Review

- Theories
- Congruence Closure
- 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 $T = Cn \cup \bigcup_i T_i$ and $\Sigma = \bigcup_i \Sigma_i$?
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.

---

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.

---

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 $\psi_1 \land \psi_2 \land \ldots \land \psi_n$, where each $\psi_i$ 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 $\phi$. Replace $t$ in $\phi$ with a new variable $z$ associated with theory $T_i$, and add $z = t$ to $\phi_i$. Goto step 1.

---

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 unsatisfiable.

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

$$\text{Arr}_\sim = \{x = y \mid x \sim y\} \cup \{x \neq y \mid x \not\sim y\}.$$
The Nelson-Oppen Method

Suppose that \( T_1 \) and \( T_2 \) are theories with disjoint signatures \( \Sigma_1 \) and \( \Sigma_2 \) respectively. Let \( T = C \cap T_1 \cup T_2 \) and \( \Sigma = \bigcup \Sigma_i \). Given a \( \Sigma \)-formula \( \phi \) and decision procedures \( \text{Sat}_1 \) and \( \text{Sat}_2 \) 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 \( \text{Sat}_1 \) on \( \phi_1 \cup \mathcal{A} \sim \).
4. Run \( \text{Sat}_2 \) on \( \phi_2 \cup \mathcal{A} \sim \).

If there exists an equivalence relation \( \sim \) such that both \( \text{Sat}_1 \) and \( \text{Sat}_2 \) succeed, then we claim that \( \phi \) is \( T \)-satisfiable.

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. If \( \mathcal{A} \) is an interpretation, we write \( \mathcal{A} \models \phi \) to mean that \( \phi \) is satisfied by the model and variable assignment contained in \( \mathcal{A} \).

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

We furthermore define \( \mathcal{A}^{\Sigma_i, V} \) to be the restriction of \( \mathcal{A} \) 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 there exists a \( \Sigma_1 \)-interpretation \( \mathcal{A} \) satisfying \( \phi_1 \) and a \( \Sigma_2 \)-interpretation \( \mathcal{B} \) satisfying \( \phi_2 \) such that:

\[ \mathcal{A}^{\Sigma_1 \cap \Sigma_2, V_1 \cap V_2} \text{ is isomorphic to } \mathcal{B}^{\Sigma_1 \cap \Sigma_2, V_1 \cap V_2}. \]

Example

Consider the combination of the theory \( T_Z \) 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? \textbf{No}.

To determine this using the above algorithm, we first convert \( \phi \) to a separate form:
\[ \phi_Z = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2 \]
\[ \phi_E = f(x) \neq f(y) \land f(x) \neq f(z) \]

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 \( \phi_Z \).
5. \( \{x \neq y, x \neq z, y \neq z\} \): inconsistent with \( \phi_Z \).

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 \( \mathcal{M} \) be an interpretation satisfying \( \phi_1 \cup \phi_2 \). If we let \( \mathcal{A} = M^{\Sigma_1, V_1} \) and \( \mathcal{B} = M^{\Sigma_2, V_2} \), then clearly

\[ \mathcal{A} \models \phi_1 \]
\[ \mathcal{B} \models \phi_2 \]
\[ \mathcal{A}^{\Sigma_i, V} \text{ is isomorphic to } \mathcal{B}^{\Sigma_i, V} \]

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

We define an interpretation \( \mathcal{M} \) as follows:

\[ \text{dom}(\mathcal{M}) = \text{dom}(\mathcal{A}) \]

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

- For function symbols of arity $n$,
  \[
  f^M(a_1, \ldots, a_n) = \begin{cases} 
  f^A(a_1, \ldots, a_n) & \text{if } f \in \Sigma_1^F \\
  h^{-1}(f^B(h(a_1), \ldots, h(a_n))) & \text{otherwise}
  \end{cases}
  \]

- For predicate symbols of arity $n$,
  \[
  (a_1, \ldots, a_n) \in P^M \iff (a_1, \ldots, a_n) \in P^A \text{ if } P \in \Sigma_1^P \\
  (a_1, \ldots, a_n) \in P^M \iff (h(a_1), \ldots, h(a_n)) \in P^B \text{ otherwise}
  \]

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 $\phi_1 \cup \phi_2$.

\[\Box\]

Correctness of Nelson-Oppen

To show that $h$ is surjective, let $b \in V^B$. Then there exists a variable $x \in V^B$ 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 $\phi_1 \cup \phi_2$.

\[\Box\]

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. $|A| = |B|$, 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 $a_1 = a_2$.

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 $\phi_1 \cup A\phi_{\sim}$ 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 $\phi_i \cup A\phi_{\sim}, i = 1, 2$. 

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 $\phi_1 \cup A\phi_{\sim}$ 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 $\phi_i \cup A\phi_{\sim}, i = 1, 2$. 

13

14

15

16
Correctness of Nelson-Oppen

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

But by LST, we can take the least upper bound of $\hat{A}$ and $\hat{B}$ and obtain interpretations of that cardinality.

Then we have $|\hat{A}| = |\hat{B}|$ and $x^\hat{A} = y^\hat{A}$ iff $x^\hat{B} = y^\hat{B}$ 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

Two main types of optimizations

- Structure preserving optimizations
- Structure modifying optimizations

Structure preserving

- Use Validate proof rule
Validate Proof Rule

To verify that a target \( T \) correctly translates a source \( S \), establish:

- **control abstraction** \( \kappa \) from \( T \)'s basic blocks to \( S \)'s basic blocks
- **data abstraction** \( \alpha \) specifying source variables in terms of target expressions

\[
\alpha : PC = \kappa(pc) \land (p_1 \to V_1 = e_1) \land \cdots \land (p_n \to V_n = e_n)
\]

- invariant \( \phi_i \) for each block \( B \) referring only to target variables.
- **Verification Conditions**: For each pair of basic blocks \( i \) and \( j \), verify

\[
C_{ij} : \phi_i \land \alpha \land \rho_{ij}^T \land (\bigvee_{\pi \in \text{Paths}^S} \rho_{\pi}) \land \alpha' \land \phi_j',
\]

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**

**before**

- \( B0 : \) N:=500; Y:=0; W:=1;
- \( B1 : \) if !(N \geq W) goto B3;
- \( B2 : \) W:=W+2*Y+3; Y:=Y+1; goto B1;
- \( B3 : \) if (w < 500) goto b1;

**after**

- \( b0 : \) t:=0; y:=0; w:=1;
- \( b1 : \) \{\( \phi_1 : t=2y \)\}
- \( b2 : \) \{\( \phi_1 : t=2y \)\}

Control abstraction \( \kappa : \)

\[
b0 \to B0
\]

Data abstraction:

\[
( PC = \kappa(pc) \land (Y = y) \land (W = w) \land (pc \neq b0 \to N = 500))
\]

**CVC Input**

\[
PC', Y', W' \land \text{query} \land \text{assertions}
\]

\[
\text{assert} (pc = 0) \quad \text{then} \quad 0
\]

\[
\text{elsif} (pc = 1) \quad \text{then} \quad 2
\]

\[
\text{else} \quad 3 \quad \text{endif}
\]

\[
Y = y \quad \text{and} \quad W = w \quad \text{and} \quad (pc / 0) \Rightarrow (N = 500)
\]

\[
\text{assert} (pc = 0) \quad \text{and} \quad pc' = 1 \quad \text{and} \quad t' = 0 \quad \text{and} \quad y' = 0 \quad \text{and} \quad w' = 1
\]

\[
\text{assert} (pc = 0) \quad \text{and} \quad pc' = 2 \quad \text{and} \quad N' = 500 \quad \text{and} \quad Y' = 0 \quad \text{and} \quad W' = 1 \quad \text{and} \quad (N' \geq W')
\]

\[
\text{query} (pc' = 0) \quad \text{then} \quad 0
\]

\[
\text{elsif} (pc' = 1) \quad \text{then} \quad 2
\]

\[
\text{else} \quad 3 \quad \text{endif}
\]

\[
Y' = y' \quad \text{and} \quad W' = w' \quad \text{and} \quad ((pc' / 0) \Rightarrow (N' = 500)) \quad \text{and} \quad t' = 2 \cdot y'
\]

\[
\text{query} (pc' = 0) \quad \text{then} \quad 0
\]

\[
\text{elsif} (pc' = 1) \quad \text{then} \quad 2
\]

\[
\text{else} \quad 3 \quad \text{endif}
\]

\[
Y' = y' \quad \text{and} \quad W' = w' \quad \text{and} \quad ((pc' / 0) \Rightarrow (N' = 500)) \quad \text{and} \quad t' = 2 \cdot y'
\]
Reordering Transformations

Important class of \textit{structure modifying} transformations.

Transformation is a simple \textit{permutation} of the original execution order.

Example: Loop Reversal

\begin{align*}
B(1) & \quad B(n) \\
B(2) & \quad B(n-1) \\
\vdots & \quad \vdots \\
B(n) & \quad B(1)
\end{align*}

Example: Loop Interchange

\begin{align*}
I & = J = \{1..n\} \\
F(j) & = n - j + 1
\end{align*}

Loop Transformations: Loop Fusion

\begin{align*}
& \text{for } i \in I \text{ by } \prec_I \text{ do } B_1(i) \\
& \text{for } i \in I \text{ by } \prec_I \text{ do } B_2(i)
\end{align*}

\begin{align*}
& \text{for } i \in I \text{ by } \prec_I \text{ do } B_1(i) \\
& \text{for } i \in I \text{ by } \prec_I \text{ do } B_2(i)
\end{align*}

\begin{align*}
& \text{for } i \in I \text{ by } \prec_I \text{ do } B_1(i) \\
& \text{for } i \in I \text{ by } \prec_I \text{ do } B_2(i)
\end{align*}

\begin{align*}
B_1(1) & \quad B_1(1) \\
\vdots & \quad \vdots \\
B_1(n) & \quad B_1(n) \\
B_2(1) & \quad B_2(n)
\end{align*}

\begin{align*}
B_1(1) & \quad B_2(1) \\
\vdots & \quad \vdots \\
B_1(n) & \quad B_2(n)
\end{align*}

Loop Transformations

\begin{align*}
& \text{for } i \in I \text{ by } \prec_I \text{ do } B(i) \implies \text{for } j \in J \text{ by } \prec_J \text{ do } B(F(j))
\end{align*}

Example: Loop Fusion

\begin{align*}
I & = \{1..2\} \times \{1..m\} \times \{1\} \\
J & = \{1\} \times \{1..m\} \times \{1..2\} \\
F(1, j, b) & = (b, j, 1)
\end{align*}
**Permute Proof Rule**

\[
\begin{align*}
\vec{i}_1 \ll \vec{i}_2 \land F^{-1}(\vec{i}_2) \ll F^{-1}(\vec{i}_1) \\
\implies B(\vec{i}_1) \sim B(\vec{i}_2) \sim B(F(\vec{i}_1))
\end{align*}
\]

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

---

**Example**

\[
\begin{align*}
\text{for } i = 1 \text{ to } M \\
\quad \text{for } j = 1 \text{ to } N \\
\implies \text{for } i = 1 \text{ to } M \\
\quad \text{for } j = 1 \text{ to } N \\
\end{align*}
\]

**Verification Condition**

\[
\begin{align*}
(i_1, j_1) < (i_2, j_2) \land (j_2, j_1) < (i_1, i_1) \\
\end{align*}
\]

---

**Speculative Optimizations**

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

**Example**

\[
\begin{align*}
\text{for } i = 1 \text{ to } M \\
\quad \text{for } j = 1 \text{ to } N \\
\implies \text{for } i = 1 \text{ to } M \\
\quad \text{for } j = 1 \text{ to } N \\
\end{align*}
\]

---

**CVC Input**

\[
i_1, j_1, i_2, j_2, \text{arb_addr} : \text{INT};
\]

\[
a : \text{ARRAY INT OF ARRAY INT OF INT};
\]

\[
\text{QUERY} \\
\quad ((i_1 < i_2 \text{ OR } (i_1 = i_2 \text{ AND } j_1 < j_2)) \text{ AND } \quad \\
\quad (j_2 < j_1 \text{ OR } (j_2 = j_1 \text{ AND } i_2 < i_1)) \rightarrow \\
\quad ((\text{LET } a_1 : \text{ARRAY INT OF ARRAY INT OF INT} = \\
\quad a \text{ WITH } [i_1][j_1] := a[i_1-1][j_1-1] \text{ IN } \\
\quad a_1 \text{ WITH } [i_2][j_2] := a_1[i_2-1][j_2-1][\text{arb_addr}] = \\
\quad (\text{LET } a_1 : \text{ARRAY INT OF ARRAY INT OF INT} = \\
\quad a \text{ WITH } [i_2][j_2] := a[i_2-1][j_2-1] \text{ IN } \\
\quad a_1 \text{ WITH } [i_1][j_1] := a_1[i_1-1][j_1-1][\text{arb_addr}]);
\]

---

**Permute Proof Rule**

\[
\text{Example}
\]

---
Speculative Optimizations

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

Example

\[
\text{for } i = 1 \text{ to } M \\
\text{for } j = 1 \text{ to } N \\
\]

\[
\text{if } k \geq 0 \\
\text{for } j = 1 \text{ to } N \\
\text{for } i = 1 \text{ to } M \\
\]

\[
\text{else} \\
\text{for } i = 1 \text{ to } M \\
\text{for } j = 1 \text{ to } N \\
\]

Deriving Run-Time Tests with CVC

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

1. Let \( \psi = \text{true} \)
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

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

Deriving Run-Time Tests with CVC

Formula Selection Heuristics

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

Loop variables can be eliminated using known inequalities
Deriving Run-Time Tests with CVC

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

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

Verification Condition
\[
\begin{align*}
&(i_1, j_1) < (i_2, j_2) \land (j_2, i_2) < (j_1, i_1) \\
&\quad \implies A[i_1, j_1] := A[i_1 - 1, j_1 - k] \\
&\quad A[i_2, j_2] := A[i_2 - 1, j_2 - k] \\
&\quad A[i_2, j_2] := A[i_2 - 1, j_2 - k] \\
&\quad A[i_1, j_1] := A[i_1 - 1, j_1 - k]
\end{align*}
\]

CVC Output
InValid.
Current stack level is 0 (scope 7).
% Active assumptions:
ASSERT (arb_addr = i2);
ASSERT ((1 + (-1 * i2) + i1) = 0);
ASSERT ((0 + k + (-1 * j2) + j1) = 0);
ASSERT NOT (j2 = j1);
Only the third assertion meets our criteria.
Negating gives the condition: \( k \neq j2 - j1 \).
Using the known inequality \( j2 - j1 < 0 \) results in the run-time test: \( k \geq 0 \).

CVC Input
\[
i_1, j_1, i_2, j_2, k, \text{ arb_addr : INT};
a : \text{ARRAY INT OF ARRAY INT OF INT};
\]
QUERY ((i1 < i2 OR (i1 = i2 AND j1 < j2)) AND
\[
(j2 < j1 OR (j2 = j1 AND i2 < i1))) \Rightarrow
\[
((\text{LET } a1 : \text{ARRAY INT OF ARRAY INT OF INT} =
\quad a \text{ WITH } [i2][j2] := a[i2-1][j2-k][\text{arb_addr}] =
\text{LET } a1 : \text{ARRAY INT OF ARRAY INT OF INT} =
\quad a \text{ WITH } [i2][j2] := a[i2-1][j2-k] \text{ IN}
\quad a \text{ WITH } [i1][j1] := a[i1-1][j1-k][\text{arb_addr}]));
\]

Deriving Run-Time Tests with CVC

More Interesting Example
procedure copy\( (p, r, N) \)
begin
for \( i = 0 \) to \( N - 1 \)
\[
*(p + i) := *(r + i)
\]
end
... copy\( (p, r, N) \)
copy\( (q, r, N) \)
**Deriving Run-Time Tests with CVC**

**After Inlining**

\[
\begin{align*}
& \text{for } i = 0 \text{ to } N - 1 \\
& \quad (p + i) := (r + i) \\
& \text{for } i = 0 \text{ to } N - 1 \\
& \quad (q + i) := (r + i)
\end{align*}
\]

**Perfect Candidate for Fusion**

\[
\begin{align*}
& \text{for } i = 0 \text{ to } N - 1 \\
& \quad (p + i) := (r + i) \\
& \quad (q + i) := (r + i)
\end{align*}
\]

**CVC Input**

\[
\begin{align*}
p, q, r & : \text{INT} \\
i_1, i_2, \text{arb_addr} & : \text{INT} \\
M & : \text{ARRAY INT OF INT} \\
\text{QUERY} & \\
(i_1 < i_2) & => \\
(\text{LET } M1 & : \text{ARRAY INT OF INT} = \\
M & \text{WITH } [q + i_1] := M[r + i_1] \text{ IN} \\
M1 & \text{WITH } [p + i_2] := M1[r + i_2])\{\text{arb_addr}\} = \\
(\text{LET } M1 & : \text{ARRAY INT OF INT} = \\
M & \text{WITH } [p + i_2] := M[r + i_2] \text{ IN} \\
M1 & \text{WITH } [q + i_1] := M1[r + i_1])\{\text{arb_addr}\};
\end{align*}
\]

**CVC Output**

We initially get a counter-example which includes the assertion:

\[
q - r = i_2 - i_1
\]

Asserting its negation, we get another counter-example with the assertion:

\[
r - p = i_2 - i_1
\]

Repeating this one more time yields:

\[
q - p = i_2 - i_1.
\]

Under the negation of these three assertions, the verification condition is valid.

Using the inequality \(0 < i_2 \quad i_1 < N\), we get the run-time test:

\[
\begin{align*}
&q - r \leq 0 \text{ OR } q - r \geq N \text{ AND} \\
&q - p \leq 0 \text{ OR } q - p \geq N \text{ AND} \\
r - p \leq 0 & \text{ OR } r - p \geq N
\end{align*}
\]
Fusion Example

if \((q - r \leq 0 \text{ OR } q - r \geq N) \text{ AND} \)
\((q - p \leq 0 \text{ OR } q - p \geq N) \text{ AND} \)
\((r - p \leq 0 \text{ OR } r - p \geq N)\)

for \(i = 0 \text{ to } N - 1\)
\((p + i) := (r + i)\)
\((q + i) := (r + i)\)

else

for \(i = 0 \text{ to } N - 1\)
\((p + i) := (r + i)\)
\((q + i) := (r + i)\)