G22.2390-001 Logic in Computer Science
Fall 2009

Lecture 9

Review

- Theories
- Satisfiablity Modulo Theories
- Congruence Closure
- Shostak's Method

Outline

- Satisfiability Modulo Theories
- Theory Solvers
- Combining Decision Procedures
- Abstract DPLL Modulo Theories
- Example Application: Translation Validation

Sources:

Section 2.7 of Enderton.

C. Tinelli and M. Harandi. *A New Correctness Proof of the Nelson-Oppen Combination Procedure*. FroCoS '96.

Z. Manna and C. Zarba. *Combining Decision Procedures*. Draft available from http://theory.stanford.edu/∼zm/new-papers.html.

C. Barrett. *Checking Validity of Quantifier-Free Formulas in Combinations of First-Order Theories*. PhD Dissertation. Stanford University, 2003.

C. Barrett, B. Goldberg, and L. Zuck. *Run-Time Validation of Speculative Optimizations using CVC.* In *Third International Workshop on Run-time Verification (RV)*.

Motivation

Automatic analysis of computer hardware and software requires *engines* capable of reasoning efficiently about large and complex systems.

Boolean engines such as *Binary Decision Diagrams* and *SAT solvers* are typical engines of choice for today's industrial verification applications.

However, systems are usually designed and modeled at a higher level than the Boolean level and the translation to Boolean logic can be expensive.

A primary goal of research in *Satisfiability Modulo Theories* (SMT) is to create verification engines that can reason natively at a higher level of abstraction, while still retaining the speed and automation of today's Boolean engines.

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 Σ_i-theory.

Then, let Sat_i be a decision procedure for deciding the T_i-satisfiability of Σ_i-formulas.

How can we use these to construct a decision procedure for the T-satisfiability of Σ-formulas, where $T = \mathit{Cn} \bigcup T_i$ 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 Σ_i are disjoint.
2. The theories T_i are stably-infinite.

   A Σ-theory T is *stably-infinite* if every T-satisfiable quantifier-free Σ-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 Σ_i is an i-*symbol*.
2. A Σ-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, ϕ, we desire to convert it into a *separate form*: a T-equisatisfiable conjunction of literals $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$, where each ϕ_i is a Σ_i-formula.

The following algorithm accomplishes this.

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

The Nelson-Oppen Method

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

Furthermore, because each ϕ_i is a Σ_i-formula, we can run *Sat* $_i$ on each ϕ_i.

Clearly, if *Sat* $_i$ reports that any ϕ_i is unsatisfiable, then ϕ 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
$Ar_{\sim} = \{x = y \mid x \sim y\} \cup \{x \neq y \mid x \not\sim y\}$.

The Nelson-Oppen Method

Suppose that T_1 and T_2 are theories with disjoint signatures Σ_1 and Σ_2 respectively. Let $T = \mathit{Cn} \bigcup T_i$ and $\Sigma = \bigcup \Sigma_i$. Given a Σ-formula ϕ and decision procedures *Sat* $_1$ and *Sat* $_2$ for T_1 and T_2 respectively, we wish to determine if ϕ is T-satisfiable. The non-deterministic Nelson-Oppen algorithm for this is as follows:

1. Convert ϕ to its separate form $\phi_1 \wedge \phi_2$.
2. Let S be the set of variables shared between ϕ_1 and ϕ_2. Guess an equivalence relation $\sim$ on S.
3. Run *Sat* $_1$ on $\phi_1 \cup Ar_\sim$.
4. Run *Sat* $_2$ on $\phi_2 \cup Ar_\sim$.

If there exists an equivalence relation $\sim$ such that both *Sat* $_1$ and *Sat* $_2$ succeed, then we claim that ϕ is T-satisfiable.

If no such equivalence relation exists, then we claim that ϕ is t-unsatisfiable.

The generalization to more than two theories is straightforward.

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable?

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable? **No.**

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable? **No.**

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

$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$
$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$

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

1. $\{x = y, x = z, y = z\}$
2. $\{x = y, x \neq z, y \neq z\}$
3. $\{x \neq y, x = z, y \neq z\}$
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable? **No.**

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

$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$
$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$

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 $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$
3. $\{x \neq y, x = z, y \neq z\}$
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable? **No.**

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

$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$
$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$

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 $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable? **No.**

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

$$\begin{aligned}
\phi_{\mathcal{Z}} &= 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2 \\
\phi_{\mathcal{E}} &= f(x) \neq f(y) \wedge f(x) \neq f(z)
\end{aligned}$$

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 $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable? **No.**

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

$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$
$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$

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 $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
4. $\{x \neq y, x \neq z, y = z\}$: inconsistent with $\phi_{\mathcal{Z}}$.
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

Consider the combination of the theory $T_{\mathcal{Z}}$ with the theory $T_{\mathcal{E}}$ of equality.

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

Is this satisfiable? **No.**

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

$$\begin{aligned}
\phi_{\mathcal{Z}} &= 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2 \\
\phi_{\mathcal{E}} &= f(x) \neq f(y) \wedge f(x) \neq f(z)
\end{aligned}$$

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 $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
4. $\{x \neq y, x \neq z, y = z\}$: inconsistent with $\phi_{\mathcal{Z}}$.
5. $\{x \neq y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{Z}}$.

Correctness of Nelson-Oppen

We define an *interpretation* of a signature Σ to be a model of Σ together with a variable assignment. If A is an interpretation, we write $A \models \phi$ to mean that ϕ is satisfied by the model and variable assignment contained in A.

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 $A^{\Sigma,V}$ to be the restriction of A to the symbols in Σ and the variables in V.

Theorem

Let Σ_1 and Σ_2 be signatures, and for $i = 1, 2$, let ϕ_i be a set of Σ_i-formulas, and V_i the set of variables appearing in ϕ_i. Then $\phi_1 \cup \phi_2$ is satisfiable iff there exists a Σ_1-interpretation A satisfying ϕ_1 and a Σ_2-interpretation B satisfying ϕ_2 such that:

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

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:

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

Correctness of Nelson-Oppen

- For function symbols of arity n,

$$f^M(a_1,\ldots,a_n) = \begin{cases} f^A(a_1,\ldots,a_n) & \text{if } f \in \Sigma_1^F \\ h^{-1}(f^B(h(a_1),\ldots,h(a_n))) & \text{otherwise} \end{cases}$$

- For predicate symbols of arity n,

$$(a_1,\ldots,a_n) \in P^M \text{ iff } (a_1,\ldots,a_n) \in P^A \text{ if } P \in \Sigma_1^P$$
$$(a_1,\ldots,a_n) \in P^M \text{ iff } (h(a_1),\ldots,h(a_n)) \in P^B \text{ otherwise}$$

By construction, M^{Σ_1,V_1} is isomorphic to A. In addition, it is easy to verify that h is an isomorphism of M^{Σ_2,V_2} to B.

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

$\square$

Correctness of Nelson-Oppen

Theorem

Let Σ_1 and Σ_2 be signatures, with $\Sigma_1 \cap \Sigma_2 = \emptyset$, and for $i = 1, 2$, let ϕ_i be a set of Σ_i-formulas, and V_i the set of variables appearing in ϕ_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 ϕ_1 and an interpretation B satisfying ϕ_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 $h : V^A \to V^B$ 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

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 $\phi_1 \cup \phi_2$.

$\square$

Correctness of Nelson-Oppen

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

Theorem

Let T_i be a stably-infinite Σ_i-theory, for $i = 1, 2$, and suppose that $\Sigma_1 \cap \Sigma_2 = \emptyset$. Also, let ϕ_i be a set of Σ_i literals, $i = 1, 2$, and let S be the set of variables appearing in both ϕ_1 and ϕ_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 $\phi_i \cup Ar_\sim$ is T_i-satisfiable, $i = 1, 2$. Since T_1 is stably-infinite, there is an infinite interpretation A satisfying $\phi_1 \cup Ar_\sim$. Similarly, there is an infinite interpretation B satisfying $\phi_2 \cup Ar_\sim$.

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

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 a target T correctly translates a source S, establish:

- *control abstraction* κ from T's basic blocks to S's basic blocks
- *data abstraction* α specifying source variables in terms of target expressions

$$\alpha : PC = \kappa(pc) \wedge (p_1 \rightarrow V_1 = e_1) \wedge \cdots \wedge (p_n \rightarrow V_n = e_n)$$

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

$$C_{ij}: \qquad \phi_i \wedge \alpha \wedge \rho^T_{ij} \wedge (\bigvee_{\pi \in \mathrm{Paths}^S} \rho_\pi) \quad \rightarrow \quad \alpha' \wedge \phi'_j,$$

where Paths^S is the set of all simple source paths and ρ_π is the transition relation for the simple source path π.

Example: INTSQRT

before		after	
$B0:$	N:=500; Y:=0; W:=1;	$b0:$	t:=0; y:=0; w:=1;
$B1:$	**if** !(N $\geq$ W) **goto** B3;	$b1:$	$\{\phi_1 : t = 2y\}$
$B2:$	W:=W+2*Y+3; Y:=Y+1;		w:=t + w +3; y:= y+1; t:= t+2;
	goto B1;		**if** (w $<$ 500) **goto** b1;
$B3:$		$b2:$	

Control abstraction κ:

$$b0 \mapsto B0$$
$$b1 \mapsto B2$$
$$b2 \mapsto B3$$

Data abstraction:

$$(PC = \kappa(pc) \wedge (Y = y) \wedge (W = w) \wedge$$
$$(pc \neq b0 \rightarrow N = 500)$$

Example: INTSQRT

before		after	
$B0:$	N:=500; Y:=0; W:=1;	$b0:$	t:=0; y:=0; w:=1;
$B1:$	**if** !(N $\geq$ W) **goto** B3;	$b1:$	$\{\phi_1 : t = 2y\}$
$B2:$	W:=W+2*Y+3; Y:=Y+1;		w:=t + w +3; y:= y+1; t:= t+2;
	goto B1;		**if** (w $<$ 500) **goto** b1;
$B3:$		$b2:$	

C_{01}: $\phi_0 \wedge \alpha \wedge \rho_{01}^T \wedge (\bigvee_{\pi \in \text{Paths}^S} \rho_\pi) \quad \rightarrow \quad \alpha' \wedge \phi_1'$ expands to:

$$
true \wedge \begin{pmatrix} & PC = \kappa(pc) \\ \wedge & Y = y \\ \wedge & W = w \\ \wedge & pc \neq b0 \rightarrow N = 500 \end{pmatrix} \wedge \begin{pmatrix} & pc = b0 \\ \wedge & pc' = b1 \\ \wedge & t' = 0 \\ \wedge & y' = 0 \\ \wedge & w' = 1 \end{pmatrix} \wedge \begin{pmatrix} & PC = B0 \\ \wedge & PC' = B2 \\ \wedge & N' = 500 \\ \wedge & Y' = 0 \\ \wedge & W' = 1 \\ \wedge & N' \geq W' \end{pmatrix}
$$

$$
\longrightarrow \begin{pmatrix} & PC' = \kappa(pc') \wedge Y' = y' \\ \wedge & W' = w' \wedge pc' \neq b0 \rightarrow N' = 500 \end{pmatrix} \wedge (t' = 2 \cdot y')
$$

CVC Input

```
PC′,y′,Y′,w′,W′,N′,pc,PC,y,Y,w,W,N,pc′,t′:INT;
ASSERT (PC = IF (pc = 0) THEN 0
             ELSIF (pc = 1) THEN 2
             ELSE 3 ENDIF) AND
        Y=y AND W=w AND ((pc /= 0) => (N = 500));
ASSERT pc=0 AND pc′=1 AND t′=0 AND y′=0 AND w′=1;
ASSERT PC=0 AND PC′=2 AND N′=500 AND Y′=0 AND W′=1 AND (N′>=W′);
QUERY (PC′ = IF (pc′ = 0) THEN 0
             ELSIF (pc′ = 1) THEN 2
             ELSE 3 ENDIF) AND
       Y′=y′ AND W′=w′ AND ((pc′ /= 0) => (N′ = 500)) AND
       t′ = 2 * y′;
```

Reordering Transformations

Important class of *structure modifying* transformations.

Transformation is a simple *permutation* of the original execution order.

Example: Loop Reversal

$$
\begin{array}{ccc}
\mathtt{B}(1) & & \mathtt{B}(n) \\
\mathtt{B}(2) & & \mathtt{B}(n-1) \\
\vdots & \Longrightarrow & \vdots \\
\mathtt{B}(n) & & \mathtt{B}(1)
\end{array}
$$

Loop Transformations

$$\textbf{for } \vec{i} \in \mathcal{I} \textbf{ by } \prec_{\mathcal{I}} \textbf{ do } \mathtt{B}(\vec{i}) \quad \Longrightarrow \quad \textbf{for } \vec{j} \in \mathcal{J} \textbf{ by } \prec_{\mathcal{J}} \textbf{ do } \mathtt{B}(F(\vec{j}))$$

Example: Reversal

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

Example: Loop Interchange

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

Loop Transformations: Loop Fusion

for $\vec{i} \in \mathcal{I}$ **by** $\prec_{\mathcal{I}}$ **do**

$\quad \mathrm{B}_1(i)$

for $\vec{i} \in \mathcal{I}$ **by** $\prec_{\mathcal{I}}$ **do**

$\quad \mathrm{B}_2(i)$

$$\Longrightarrow$$

for $\vec{i} \in \mathcal{I}$ **by** $\prec_{\mathcal{I}}$ **do**

$\quad \mathrm{B}_1(i)$

$\quad \mathrm{B}_2(i)$

Loop Transformations: Loop Fusion

for $\vec{i} \in \mathcal{I}$ **by** $\prec_{\mathcal{I}}$ **do**
  $\mathrm{B}_1(i)$

for $\vec{i} \in \mathcal{I}$ **by** $\prec_{\mathcal{I}}$ **do**
  $\mathrm{B}_2(i)$

$\Longrightarrow$

for $\vec{i} \in \mathcal{I}$ **by** $\prec_{\mathcal{I}}$ **do**
  $\mathrm{B}_1(i)$
  $\mathrm{B}_2(i)$

$$\begin{array}{c} \mathrm{B}_1(1) \\ \vdots \\ \mathrm{B}_1(n) \\ \mathrm{B}_2(1) \\ \vdots \\ \mathrm{B}_2(n) \end{array} \Longrightarrow \begin{array}{c} \mathrm{B}_1(1) \\ \mathrm{B}_2(1) \\ \vdots \\ \mathrm{B}_1(n) \\ \mathrm{B}_2(n) \end{array}$$

Loop Transformations

$$\textbf{for } \vec{i} \in \mathcal{I} \textbf{ by } \prec_{\mathcal{I}} \textbf{ do } \mathtt{B}(\vec{i}) \quad \Longrightarrow \quad \textbf{for } \vec{j} \in \mathcal{J} \textbf{ by } \prec_{\mathcal{J}} \textbf{ do } \mathtt{B}(F(\vec{j}))$$

Example: Loop Fusion

- $\mathcal{I} = \{1..2\} \times \{1..m\} \times \{1\}$
- $\mathcal{J} = \{1\} \times \{1..m\} \times \{1..2\}$
- $F(1, j, b) = (b, j, 1)$

Permute Proof Rule

$$\vec{i}_1 \prec_{\mathcal{I}} \vec{i}_2 \wedge F^{-1}(\vec{i}_2) \prec_{\mathcal{J}} F^{-1}(\vec{i}_1)$$

$$\longrightarrow$$

$$\mathtt{B}(\vec{i}_1); \mathtt{B}(\vec{i}_2) \;\sim\; \mathtt{B}(\vec{i}_2); \mathtt{B}(\vec{i}_1)$$

---

for $\vec{i} \in \mathcal{I}$ **by** $\prec_{\mathcal{I}}$ **do** $\mathtt{B}(\vec{i})$ $\quad\sim\quad$ **for** $\vec{j} \in \mathcal{J}$ **by** $\prec_{\mathcal{J}}$ **do** $\mathtt{B}(F(\vec{j}))$

Permute Proof Rule

Example

for $i = 1$ **to** M
  for $j = 1$ **to** N
    $A[i, j]$ **:=** $A[i - 1, j - 1]$

$$\Longrightarrow$$

for $j = 1$ **to** N
  for $i = 1$ **to** M
    $A[i, j]$ **:=** $A[i - 1, j - 1]$

Permute Proof Rule

Example

for $i = 1$ **to** M
  for $j = 1$ **to** N
    $A[i, j]$ **:=** $A[i - 1, j - 1]$

$\Longrightarrow$

for $j = 1$ **to** N
  for $i = 1$ **to** M
    $A[i, j]$ **:=** $A[i - 1, j - 1]$

Verification Condition

$$(i_1, j_1) < (i_2, j_2) \quad \wedge \quad (j_2, i_2) < (j_1, i_1)$$

$$\longrightarrow$$

$$A[i_1, j_1] \text{ := } A[i_1 - 1, j_1 - 1] \text{ ; } A[i_2, j_2] \text{ := } A[i_2 - 1, j_2 - 1]$$

$$\sim$$

$$A[i_2, j_2] \text{ := } A[i_2 - 1, j_2 - 1] \text{ ; } A[i_1, j_1] \text{ := } A[i_1 - 1, j_1 - 1]$$

CVC Input

```
i1, j1, i2, j2, arb_addr : INT;

a : 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))) =>

((LET a1 : ARRAY INT OF ARRAY INT OF INT =

    a WITH [i1][j1] := a[i1-1][j1-1] IN

    a1 WITH [i2][j2] := a1[i2-1][j2-1])[arb_addr] =

 (LET a1 : ARRAY INT OF ARRAY INT OF INT =

    a WITH [i2][j2] := a[i2-1][j2-1] IN

    a1 WITH [i1][j1] := a1[i1-1][j1-1])[arb_addr]);
```

Speculative Optimizations

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

Speculative Optimizations

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

Example

for $i = 1$ **to** M
  for $j = 1$ **to** N
    $A[i, j]$ **:=** $A[i - 1, j - k]$

$\Longrightarrow$

for $j = 1$ **to** N
  for $i = 1$ **to** M
    $A[i, j]$ **:=** $A[i - 1, j - k]$

Speculative Optimizations

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

Example

for $i = 1$ **to** M
  for $j = 1$ **to** N
    $A[i, j]$ **:=** $A[i - 1, j - k]$

$\Longrightarrow$

if $k \geq 0$
  for $j = 1$ **to** N
    for $i = 1$ **to** M
      $A[i, j]$ **:=** $A[i - 1, j - k]$
else
  for $i = 1$ **to** M
    for $j = 1$ **to** N
      $A[i, j]$ **:=** $A[i - 1, j - k]$

Where do run-time tests come from?

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

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

Input: Verification Condition ϕ

Output: Run-Time Test ψ

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

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

for $i = 1$ **to** M
  for $j = 1$ **to** N
    $A[i, j]$ **:=** $A[i - 1, j - k]$

$\Longrightarrow$

for $j = 1$ **to** N
  for $i = 1$ **to** M
    $A[i, j]$ **:=** $A[i - 1, j - k]$

Verification Condition

$$(i_1, j_1) < (i_2, j_2) \quad \wedge \quad (j_2, i_2) < (j_1, i_1)$$

$$\longrightarrow$$

$$A[i_1, j_1] \text{ := } A[i_1 - 1, j_1 - k] \text{ ; } A[i_2, j_2] \text{ := } A[i_2 - 1, j_2 - k]$$

$$\sim$$

$$A[i_2, j_2] \text{ := } A[i_2 - 1, j_2 - k] \text{ ; } A[i_1, j_1] \text{ := } A[i_1 - 1, j_1 - k]$$

CVC Input

```
i1, j1, i2, j2, k, arb_addr : INT;
a : 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))) =>
((LET a1 : ARRAY INT OF ARRAY INT OF INT =
    a WITH [i1][j1] := a[i1-1][j1-k] IN
    a1 WITH [i2][j2] := a1[i2-1][j2-k])[arb_addr] =
 (LET a1 : ARRAY INT OF ARRAY INT OF INT =
    a WITH [i2][j2] := a[i2-1][j2-k] IN
    a1 WITH [i1][j1] := a1[i1-1][j1-k])[arb_addr]);
```

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: $k \geq 0$.

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

for $i = 0$ **to** $N - 1$

$*(p + i)$ **:=** $*(r + i)$

for $i = 0$ **to** $N - 1$

$*(q + i)$ **:=** $*(r + i)$

Perfect Candidate for Fusion

for $i = 0$ **to** $N - 1$

$*(p + i)$ **:=** $*(r + i)$

$*(q + i)$ **:=** $*(r + i)$

Deriving Run-Time Tests with CVC

Fusion Example

for $i = 0$ **to** $N - 1$
$\quad *(p + i)$ **:=** $*(r + i)$
for $i = 0$ **to** $N - 1$
$\quad *(q + i)$ **:=** $*(r + i)$

$\Longrightarrow$

for $i = 0$ **to** $N - 1$
$\quad *(p + i)$ **:=** $*(r + i)$
$\quad *(q + i)$ **:=** $*(r + i)$

Verification Condition

$$i_1 < i_2 \longrightarrow$$
$$*(p + i_2) \text{ := } *(r + i_2) \text{ ; } *(q + i_1) \text{ := } *(r + i_1)$$
$$\sim$$
$$*(q + i_1) \text{ := } *(r + i_1) \text{ ; } *(p + i_2) \text{ := } *(r + i_2)$$

CVC Input

```
p, q, r : INT;

i1, i2, arb_addr : INT;

M : ARRAY INT OF INT;

QUERY

(i1 < i2) =>

(LET M1 : ARRAY INT OF INT =

  M WITH [q + i1] := M[r + i1] IN

  M1 WITH [p + i2] := M1[r + i2])[arb_addr] =

(LET M1 : ARRAY INT OF INT =

 M WITH [p + i2] := M[r + i2] IN

 M1 WITH [q + i1] := M1[r + i1])[arb_addr]);
```

CVC Output

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

$$q - r = i2 - i1$$

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

$$r - p = i2 - i1$$

Repeating this one more time yields:

$$q - p = i2 - i1.$$

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

Using the inequality $0 < i2 - i1 < N$, we get the run-time test:

($q - r \leq 0$ OR $q - r \geq N$) AND
($q - p \leq 0$ OR $q - p \geq N$) AND
($r - p \leq 0$ OR $r - p \geq N$)

Deriving Run-Time Tests with CVC

Fusion Example

for $i = 0$ **to** $N - 1$
  $*(p + i)$ **:=** $*(r + i)$
for $i = 0$ **to** $N - 1$
  $*(q + i)$ **:=** $*(r + i)$

$\Longrightarrow$

if (($q - r \leq 0$ **OR** $q - r \geq N$**) AND**
  ($q - p \leq 0$ **OR** $q - p \geq N$**) AND**
  ($r - p \leq 0$ **OR** $r - p \geq N$**))**
  for $i = 0$ **to** $N - 1$
    $*(p + i)$ **:=** $*(r + i)$
    $*(q + i)$ **:=** $*(r + i)$
else
  for $i = 0$ **to** $N - 1$
    $*(p + i)$ **:=** $*(r + i)$
  for $i = 0$ **to** $N - 1$
    $*(q + i)$ **:=** $*(r + i)$