Using Constraint Solvers in Interactive and Automated Theorem Proving[1]

Natarajan Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Nov 10, 2011

---

[1] This work was supported by NSF Grant CSR-EHCS(CPS)-0834810 and NASA Cooperative Agreement NNX08AY53A.

John McCarthy: Logic and Artificial Intelligence

It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last. The development of this relationship demands a concern for both applications and for mathematical elegance.

Constraints and Deduction

- Constraint solving over finite and infinite domains form the core of inference.
- Boolean (SAT) and theory satisfiability (SMT) are critical techniques for hardware and software verification.
- These techniques have many interesting applications.
- We have implemented and used several constraint solvers in tools such as PVS, SAL, Yices, Probabilistic Consistency Engine (PCE), and SimCheck and DimSim.
- We review some of the techniques and their applications (e.g., dimension analysis) in automated and interactive tools for theorem proving and verification.

A Suite of Constraint-Based Tools

- STP: Shostak's decision procedure combining equality and arithmetic
- PVS: Interactive theorem prover for higher-order logic with subtype constraints
- SAL: Transition system framework with model checking and analysis tools
- Yices: Solver for Boolean + Theory satisfiability
- PCE: SAT solver with probabilities
- SimCheck/SimProver: Assertion-based verification for Simulink models
- DimSim: Modular dimension checker for Simulink

Talk Outline

- Basic principles of inference-based constraint solving
- Resolution and Satisfiability with Conflict-Directed Clause Learning
- Satisfiability Modulo Theories (with Bruno Dutertre)
- Modular dimension checking (with Sam Owre and Indranil Saha)
- Timing verification and Scheduling (based on slides by Bruno Dutertre)
- Constraints in Interactive Proving
- Probabilistic Inference (with Sam Owre and Shalini Ghosh)

Logic Basics

- Logic studies the *trinity* between *language*, *interpretation*, and *proof*.
- *Language* circumscribes the syntax that is used to construct sensible assertions.
- *Interpretation* *fixes* the meaning of certain symbols, e.g., the logical connectives, equality, and *delimiting the variation* in the meanings of other symbols, e.g., variables, functions, and predicates.
- Constraint solving is about finding satisfying variable assignments for a formula.
- When there is no such assignment, the formula is unsatisfiable,
- *Both satisfiability and unsatisfiability have positive applications.*

Inference System

- An inference system $\mathcal{I}$ for a language and theory is an inference structure $\langle \Psi, \Lambda, \vdash \rangle$ (state, logical interpretation, and inference relation) that is
  1. **Conservative:** Whenever $\varphi \vdash_{\mathcal{I}} \varphi'$, $\Lambda(\varphi)$ and $\Lambda(\varphi')$ are $\mathcal{T}$-equisatisfiable.
  2. **Progressive:** The reduction relation $\vdash_{\mathcal{I}}$ should be well-founded, i.e., infinite sequences of the form $\langle \varphi_0 \vdash \varphi_1 \vdash \varphi_2 \vdash \ldots \rangle$ must not exist.
  3. **Canonizing:** A state is irreducible only if it is either $\bot$ or is $\mathcal{T}$-satisfiable.
- *If formulas can be coded as a state, the inference system is a sound and complete inference procedure for satisfiability.*
- Implementing inference relation yields a decision procedure.

Ordered Resolution

- Input K is a set of clauses.
- Atoms are ordered by $\succ$ which is lifted to literals so that $\neg p \succ p \succ \neg q \succ q$, if $p \succ q$.
- Literals appear in clauses in decreasing order without duplication.
- Tautologies, clauses containing both l and $\bar{l}$, are deleted from initial input.

Res	$\dfrac{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2}{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2, \Gamma_1 \vee \Gamma_2}$	$\Gamma_1 \vee \Gamma_2 \notin K$ $\Gamma_1 \vee \Gamma_2$ is not tautological
Contrad	$\dfrac{K, l, \bar{l}}{\bot}$	

Ordered Resolution: Example

$$\cfrac{\cfrac{\cfrac{\cfrac{(K_0 =)\ \neg p \vee \neg q \vee r,\quad \neg p \vee q,\quad p \vee r,\quad \neg r}{(K_1 =)\ \neg q \vee r,\quad K_0}\ \textbf{Res}}{(K_2 =)\ q \vee r,\quad K_1}\ \textbf{Res}}{(K_3 =)\ r,\quad K_2}\ \textbf{Res}}{\bot}\ \textbf{Contrad}$$

Correctness

- **Progress:** Bounded number of clauses in the given literals. Each application of **Res** generates a new clause.
- **Conservation:** For any model M, if $M \models l \vee \Gamma_1$ and $M \models \bar{l} \vee \Gamma_2$, then $M \models \Gamma_1 \vee \Gamma_2$.
- **Canonicity:** Given an irreducible non-$\bot$ configuration K in the atoms $p_1, \ldots, p_n$ with $p_i \prec p_{i+1}$ for $1 \leq i \leq n$, build a series of partial interpretations M_i as follows:
  1. Let $M_0 = \emptyset$
  2. If p_{i+1} is the maximal literal in a clause $p_{i+1} \vee \Gamma \in K$ and $M_i \not\models \Gamma$, then let $M_{i+1} = M_i\{p_{i+1} \mapsto \top\}$.
  3. Otherwise, let $M_{i+1} = M_i\{p_{i+1} \mapsto \bot\}$.
- Each M_i satisfies all the clauses in K in the atoms $p_1, \ldots, p_i$.

CDCL Informally

- Goal: Does a given set of clauses K have a satisfying assignment?
- If M is a total assignment such that $M \models \Gamma$ for each $\Gamma \in K$, then $M \models K$.
- If M is a partial assignment at level h, then *propagation* extends M at level h with the *implied literals* l such that $l \vee \Gamma \in K \cup C$ and $M \models \neg\Gamma$.
- If M detects a conflict, i.e., a clause $\Gamma \in K \cup C$ such that $M \models \neg\Gamma$, then the conflict is *analyzed* to construct a conflict clause that allows the search to be continued from a prior level.
- If M cannot be extended at level h and no conflict is detected, then an unassigned literal l is *selected* and assigned at level $h + 1$ where the search is continued.

Conflict-Driven Clause Learning (CDCL) SAT

Name	Rule	Condition
Propagate	$\dfrac{h, \langle M \rangle, K, C}{h, \langle M, l[\Gamma] \rangle, K, C}$	$\Gamma \equiv l \vee \Gamma' \in K \cup C$ $M \models \neg\Gamma'$
Select	$\dfrac{h, \langle M \rangle, K, C}{h+1, \langle M; l[] \rangle, K, C}$	$M \not\models l$ $M \not\models \neg l$
Conflict	$\dfrac{0, \langle M \rangle, K, C}{\bot}$	$M \models \neg\Gamma$ for some $\Gamma \in K \cup C$
Backjump	$\dfrac{h+1, \langle M \rangle, K, C}{h', \langle M_{\leq h'}, l[\Gamma'] \rangle, K, C \cup \{\Gamma'\}}$	$M \models \neg\Gamma$ for some $\Gamma \in K \cup C$ $\langle h', \Gamma' \rangle$ $= \mathit{analyze}(\psi)(\Gamma)$ for $\psi = h, \langle M \rangle, K, C$

CDCL Example

- Let K be $\{p \vee q, \neg p \vee q, p \vee \neg q, s \vee \neg p \vee q, \neg s \vee p \vee \neg q, \neg p \vee r, \neg q \vee \neg r\}$.
-

step	h	M	K	C	Γ
select s	1	$; s$	K	$\emptyset$	_
select r	2	$; s; r$	K	$\emptyset$	_
propagate	2	$; s; r, \neg q[\neg q \vee \neg r]$	K	$\emptyset$	_
propagate	2	$; s; r, \neg q, p[p \vee q]$	K	$\emptyset$	_
conflict	2	$; s; r, \neg q, p$	K	$\emptyset$	$\neg p \vee q$

CDCL Example (contd.)

step	h	M	K	C	Γ
conflict	2	$;s;r,\neg q,p$	K	$\emptyset$	$\neg p \vee q$
backjump	0	$\emptyset$	K	q	_
propagate	0	$q[q]$	K	q	_
propagate	0	$q,p[p \vee \neg q]$	K	q	_
propagate	0	$q,p,r[\neg p \vee r]$	K	q	_
conflict	0	q,p,r	K	q	$\neg q \vee \neg r$

CDCL Correctness

- **Progress:** Each backjump step adds a new assignment at the level h' so that $\sum_{i=0}^{h} |M_i| * (N+1)^{(N-h)}$ increases toward the bound $(N+1)^{(N+1)}$ for $N = |vars(K)|$. In the example, $N = 4$, the backjump step goes from a value 1300 in base 5 to the value 10000 which is closer to the bound 40000.
- **Conservation:** In each transition from $\langle M, K, C\rangle$ to $\langle M', K', C'\rangle$ (or $\perp$), the clause sets $M_0 \cup K \cup C$ and $M_0 \cup K' \cup C'$ are equisatisfiable.
- **Canonicity:** In an irreducible non-$\perp$ state, M is total assignment and there is no conflict so for each clause Γ in $K \cup C$, $M \models \Gamma$.

Example Inference Systems

- Inference systems help structure the correctness arguments.
- Several theoretical results are in *Modularity and refinement in inference systems* [Ganzinger, R, S].
- Simplifiers are inference systems without canonicity.
- Many inference algorithms can be described as inference systems, e.g.,
  1. Union-find for equality
  2. Propositional resolution
  3. Basic superposition for equality/propositional reasoning
  4. CDCL
  5. Simplex-based linear arithmetic reasoning
  6. SMT

SMT Overview

- In SMT solving, the Boolean atoms represent constraints over individual variables ranging over integers, reals, datatypes, and arrays.
- The constraints can involve theory operations, equality, and inequality.
- The SAT solver has to interact with a theory constraint solver which propagates truth assignments and adds new clauses.
- The theory solver can detect conflicts involving theory reasoning, e.g.,
    1. $f(x) = f(y) \vee x \neq y$
    2. $f(x-2) \neq f(y+3) \vee x - y \leq 5 \vee y - z \leq -2 \vee z - x \leq -3$
    3. $x\ XOR\ y \neq 0b0000000 \vee select(store(A, x, v), y) = v$
- The theory solver must produce efficient explanations, incremental assertions, and efficient backtracking.

Example Constraint Solvers

- **Core theory:** Equalities between variables $x = y$, offset equalities $x = y + c$.
- **Term equality:** Congruence closure for uninterpreted function symbols
- **Difference constraints:** Incremental negative cycle detection for inequality constraints of the form $x - y \leq k$.
- **Linear arithmetic constraints:** Fourier's method, Simplex.
- **Bit Vectors:** Bit-blasting

Theory Constraint Solver Interface

The satisfiability procedure uses a theory constraint solver oracle which maintains the theory state S with the interface operations:

1. $assert(l, S)$ adds literal l to the theory state S returning a new state S' or $\bot[\Delta]$
2. $check(S)$ checks if the conjunction of literals asserted to S is satisfiable, and returns either $\top$ or $\bot[\Delta]$.
3. $retract(S, l)$: Retracts, in reverse chronological order, the assertions up to and including l from state S.
4. $model(S)$: Builds a model for a state known to be satisfiable.

Satisfiability Modulo Theories

- SMT deals with formulas with theory atoms like $x = y$, $x \neq y$, $x - y \leq 3$, and $select(store(A, i, v), j) = w$.
- The CDCL search state is augmented with a *theory state* S in addition to the partial assignment.
- Total assignments are *checked* for theory satisfiability.
- 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, the literals deleted from M are also *retracted* from S.

A Theory Solver: Gauss–Jordan Elimination

- GJ is a constraint solver for linear arithmetic equalities.
- The logical state consists of the input constraints G, where each constraint is of the form $p = 0$ and the solution state S.
- For each variable x, $S(x)$ returns a polynomial.
- x is a basis variable iff $S(x) \neq x$.
- The operation S[p] replaces each variable x in p with $S(x)$ and renormalizes to an order sum-of-products form.
-

Delete	$\dfrac{G, p; S}{G; S}$	if $S[p] = 0$
Contrad	$\dfrac{G, p; S}{\bot}$	if $S[p] = k \neq 0$
Solve	$\dfrac{G, p; S}{G; S\{x \leftarrow q\}}$	if $S[p] = kx + r$ with $k \neq 0$ and $q = -r/k$

SMT Applications

- **Test generation:** Find assignments to the individual variables satisfying a path constraint in a program.
- **Infinite-state bounded model checking:** BMC for programs with assignments, unbounded arithmetic, arrays, datatypes, and timers.
- **Predicate abstraction and abstract reachability:** For an atom substitution γ and formula ϕ, find Boolean formula $\hat{\phi}$ such that $\phi \implies \gamma(\hat{\phi})$.
- Scheduling, planning, constraint solving, and MaxSAT in unbounded domains.

Example Uses of Yices

Model Checking

- Backend solver to the SAL model checkers (SRI)
- MCMT (Ghilardi & Ranise)
- Model checking of Lustre Programs (Hagen & Tinelli)

Program Analysis

- Symbolic Execution: Sireum/Kiasan (Deng, Robby, Hatcliff), JPF (Anand, Păsăreanu, Visser)
- Backend prover for SPARK-ADA (Jackson, Ellis, Sharp)

Within Interactive Theorem Provers

- PVS, Isabelle/HOL can use Yices as an *end-game* solver

Applying Gauss–Jordan in Dimension Checking

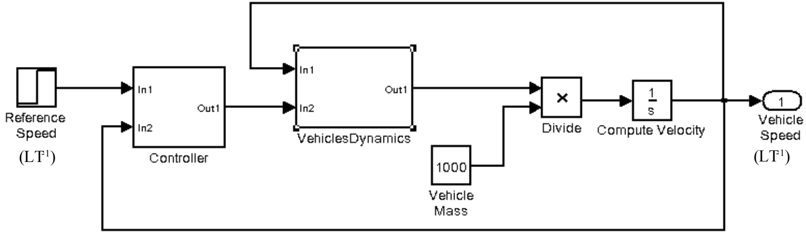
- Dimensional mismatches have led to some spectacular system failures: Mars Climate Orbiter (1999), SDI test using Space Shuttle Discovery (1985)
- Assume a fixed number of basic dimensions, e.g., mass, length, time.
- Encode the dimension of each variable as a triple $\langle l, m, t\rangle$, representing the product $L^l M^m T^t$, e.g., LMT^{-2}
- The dimension signature for operations yields a constraint system
  - $z = x + y$ generates the constraint that $dim(x) = dim(y) = dim(z)$.
  - $z = x * y$ generates the constraint that $dim(z) = dim(y) + dim(z)$

Dimension Checking Simulink Models

- Simulink represents state machines by flow diagrams.
- Each model is a block consisting of input and output signals.
- Blocks can be composed of primitive blocks for operations such as addition, multiplication, differentiation, and integration.
- The signals are numeric data and typically have physical interpretations.
- Errors do occur from dimensional mismatches, e.g., velocity instead of acceleration.
- We only handle dimension solving, but are extending the analysis to units and conversions between units.
- DimSim is a dimension checker for Simulink that uses a constraint solver based on Gauss–Jordan elimination.

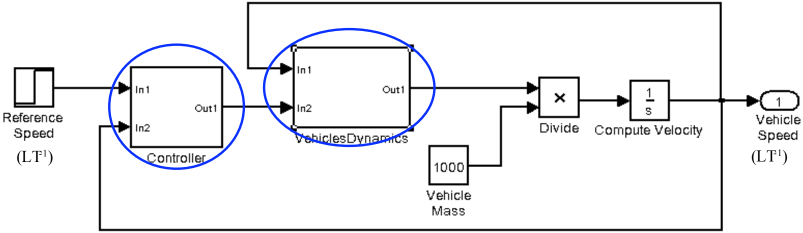
DimSim: Dimension Checker for Simulink

- Input: A Simulink model whose signals may be annotated with their dimensions
- Objective:
  - Determine the dimensions of all signals in the model uniquely, if possible
  - Otherwise, check dimensional consistency of the model, and find out the most general dimensions of the signals
  - In case of an inconsistency, provide the root cause



Compositional Dimension Analysis

- Dimension checking algorithm is compositional
  - Dimension consistency of the lower level subsystems is first checked
  - To check the higher level subsystem only the interfaces of the lower level subsystems are considered
  - Helps in achieving scalability



Modular Dimension Solving : Gauss-Jordan Elimination

- Each block consists of input, output, and internal variables connected into data flow diagrams using primitive and compound sub-blocks.
- Each sub-block exports the dimensional constraints on its port variables.
- These are imported by the block by suitably renaming the constraints.
- The local and imported constraints are asserted to a GJ solver.
- The solver detects
  - Inconsistency, i.e., the absence of a valid dimensional assignment: DimSim identifies the core unsatisfiable constraints
  - Under-constraint, i.e., an internal signal whose dimension is determined by those of the external variables
- The dimensional constraints on the inputs and outputs are exported to any parent subsystem.

Case Studies

Model	Domain
Thermal model of a house (TMH)	General application
Collision avoidance system (CD2D)	Aerospace
Cruise control system (CC)	Automotive
Rotating clutch system (RC)	Automotive
Engine timing control system (ETC)	Automotive
Transmission control system (TC)	Automotive
Robot motion control system (RMC)	Robotics

Experimental Results

Model	Blocks	Variables	Subsystems	Required Annotations	Constraints	Errors found
TMH	48	79	3	12	95	0
CD2D	93	164	9	23	213	1
CC	74	139	6	28	149	0
RC	102	201	10	41	295	0
ETC	113	220	12	43	304	0
TC	930	1935	34	425	3240	1
RMC	276	526	17	78	2637	1

Table: Model data

Types of Errors Found

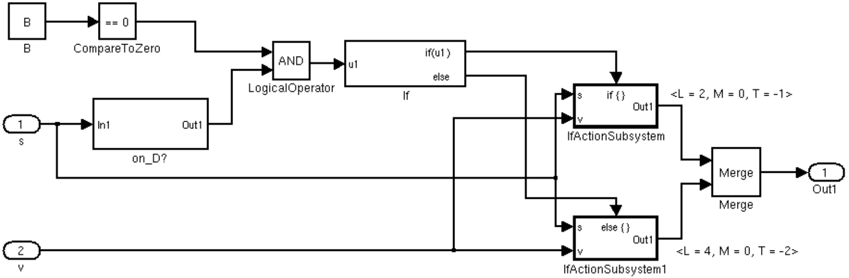
- Erroneous Annotation
- Erroneous Design
- Erroneous Constant
- Incorrect Blocks Usage
- Missing Blocks

Experimental Results

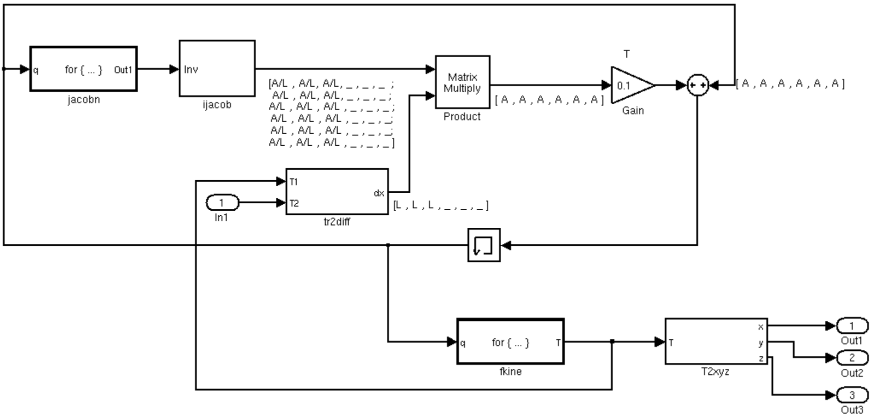
Error	Model	Type of Error	No of UC Constraints
Error1	CD2D	Erroneous Design	3
Error2	TC	Erroneous constant	11
Error3	RMC	Incorrect presence of a block	31

Table: Error Data

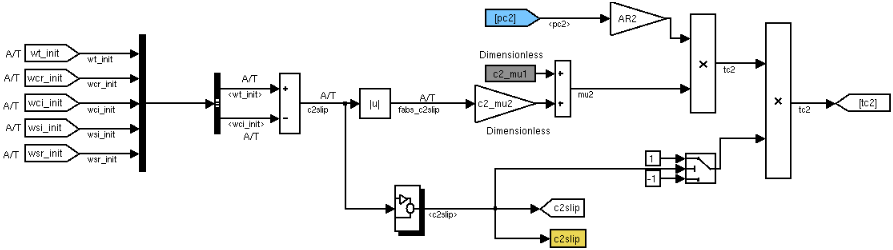
Dimension Error in CD2D Model



Dimension Error in M7 Model



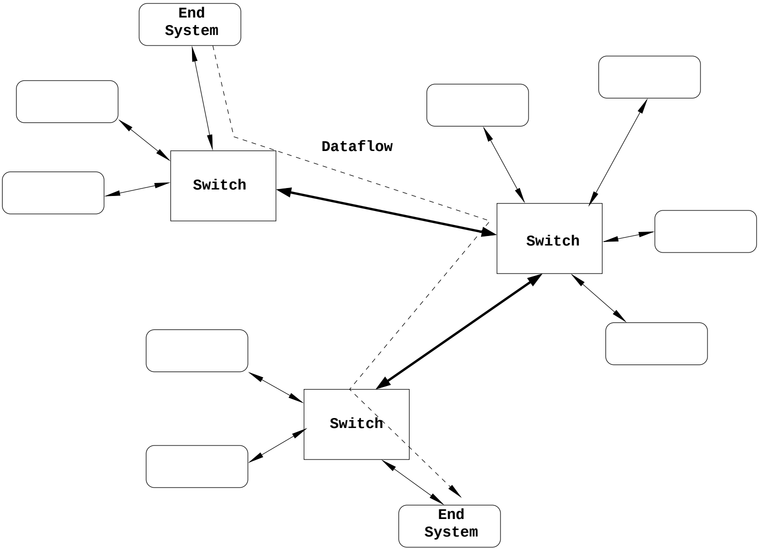
Dimension Error in ETC Model



Related Work

- Wand and OḰeefe [1991] add Kennedy [1994] worked on dimension analysis of functional programming languages.
  - provided unification based algorithm to find the most general dimensions for every typable dimension preserving terms
- A number of earlier works on different programming languages
  - Pascal - [Agrawal and Garg, 1984]
  - ADA - [Hilfinger, 1988] and [Rogers, 1988]
  - C++ - [Umrigar, 1994] and [Cmelik and Gehani, 1988]
  - Java - [VanDelft, 1999]
  - FORTRAN - [Petty 2001]
  - Fortress (Extension of Java) - [Allen et al., 2004]
  - Spreadsheets - [Antoniu et al., 2004]
  - C - [Jiang and Su, 2006]

Application: Scheduling for TTEthernet



Ethernet for real-time, distributed systems:

Computing a Communication Schedule

Input

- a set of virtual links: dataflows from one end system to one or more end systems
- the communication period

Constraints

- no contention: all frames on every link are in a different time slot
- path constraints: relayed frames must be scheduled after they are received
- other constraints: limits on switch memory, application constraints, etc.

TTE Scheduling as an SMT Problem (Steiner, 2010)

Frames

- Messages are called frames in TTE.
- A frame f is characterized by its period $f.period$ and its length $f.length$.
- Routing is static: we know a priori the source of f, all receivers, and the set of communication links that will transport f.
- Given a link i, our goal is to compute when to send f over that link. The start of this transmission is denoted by $offset_{f,i}$

Simplification: in the simplest case, all frames have the same period (equal to the schedule cycle).

Example Scheduling Constraints

No Collisions: if distinct frames f and g use link i:

$offset_{f,i} + f.length \leq offset_{g,i}$ or $offset_{g,i} + g.length \leq offset_{f,i}$

Path Constraints: if a switch receives f on link i and relays it on link j

$$offset_{f,j} - offset_{f,i} \geq maxhopdelay$$

End-to-End Latency: along a path $i_0, i_1, \ldots, i_n$

$$offset_{f,i_n} - offset_{f,i_0} \leq maxlatency$$

Resulting SMT Problem

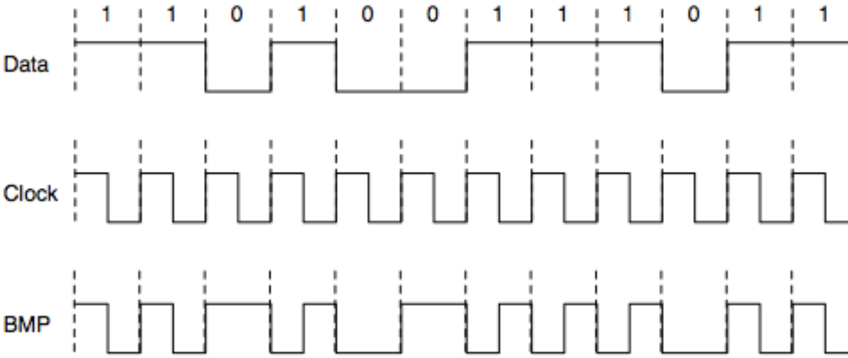
Large Difference Logic Problem (over the integers)

- Typical size: 10000-20000 variables, 10^6 to 10^7 constraints
- This depends on the network topology and number of virtual links

Solving this with Yices

- Yices 1 can solve moderate size instances (about 120 virtual links) out of the box
- In Wilfried Steiner's RTSS 2010 paper: incremental approach using push/pop can solve much larger instances (up to 1000 virtual links)

Example: Biphase Mark Protocol (BMP)



Biphase Mark: Physical layer protocol for data transmission (over serial links)

- transmitter and receiver have independent clocks
- encoding merges transmitter clock + data into a single bit stream

BMP Verification [Brown and Pike, TACAS'06]

Proof Process

- The correctness property is not invariant (for any reasonable *k*)
- We need auxiliary lemmas:

```
l0 : LEMMA system |- G(phase = Settle OR tda
l1 : LEMMA system |- G(phase = Stable => (tc
l2 : LEMMA system |- G(phase = Settle => (tc
```

- The full proof requires four auxiliary lemmas, the main one is proved by *k* induction for $k = 5$.
- All proofs run in a few seconds.

Much Easier than Previous Proofs of BMP

- Vaandrager and de Groot, 2004, use PVS and Uppaal
  Difficult proof: need 37 invariants, 4000 proof steps, hours to run

PVS in a Slide

- Prototype Verification System (PVS) is an interactive specification/verification system developed over the last twenty years.
- The PVS specification language extends higher-order logic with predicate subtypes, dependent types, parametric theories, and theory interpretations.
- Type constraints handle array index bounds, division by zero, and a range of other sanity checks – a well-typed program can only crash due to resource limitations.
- Arbitrary formulas can be used as type constraints – type checking is undecidable.
- Many features of the language generate proof obligations that can be discharged interactively or automatically.
- The PVS interactive prover integrates SMT solvers, BDDs, rewriting, case analysis, and quantifier instantiation.

Interactive Proof: N-Queens in PVS

- We use PVS to define and verify an algorithm to check find a legal placement for N queens on an $N \times N$ chess board, if there is one.
- A placement is just a mapping from the column index to the row index containing the queen for that column.

```
nqueens   [N: nat ]: THEORY
  BEGIN

   board : TYPE = [below(N)->below(N)]
   A, B, queen, new_queen: VAR board

   i, j, k: VAR upto(N)

   extends(i, A, queen): bool =
      (FORALL (j: below(i)): A(j) = queen(j))

   p: VAR [board -> bool]
   .
   .
   .
  END nqueens
```

Guarded Lifted Type

For search problems, the result type should capture the meaning of success and failure.

```
qlift?(p)(x : lift[board]): bool =
  CASES x OF
   bottom: (FORALL queen: NOT p(queen)),
   up(queen): p(queen)
  ENDCASES

good_extension?(i, A, p)(B): bool =
  (p(B) AND extends(i, A, B))
```

An invariant of the search is that for any partial assignment

1. The prior assignments have no good extensions, and
2. The continuation of the search with this assignment yields a good extension, if there is one.

Search Within Column

To position, a queen within a column, try each position to see if the continuation of the search on the remaining columns (with parameter `f`) succeeds.

```
    search((i: below(N)), A, p,
           (j | (FORALL (k: below(j), B):
                  NOT good_extension?(i+1, A WITH [i:= k], p)(B))),
           (f: [B: board -> (qlift?(good_extension?(i+1, B, p)))]))
      : RECURSIVE
      (qlift?(good_extension?(i, A, p))) =
     (IF j = N THEN bottom
       ELSE LET B = A WITH [i := j]
                IN CASES f(B) OF
                         bottom: search(i, A, p, j+1, f),
                         up(C): up(C)
                    ENDCASES
      ENDIF)
      MEASURE N - j
```

Scan Across Columns

To position the queens from columns `i` upwards, search for a position in column `i` that can be extended with a solution from column `i+1` upwards.

```
    scan(i, p)(queen): RECURSIVE
      (qlift?(good_extension?(i, queen, p)))
      =
     (IF i = N
       THEN IF p(queen)
              THEN up(queen)
              ELSE bottom
             ENDIF
       ELSE search(i, queen, p, 0, scan(i+1, p))
      ENDIF)
      MEASURE N - i
```

N-Queens Search

The search operation is exhaustive.

```
   findboard(p): (qlift?(p)) =
      scan(0, p)(LAMBDA (i: below(N)): 0)

   goodqueen?(queen): bool =
     (FORALL (i, j: below(N)):  i /= j IMPLIES
         (queen(i) /= queen(j) AND
          (i - j /= queen(i) - queen(j)) AND
          (j - i /= queen(i) - queen(j))))
```

A good placement is one where no two queens are on the same row or on the same upward or downward diagonal.

N-Queens Summary

- There are no explicit theorems, since the proofs are all in the proof obligations (TCCs) generated by the typechecker.
- There are 23 TCCs, 4 are subsumed, 12 are discharged by the default strategy.
- The remaining seven TCCs are proved with a modest amount (five to ten steps) of interaction.

Logic and Probability (Wikipedia)

- Medical diagnosis offers a simple example of Bayesian reasoning.
- We have a test for a disease that returns positive or negative results.
- If the patient has the disease, the test is positive with probability .99.
- If the patient does not have the disease, the test is positive with probability .05.
- A patient has the disease with probability .001.
- What is the probability that a patient with a positive test has the disease?
- $Pr(D|pos) = Pr(pos|D)Pr(D)/P(pos) = .99 \times .001/(.99 \times .001 + .05 \times .999) = 99/5094 = .0194$

Medical Diagnosis in PCE

```
sort Patient;
const a: Patient;
predicate testedPositive(Patient) hidden;
predicate diseased(Patient) hidden;
add testedPositive(a) or ~diseased(a) 4.6; # 99%
add ~testedPositive(a) or ~diseased(a) .01; # 1%
add testedPositive(a) or diseased(a) .05;   # 5%
add ~testedPositive(a) or diseased(a) 3.0; # 95%
add ~diseased(a) 6.9; # 99.9%
add diseased(a) .001; # .1%

add testedPositive(a);
mcsat_params 1000000, 0.5, 20.0, 0.5, 30;
ask diseased(a);
```

Result:

```
[] 0.020: (diseased(a))
```

Conclusions

- Constraint solving is widely used in verification to
    - Generate test cases
    - Perform extended static analysis
    - Prove assertions
    - Check dimensional correctness
    - Schedule tasks
    - Determine probabilistic outcomes

  These technologies have already had a revolutionary impact.
- We have argued that are principled ways of constructing constraint solvers
- There are also many unconventional uses for constraint solving
- Building and integrating scalable solvers is an ongoing challenge.