Sigma$ to be a model of $\Sigma$ together with a variable assignment.

Two interpretations $A$ and $B$ are isomorphic if there exists an isomorphism $h$ of the model of $A$ into the model of $B$ and $h(x^A) = x^B$ for each variable $x$ (where $x^A$ signifies the value assigned to $x$ by the variable assignment of $A$).

We furthermore define $A^{\Sigma_i, V}$ to be the restriction of $A$ to the symbols in $\Sigma_i$ and the variables in $V$.

Theorem

Let $\Sigma_1$ and $\Sigma_2$ be signatures, and for $i = 1, 2$, let $\phi_i$ be a set of $\Sigma_i$-formulas, and $V_i$ the set of variables appearing in $\phi_i$. Then $\phi_1 \cup \phi_2$ is satisfiable if and only if there exists a $\Sigma_1$-interpretation $A$ satisfying $\phi_1$ and a $\Sigma_2$-interpretation $B$ satisfying $\phi_2$ such that:

$A^{\Sigma_1 \cap \Sigma_2, V_1 \cap V_2}$ is isomorphic to $B^{\Sigma_1 \cap \Sigma_2, V_1 \cap V_2}$.

Example

Consider the combination of the theory $T_2$ with the theory $T_{E}$ of equality.

Let $\phi = 1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2)$.

Is this satisfiable? No.

To determine this using the above algorithm, we first convert $\phi$ to a separate form:

$\phi_2 = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2$
$\phi_E = f(x) \neq f(y) \land f(x) \neq f(z)$

Now, the shared variables are $\{x, y, z\}$. There are 5 possible arrangements based on equivalence classes of $x, y,$ and $z$:

1. $\{x = y, x = z, y = z\}$: inconsistent with $\phi_E$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_E$.
3. $\{x \neq y, x = z, y \neq z\}$: inconsistent with $\phi_E$.
4. $\{x \neq y, x \neq z, y = z\}$: inconsistent with $\phi_2$.
5. $\{x \neq y, x \neq z, y \neq z\}$: inconsistent with $\phi_2$.

Correctness of Nelson-Oppen

Proof

Let $\Sigma = \Sigma_1 \cap \Sigma_2$ and $V = V_1 \cap V_2$.

Suppose $\phi_1 \cup \phi_2$ is satisfiable. Let $M$ be an interpretation satisfying $\phi_1 \cup \phi_2$. If we let $A = M^{\Sigma_1, V_1}$ and $B = M^{\Sigma_2, V_2}$, then clearly

- $A \models \phi_1$
- $B \models \phi_2$
- $A^{\Sigma, V}$ is isomorphic to $B^{\Sigma, V}$

On the other hand, suppose that we have $A$ and $B$ satisfying the three conditions listed above. Let $h$ be an isomorphism from $A^{\Sigma, V}$ to $B^{\Sigma, V}$.

We define an interpretation $M$ as follows:

- $dom(M) = dom(A)$
- For each variable or constant $u$, $u^M = \begin{cases} u^A & \text{if } u \in (\Sigma_1 \in V_1) \\ h^{-1}(u^B) & \text{otherwise} \end{cases}$
Correctness of Nelson-Oppen

- For function symbols of arity \(n\),
  \[
  f^M(a_1, \ldots, a_n) = \begin{cases} 
  f^A(a_1, \ldots, a_n) & \text{if } f \in \Sigma^F_1 \\
  h^{-1}(f^B(h(a_1), \ldots, h(a_n))) & \text{otherwise}
  \end{cases}
  \]

- For predicate symbols of arity \(n\),
  \[
  (a_1, \ldots, a_n) \in P^M \iff (a_1, \ldots, a_n) \in P^A \quad \text{if } P \in \Sigma^P_1 \\
  (a_1, \ldots, a_n) \in P^M \iff (h(a_1), \ldots, h(a_n)) \in P^B \quad \text{otherwise}
  \]

By construction, \(M^{\Sigma_1, V_1}\) is isomorphic to \(A\). In addition, it is easy to verify that \(h\) is an isomorphism of \(M^{\Sigma_2, V_2}\) to \(B\).

It follows by the homomorphism theorem that \(M\) satisfies \(\phi_1 \cup \phi_2\).

\[ \square \]

Correctness of Nelson-Oppen

To show that \(h\) is surjective, let \(b \in V^B\). Then there exists a variable \(x \in V^A\) such that \(x^B = b\). But then \(h(x^A) = b\).

Since \(h\) is bijective, it follows that \(|V^A| = |V^B|\), and since \(|A| = |B|\), we also have that \(|A - V^A| = |B - V^B|\). We can therefore extend \(h\) to a bijective function \(h'\) from \(A\) to \(B\).

By construction, \(h'\) is an isomorphism of \(A^V\) to \(B^V\). Thus, by the previous theorem, we can obtain an interpretation satisfying \(\phi_1 \cup \phi_2\).

\[ \square \]

Correctness of Nelson-Oppen

Theorem

Let \(\Sigma_1\) and \(\Sigma_2\) be signatures, with \(\Sigma_1 \cap \Sigma_2 = \emptyset\), and for \(i = 1, 2\), let \(\phi_i\) be a set of \(\Sigma_i\)-formulas, and \(V_i\) the set of variables appearing in \(\phi_i\). As before, let \(V = V_1 \cap V_2\). Then \(\phi_1 \cup \phi_2\) is satisfiable iff there exists an interpretation \(A\) satisfying \(\phi_1\) and an interpretation \(B\) satisfying \(\phi_2\) such that:

1. \(|M_1| = |M_2|\), and
2. \(x^A = y^A\) iff \(x^B = y^B\) for every pair of variables \(x, y \in V\).

Proof

Clearly, if \(\phi_1 \cup \phi_2\) is satisfiable in some interpretation \(M\), then the only if direction holds by letting \(A = M\) and \(B = M\).

Consider the converse. Let \(h : V^A \rightarrow V^B\) be defined as \(h(x^A) = x^B\). This definition is well-formed by property 2 above.

In fact, \(h\) is bijective. To show that \(h\) is injective, let \(h(a_1) = h(a_2)\). Then there exist variables \(x, y \in V\) such that \(a_1 = x^A, a_2 = y^A,\) and \(x^B = y^B\). By property 2, \(x^A = y^A\), and therefore \(a_1 = a_2\).

Correctness of Nelson-Oppen

We can finally prove the correctness of the nondeterministic Nelson-Oppen method.

Theorem

Let \(T_i\) be a stably-infinite \(\Sigma_i\)-theory, for \(i = 1, 2\), and suppose that \(\Sigma_1 \cap \Sigma_2 = \emptyset\). Also, let \(\phi_i\) be a set of \(\Sigma_i\) literals, \(i = 1, 2\), and let \(S\) be the set of variables appearing in both \(\phi_1\) and \(\phi_2\). Then \(\phi_1 \cup \phi_2\) is \(T_1 \cup T_2\)-satisfiable iff there exists an equivalence relation \(\sim\) on \(S\) such that \(\phi_i \cup Ar_{\sim}\) is \(T_i\)-satisfiable, \(i = 1, 2\).

Proof

Suppose \(M\) is an interpretation satisfying \(\phi_1 \cup \phi_2\). We define an equivalence relation \(x \sim y\) iff \(x, y \in S\) and \(x^M = y^M\). By construction, \(M\) is a \(T_i\)-interpretation satisfying \(\phi_i \cup Ar_{\sim}, i = 1, 2\).
Correctness of Nelson-Oppen

Suppose on the other hand that there exists an equivalence relation \( \sim \) of \( S \) such that \( \phi_i \cup A_{r,\sim} \) is \( T_i \)-satisfiable, \( i = 1, 2 \). Since \( T_1 \) is stably-infinite, there is an infinite interpretation \( A \) satisfying \( \phi_1 \cup A_{r,\sim} \). Similarly, there is an infinite interpretation \( B \) satisfying \( \phi_2 \cup A_{r,\sim} \).

But by LST, we can take the least upper bound of \( |A| \) and \( |B| \) and obtain interpretations of that cardinality.

Then we have \( |A| = |B| \) and \( x^A = y^A \) iff \( x^B = y^B \) for every variable \( x, y \in S \). We can thus apply the previous theorem and obtain the existence of a \((\Sigma_1 \cup \Sigma_2)\)-interpretation satisfying \( \phi_1 \cup \phi_2 \).

\( \square \)

Translation Validation

Ultimate Goal

- Guarantee correctness of optimizing compilers

Important in:

- Safety critical applications, where standards and regulations require that every compiler be certified
- Compilation into silicon, where a translation error is critically expensive

Translator vs. Translation Validation

Rather than verify the translator itself, verify the results of each run of the translator.

Advantages

- Much easier
- Less sensitive to changes in the translator

Drawback

- Additional overhead during compilation
- But not enough to outweigh the benefits

Translation Validation

Two main types of optimizations

- Structure preserving optimizations
- Structure modifying optimizations

Structure preserving

- Use Validate proof rule
Validate Proof Rule

To verify that $T$ correctly translates $S$, establish:

- **control abstraction** $\kappa$ from $T$’s basic blocks to $S$’s basic blocks
- **data abstraction**
  \[
  \alpha : PC = \kappa(pc) \land (p_1 \rightarrow V_1 = e_1) \land \cdots \land (p_n \rightarrow V_n = e_n)
  \]
- **invariant** $\phi_i$ for each block $B$ referring only to concrete variables.
- **Verification Conditions**: For each pair of basic blocks $i$ and $j$, verify
  \[
  C_{ij} : \phi_i \land \alpha \land \rho_{ij}^T \land \left( \bigvee_{\pi \in \text{Paths}^S} \rho_\pi \right) \rightarrow \alpha' \land \phi'_j,
  \]

where $\text{Paths}^S$ is the set of all simple source paths and $\rho_\pi$ is the transition relation for the simple source path $\pi$.

---

**Example: INTSQRT**

<table>
<thead>
<tr>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B0 :$ N:=500; Y:=0; W:=1;</td>
<td>$b0 :$ t:=0; y:=0; w:=1;</td>
</tr>
<tr>
<td>$B1 :$ if !(N \geq W) goto B3;</td>
<td>$b1 :$ {\phi_1 : t = 2y}</td>
</tr>
<tr>
<td>$B2 :$ W:=W+2*Y+3; Y:=Y+1;</td>
<td>w:=t + w +3; y:=y+1; t:=t+2;</td>
</tr>
<tr>
<td>goto B1;</td>
<td>if (w &lt; 500) goto b1;</td>
</tr>
<tr>
<td>$B3 :$</td>
<td>$b2 :$</td>
</tr>
</tbody>
</table>

Control abstraction $\kappa$:
\[ b0 \mapsto B0 \]
\[ b1 \mapsto B2 \]
\[ b2 \mapsto B3 \]

Data abstraction:
\[ (PC = \kappa(pc) \land (Y = y) \land (W = w) \land (pc \neq b0 \rightarrow N = 500) \]

---

**Reordering Transformations**

Important class of structure-modifying transformations.

Transformation is a simple permutation of the original execution order.

---

**Example: Loop Reversal**

B(1) \[ \implies \] B(n)  
B(2) \[ \implies \] B(n - 1)  
\[ : \] \[ : \]  
B(n) \[ \implies \] B(1)
Type I Loop Transformations

\[ \text{for } \vec{i} \in \mathcal{I} \text{ by } \prec_I \text{ do } B(\vec{i}) \implies \text{for } \vec{j} \in \mathcal{J} \text{ by } \prec_J \text{ do } B(F(\vec{j})) \]

Example: Reversal

- \( \mathcal{I} = \mathcal{J} = \{1..n\} \)
- \( F(j) = n - j + 1 \)

Example: Loop Interchange

- \( \mathcal{I} = \{1..m\} \times \{1..n\} \)
- \( \mathcal{J} = \{1..n\} \times \{1..m\} \)
- \( F(j_1, j_2) = (j_2, j_1) \)

Type II Loop Transformations (Loop Fusion)

\[ \text{for } \vec{i} \in \mathcal{I} \text{ by } \prec_I \text{ do } \]
\[ B_1(\vec{i}) \]
\[ \implies \text{for } \vec{i} \in \mathcal{I} \text{ by } \prec_I \text{ do } \]
\[ B_2(\vec{i}) \]

\[ B_1(1) \]
\[ \vdots \]
\[ B_1(n) \]
\[ B_2(1) \]
\[ \vdots \]
\[ B_2(n) \]

Proof Rule for Type I Loop Transformations

\[ \vec{i}_1 \prec_I \vec{i}_2 \land F^{-1}(\vec{i}_2) \prec_J F^{-1}(\vec{i}_1) \implies \]
\[ B(\vec{i}_1); B(\vec{i}_2) \sim B(\vec{i}_2); B(\vec{i}_1) \]

\[ \text{for } \vec{i} \in \mathcal{I} \text{ by } \prec_I \text{ do } B(\vec{i}) \sim \text{for } \vec{j} \in \mathcal{J} \text{ by } \prec_J \text{ do } B(F(\vec{j})) \]

Proof Rule for Type II Loop Transformations

\[ \vec{i}_1 \prec_I \vec{i}_2 \implies \]
\[ B_1(\vec{i}_1); B_2(\vec{i}_1) \sim B_2(\vec{i}_1); B_1(\vec{i}_1) \]

\[ \text{for } \vec{i} \in \mathcal{I} \text{ by } \prec_I \text{ do } B_1(\vec{i}) \sim \text{for } \vec{i} \in \mathcal{I} \text{ by } \prec_I \text{ do } B_1(\vec{i}); B_2(\vec{i}) \]
Speculative Optimizations

- Optimizations which only apply under certain conditions
- Require a *run-time* test to check the condition

Example

\[
\begin{align*}
&\text{for } i = 1 \text{ to } M \\
&\quad \text{for } j = 1 \text{ to } N \\
&\quad A[i, j] := A[i - 1, j - k]
\end{align*}
\]

Speculative Optimizations

Where do run-time tests come from?

- Hard-coded into compiler
- Dangerous potential source of compiler bugs

Can they be automatically generated?

- Use translation validation infrastructure
- Find necessary conditions under which validation fails
- Use these conditions to derive run-time test
- Tests are correct by construction

Checking Verification Conditions with CVC

Example

\[
\begin{align*}
&\text{for } i = 1 \text{ to } M \\
&\quad \text{for } j = 1 \text{ to } N \\
&\quad A[i, j] := A[i - 1, j - k]
\end{align*}
\]

Verification Condition

\[
(i_1, j_1) < (i_2, j_2) \land (j_2, i_2) < (j_1, i_1) \\
\leadsto
\]

\[
\]
Deriving Run-Time Tests with CVC

Input: Verification Condition $\phi$
Output: Run-Time Test $\psi$

1. Let $\psi = true$
2. Check $\psi \rightarrow \phi$
3. If valid, return $\psi$
4. If invalid, get a counterexample $\theta$
5. Select a formula from $\theta$, negate it, and add it (via conjunction) to $\psi$
6. Goto 2

Deriving Run-Time Tests with CVC

Example

\[
\begin{align*}
\text{for } i & = 1 \text{ to } M \\
\text{for } j & = 1 \text{ to } N \\
\Rightarrow \\
\text{for } j & = 1 \text{ to } N \\
\text{for } i & = 1 \text{ to } M \\
A[i, j] & := A[i - 1, j - k]
\end{align*}
\]

Verification Condition

\[
\begin{align*}
(i_1, j_1) < (i_2, j_2) \land (j_2, i_2) < (j_1, i_1) \\
&\Rightarrow \\
&\sim \\
A[i_2, j_2] & := A[i_2 - 1, j_2 - k] ; A[i_1, j_1] := A[i_1 - 1, j_1 - k]
\end{align*}
\]

Deriving Run-Time Tests with CVC

Formula Selection Heuristics

- Must constrain a testable variable
- Prefer positive assertions to negated assertions
- Prefer smaller (simpler) formula to larger formula

Deriving Run-Time Tests with CVC

More Interesting Example

procedure copy($p$, $r$, $N$)
begin
\[
\text{for } i = 0 \text{ to } N - 1 \\
* (p + i) := * (r + i)
\]
end
\[
\ldots
\]
\[
\text{copy}(p, r, N) \\
\text{copy}(q, r, N)
\]
Deriving Run-Time Tests with CVC

After Inlining

\[
\text{for } i = 0 \text{ to } N - 1 \\
\quad \ast(p + i) \defeq \ast(r + i) \\
\text{for } i = 0 \text{ to } N - 1 \\
\quad \ast(q + i) \defeq \ast(r + i)
\]

Perfect Candidate for Fusion

\[
\text{for } i = 0 \text{ to } N - 1 \\
\quad \ast(p + i) \defeq \ast(r + i) \\
\quad \ast(q + i) \defeq \ast(r + i)
\]

Deriving Run-Time Tests with CVC

Fusion Example

\[
\text{for } i = 0 \text{ to } N - 1 \\
\quad \ast(p + i) \defeq \ast(r + i) \\
\quad \ast(q + i) \defeq \ast(r + i) \\
\implies \\
\text{for } i = 0 \text{ to } N - 1 \\
\quad \ast(p + i) \defeq \ast(r + i) \\
\quad \ast(q + i) \defeq \ast(r + i)
\]

Verification Condition

\[
i_1 < i_2 \implies \\
\ast(p + i_2) \defeq \ast(r + i_2) ; \ast(q + i_1) \defeq \ast(r + i_1) \\
\sim \\
\ast(q + i_1) \defeq \ast(r + i_1) ; \ast(p + i_2) \defeq \ast(r + i_2)
\]

Deriving Run-Time Tests with CVC

Fusion Example

\[
\begin{align*}
\text{if } & ((q - r \leq 0 \text{ OR } q - r \geq N) \text{ AND} \\
& (q - p \leq 0 \text{ OR } q - p \geq N) \text{ AND} \\
& (r - p \leq 0 \text{ OR } r - p \geq N)) \\
\text{for } & i = 0 \text{ to } N - 1 \\
\quad & \ast(p + i) \defeq \ast(r + i) \\
\quad & \ast(q + i) \defeq \ast(r + i) \\
\implies \\
\text{else } \\
\text{for } & i = 0 \text{ to } N - 1 \\
\quad & \ast(p + i) \defeq \ast(r + i) \\
\text{for } & i = 0 \text{ to } N - 1 \\
\quad & \ast(q + i) \defeq \ast(r + i)
\end{align*}
\]