Efficient Delta-decision Procedure

[Thesis Proposal]

Soonho Kong
soonhok@cs.cmu.edu

Carnegie Mellon University
Computer Science Department
Thesis Committee:

Edmund M. Clarke, Chair
Randal E. Bryant
Jeremy Avigad
Leonardo de Moura, Microsoft Research
Chapter I
Introduction
Decision Problems over the Reals

Given an arbitrary first-order sentence over \( \langle \mathbb{R}, \geq, \mathcal{F} \rangle \), such as

\[
\varphi = Q_1^{[l_1,u_1]}x_1 \ldots Q_n^{[l_n,u_n]}x_n \cdot \bigwedge_i \left( \bigvee_j f_{i,j}(\vec{x}) > 0 \lor \bigvee_k f_{i,k}(\vec{x}) \geq 0 \right)
\]

where \( f \in \mathcal{F} \), can we compute whether \( \varphi \) is true or false?

- Complexity results of non-linear arithmetic over the Reals
  - Decidable if \( \varphi \) only contains polynomials [Tarski51]
  - Undecidable if \( \varphi \) includes trigonometric functions (i.e. \( \sin \))

- Real-world problems contain complex nonlinear functions (trigonometric functions, log, exp, ODEs)
Delta-decision Problem

- Given a first-order formula over the Real $\varphi$, and a positive rational number $\delta$, the delta-decision problem asks for one of the following answers:
  - **UNSAT**: $\varphi$ is unsatisfiable
  - **$\delta$-SAT**: $\varphi^{-\delta}$ is satisfiable.

where $\varphi^{-\delta}$ is called the $\delta$-weakening of $\varphi$ which is formally defined as follows:

$$
\varphi^{-\delta} = Q_1^{[l_1,u_1]} x_1 \ldots Q_n^{[l_n,u_n]} x_n \cdot \bigwedge_i \left( \bigvee_j f_{i,j}(\bar{x}) > -\delta \vee \bigvee_j f_{i,k}(\bar{x}) \geq -\delta \right)
$$

- It is shown that this problem is **decidable** for signatures with computable functions [LICS12]

- The complexity for existential problems is **NP** (with P-time computable functions) or **PSPACE** (with Lipschitz ODEs) [LICS12]
This thesis aims to show the steps that are taken towards filling in this gap with convincing and practical examples showing the broad applicability of these procedures.
The thesis aims to show the steps that are taken towards filling in this gap with convincing and practical examples showing the broad applicability of these procedures.
This thesis aims to show the steps that are taken towards filling in this gap with convincing and practical examples showing the broad applicability of these procedures.

Research Questions:
- How to handle ODEs?
- How to integrate learning and non-chronological backtracking in solving?
- How to handle exist-forall problems and use the technique for optimization problems?
Chapter 2
Background
Design of Solver: Big Picture

- **SAT solver** finds a satisfying **Boolean** assignment
- **Theory solver** checks whether the assignment is feasible under the first order theory of **Real**
Design of Solver: Big Picture

SAT Solver

List of Constraints

Theory Solver

\( \delta \)-SAT (+Solution)

or

UNSAT (+Explanation)

Boolean Search
Non-chronological Backtracking
Learning
...

(Discrete Domain)

Constraints Solving
Validated Numerics
Optimization
Simulation/Sampling
...

(Continuous Domain)
Top-down/Bottom Approaches in Theory Solver

**Top-Down Approach**
- Maintain a set of possible solutions
- Useful to show **UNSAT**
- Validated Numerics (i.e. Interval-based methods)

**Bottom-Up Approach**
- Sample points and test them
- Useful to show **SAT**
- Use local-optimization to improve
An Algorithm in Theory Solver: ICP (Interval Constraint Propagation)

Pruning

Safely **reduce** a search space without removing solutions

Branch

**Partition** a search space into two sub-spaces

Fixedpoint Computation
Two Termination Conditions of ICP

δ-sat

Unsat
ICP Algorithm

Algorithm 1: Theory Solving in DPLL(ICP)

input : A conjunction of theory atoms, seen as constraints,
c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n), the initial interval bounds on all
variables B^0 = I_1^0 \times \cdot \cdot \cdot \times I_n^0, box stack S = \emptyset, and precision \delta \in \mathbb{Q}^+.
output: \delta\text{-sat}, or unsat with learned conflict clauses.

1. S.push(B_0);
2. while S \neq \emptyset do
3. \hspace{1em} B \leftarrow S.pop();
4. \hspace{2em} while \exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i) do
5. \hspace{3em} \text{//Pruning without branching, used as the assert() function.}
6. \hspace{4em} B \leftarrow \text{Prune}(B, c_i);
7. \hspace{1em} \text{//The \epsilon below is computed from \delta and the Lipschitz constants of}
8. \hspace{1em} \text{functions beforehand.}
9. \hspace{1em} if B \neq \emptyset then
10. \hspace{2em} if \exists 1 \leq i \leq n, |I_i| \geq \epsilon then
11. \hspace{3em} \{B_1, B_2\} \leftarrow \text{Branch}(B, i); \text{//Splitting on the intervals}
12. \hspace{3em} S.push(\{B_1, B_2\});
13. \hspace{2em} else
14. \hspace{3em} return \delta\text{-sat}; \text{//Complete check() is successful.}
15. \hspace{1em} end
16. end
17. return unsat;

Pruning
Branching
Chapter 3
Solving Delta-decision Problems with ODEs
[Completed Work]
Solving Delta-decision Problems with ODEs

Motivation

- **ODEs** are widely used in the design and verification of Hybrid Systems (i.e. in Biomedical, Robotics).

- Most of them include highly-nonlinear dynamics.

Cardiac Cell Action Potential Model

**Mode 1**

- $\text{flow}_1:$
  - $\frac{du}{dt} = \frac{u}{\tau_{a1}}$
  - $\frac{dv}{dt} = 1 - v$
  - $\frac{dw}{dt} = \frac{1 - u - w}{\tau_{c1} + \frac{\tau_{c2} - \tau_{c3}}{1 + e^{-2s(t-u)}}}$
  - $\frac{ds}{dt} = \frac{1}{1 + e^{-2s(t-u)}} - s \frac{1}{\tau_{c1}}$

**Mode 2**

- $\text{flow}_2:$
  - $\frac{du}{dt} = \frac{u}{\tau_{a2}}$
  - $\frac{dv}{dt} = -\frac{v}{\tau_{c2}}$
  - $\frac{dw}{dt} = \frac{w - w}{\tau_{c1} + \tau_{c2} - \tau_{c3}}$
  - $\frac{ds}{dt} = \frac{1}{1 + e^{-2s(t-u)}} - s \frac{1}{\tau_{c1}}$

**Mode 3**

- $\text{flow}_3:$
  - $\frac{du}{dt} = \frac{v - v}{\tau_{oi} + \frac{\tau_{oi} - \tau_{ol}}{1 + e^{-2s(t-u)}}}$
  - $\frac{dv}{dt} = \frac{v}{\tau_{oi}}$
  - $\frac{dw}{dt} = \frac{w}{\tau_{oi} + \tau_{ci}}$
  - $\frac{ds}{dt} = \frac{1}{1 + e^{-2s(t-u)}} - s \frac{1}{\tau_{oi}}$

**Mode 4**

- $\text{flow}_4:$
  - $\frac{du}{dt} = \frac{v(u - \theta)(u_0 - u)}{\tau_{oi}}$
  - $\frac{dv}{dt} = -\frac{v}{\tau_{oi}}$
  - $\frac{dw}{dt} = \frac{w}{\tau_{oi} + \tau_{ci}}$
  - $\frac{ds}{dt} = \frac{1}{1 + e^{-2s(t-u)}} - s \frac{1}{\tau_{oi}}$

---

Bing Liu, Soonho Kong, Sicun Gao, Paolo Zuliani, and Edmund Clarke, “Parameter Synthesis for Cardiac Cell Hybrid Models Using Delta-Decisions.”, CMSB 2014
Solving Delta-decision Problems with ODEs

Motivation

- **ODEs** are widely used in the design and verification of Hybrid Systems (i.e. in Biomedical, Robotics).

- Most of them include highly-nonlinear dynamics.

Solving Delta-decision Problems with ODEs

Approach

\[ X_t = X_0 + \int_0^T \text{flow}(x(s)) \, ds \]

1. Design pruning operators from an ODE constraint.

2. Use rigorous numerical ODE solvers to propagate interval assignments on initial/final/time variables.
Pruning using ODEs
(Forward)

\[ X_t = X_0 + \int_0^T \text{flow}(x(s)) \, ds \]

How can we prune \( X_t \)?
Pruning using ODEs (Forward)

\[ X_t = X_0 + \int_0^T \text{flow}(x(s)) \, ds \]

(numerically) Compute the enclosures of the solutions of ODE
Pruning using ODEs (Forward)

\[ X_t = X_0 + \int_0^T flow(x(s)) \, ds \]

(numerically) Compute the enclosures of the solutions of ODE
Pruning using ODEs
(Forward)

\[ X_t = X_0 + \int_0^T \text{flow}(x(s))\,ds \]
Pruning using ODEs (Backward)

Enclosures of Solutions of ODEs
Pruning using ODEs (on Time)

\[ X_t = X_0 + \int_0^T \text{flow}(x(s))\,ds \]

Enclosures of Solutions of ODEs
Pruning using ODEs
(using Invariant)
Solving Delta-decision Problems with ODEs

Result

* Implemented in dReal
* Can handle a formula with 250+ ODEs and 600+ Vars
* Published a paper in FMCAD’13
* There are applications and tools based on this technique
Solving Delta-decision Problems with ODEs

Result

Applications:

* Autonomous Driving (Penn) [SAE’16]
* Planning (CMU, SIFT) [AAAI’15]
* Atrial Fibrillation (Stony Brook, TU, CMU) [HSCC’15, CMSB’14]
* Diabetes (Penn) [ADHS’15]
* Prostate Cancer (Pitt, CMU) [HSCC’15]

Tools based on dReal:

* APEX: A Tool for Autonomous Vehicle Plan Verification and Execution (Toyota/UPenn)
* BioPSy: Parameter set synthesis on biological models (Univ. of Newcastle)
* dReach: Reachability analysis tool for hybrid system (CMU)
* ProbReach: Probabilistic reachability analysis of hybrid systems (Univ. of Newcastle)
* SReach: Bounded model checker for stochastic hybrid systems (CMU)
Chapter 4
SAT-driven Branch-and-Prune
[Work in Progress]
SAT-driven Branch-and-Prune

Motivation

SAT Solving
(DPLL/CDCL)

ICP

Search
Split Rule
(i.e. Making a decision)

Branching

Inference
Unit Resolution
(Boolean Constraint Propagation)

Pruning
## SAT-driven Branch-and-Prune

### Motivation

**Search**
- **Split Rule**
  - Conflict Clause Learning + Non-chronological Backtracking
- **Branching**
  - Depth-first Search without Learning
  - Chronological Backtracking with Stack

**Inference**
- **Unit Resolution**
  - Boolean Constraint Propagation

**Pruning**
- **ICP**

---

**SAT Solving**
- (DPLL/CDCL)
SAT-driven Branch-and-Prune

Motivation

Pruning: \( f_1(y) \land \{ x : [1, 3], y : [1, 5] \} \rightarrow \{ x : [1, 3], y : [2, 4] \} \)

Learned Clause: \( f_1(y) \land \{ y : [1, 5] \} \rightarrow \{ y : [2, 4] \} \)

(after generalization)
SAT-driven Branch-and-Prune Approach

ICP → Clause Manager : Report pruning/branching steps
Clause Manager → ICP : Provide the next box to visit
SAT-driven Branch-and-Prune Approach

SAT_ICP(Constraint f, Box b) {
    CM.init(b);  // set initial search space
    while (b = CM.next_box()) {
        // Pruning
        do {
            b' = f.prune(b);
            CM.learn(f, b → b');
        } while (b' ≠ ∅ ∧ b' ≠ b);
        if (b' = ∅) {
            break;  // try to get a new box
        }
        if (|b'| ≤ ε) {
            return δ-SAT(b');
        }
        // Branching
        (b1, b2) = branch(b');
        CM.learn(b' → b1 ∨ b2);
        b = b1;  // search b1 first
    }
    return UNSAT;  // no box to search
}
SAT-driven Branch-and-Prune Approach

ICP

Pruning
\[ f_1 \land B_1 \rightarrow B_2 \]
\[ f_2 \land B_3 \rightarrow \emptyset \]

Branching
\[ B \rightarrow B_1 \lor B_2 \]

Next Box?
\[ B \]

Clause Manager

Send SAT Query
Learned Clauses + Extra Constraints

Interpret Result
SAT/UNSAT

SAT Solver
**SAT-driven Branch-and-Prune Approach**

![Diagram of SAT Solver and Clause Manager]

**Boolean Encoding**

1. To each predicate \((x \geq c)\) (resp. \(x \leq c\)), associate a Boolean variable \(b_{(x \geq c)}\) (resp. \(b_{(x \leq c)}\))

\[
\{x : [1, 3], y : [1, 5]\} = (1 \leq x) \land (x \leq 3) \land (1 \leq y) \land (y \leq 5)
\]

\[
b_{1 \leq x} \land b_{x \leq 3} \land b_{1 \leq y} \land b_{y \leq 5}
\]

2. Introduce **Extra Constraints**

**Ordering Constraints:**

\((x \leq 1) \rightarrow (x \leq 3) \quad (x \geq 3) \rightarrow (x \geq 1)\)

**Disjointness Constraints:**

\((x \geq 3) \rightarrow \neg(x \leq 1)\)
SAT-driven Branch-and-Prune Approach

Clause Manager

Simplification of Clauses

1. Using resolution rule to infer new clauses

\[ b_1 \rightarrow b_2 \quad \neg b_2 \]

\[ \therefore \neg b_1 \]

2. Using subsumption rule to eliminate redundant clauses

\[ \{b_1 \rightarrow b_2, \neg b_2, \neg b_1\} \quad \Rightarrow \quad \{\neg b_1\} \]
SAT-driven Branch-and-Prune Approach

Simplification of Clauses
3. Replacing two adjacent boxes with a single box by merging them

4. Relaxing/enlarging a box using its neighbors
SAT-driven Branch-and-Prune Approach

An example:

B8 → B7
SAT-driven Branch-and-Prune Approach

An example: B8 → B7
SAT-driven Branch-and-Prune Approach

An example:

B8 → B7
SAT-driven Branch-and-Prune Approach

An example:

B8 → B7
SAT-driven Branch-and-Prune Approach

Clause Manager

An example:

B8 → B7
SAT-driven Branch-and-Prune Approach

An example:

B8 → B7
SAT-driven Branch-and-Prune Approach

Clause Manager

An example:

B8 → B7
SAT-driven Branch-and-Prune

Proposed Work

• Prove SAT+ICP algorithm **terminates**.

• Prove **correctness** of SAT+ICP algorithm.
  The outputs from naive ICP and SAT+ICP should be identical.

• Show that SAT+ICP algorithm **outperforms** naive ICP.

• Use **Boxes/LDD** data structure to implement Clause Manager and check the performance gain.

---

Chapter 5
Solving Exist-forall Formulas
[Work in Progress]
Solving Exist-forall Formulas

Motivation

- Global
- Non-convex
- Non-linear
- Multi-objective
- Combinatorial (disjunctions)

Optimization

Challenging Problems in Optimization
Solving Exist-forall Formulas

Approach

Encode optimization problems into first-order formula over Real with one alternation of quantifiers

\[
\min f(x) \text{ s.t. } \phi(x) \\
\exists x. \forall y. \phi(x) \land \phi(y) \rightarrow f(x) \leq f(y)
\]

and solve exist-forall problems.
Solving Exist-forall Formulas

Approach

An example:

$$\exists_{[-5,5]} x . \forall_{[-4,4]} y . x^2 + y^2 \leq 5^2$$

(A) Initial Search Space: x = [-5, 5]
Solving Exist-forall Formulas

Approach

An example:

\[ \exists [-5,5] x . \forall [-4,4] y . x^2 + y^2 \leq 5^2 \]

(A) Initial Search Space: \( x = [-5, 5] \)

(B) Find a counterexample

\[
\begin{align*}
x &= -4.5 \\
y &= 3
\end{align*}
\]
Solving Exist-forall Formulas

Approach

An example:

\[ \exists [-5,5] x. \forall [-4,4] y. \quad x^2 + y^2 \leq 5^2 \]

(A) Initial Search Space: \( x = [-5, 5] \)

(B) Find a counterexample

\[ x = -4.5 \]
\[ y = 3 \]

(C) Prune \( x \) using the counterexample

\[ x^2 + 3^2 \leq 5^2 \]
\[ x^2 \leq 5^2 - 3^2 = 16 = 4^2 \]
\[ -4 \leq x \leq 4 \]
Solving Exist-forall Formulas

Approach

Counterexample-guided Pruning Algorithm for exist-forall Problem

\[ \exists x. \forall y. \varphi(x, y) \]

1. Counterexample generation:

   \[ b \leftarrow \text{Solve}(y, \neg \varphi(x, y)) \]

2. Pruning on x using the counterexample y = b:

   \[ B_x \leftarrow \text{Prune}(B_x, \neg \varphi(x, b)) \]

Repeat until it fails to find a counterexample in step 1 or reaches a fixedpoint.
Solving Exist-forall Formulas

Approach
Solving Exist-forall Formulas

Approach

Counterexample
Solving Exist-forall Formulas

Approach

Counterexample

Local Optimization

$I_x''$

$I_x'$

$I_x$
Solving Exist-forall Formulas

Approach

Exploit the structure of optimization problem:

\[ \exists x. \forall y. f(x) \leq f(y) \]

1. Counterexample generation:
   \[ b \leftarrow \text{Solve}(y, \neg \varphi(x, y)) \]

2. Use local-optimization to enhance the quality of a counterexample:
   \[ b \leftarrow \text{localOpt}(f, b) \]

3. Pruning on \( x \) using the counterexample \( b = y \):
   \[ B_x \leftarrow \text{Prune}(B_x, \neg \varphi(x, b)) \]

Repeat until it fails to find a counterexample in step 1 or reaches a fixedpoint.
Solving Exist-forall Formulas

Preliminary Results

W / Wavy Function (#165 in [40])

\[
f_{165}(x_1, x_2) = 1 - \frac{1}{2} \sum_{i=1}^{2} \cos(10 \cdot x_i) e^{-\frac{x_i^2}{2}}
\]

subject to \(-3 \leq x_1, x_2 \leq 3\).
Solving Exist-forall Formulas

Preliminary Results

Salomon Function (#74 in [40])

\[ f_{110}(x_1, x_2) = 1 - \cos\left(2\pi \sqrt{x_1^2 + x_2^2}\right) + 0.1 \sqrt{x_1^2 + x_2^2} \]

subject to \(-100 \leq x_1, x_2 \leq 100\).
Solving Exist-forall Formulas

Proposed Work

• Prove that the pruning algorithms terminate.

• Prove that the pruning algorithms are well-defined.

• Finish the implementation, run experiments.
Time Line & Summary of Proposed Work
Timeline & Summary of Proposed Work

SAT-driven Branch-and-Prune

- Prove SAT+ICP algorithm terminates.
- Prove correctness of SAT+ICP algorithm. The outputs from naive ICP and SAT+ICP should be identical.
- Show that SAT+ICP algorithm outperforms naive ICP.
- Use Boxes/LDD data structure to implement Clause Manager and check the performance gain.

Solving Exist-forall Formulas

- Prove that the pruning algorithms terminate.
- Prove that the pruning algorithms are well-defined.
- Finish the implementation, run experiments.

Plan to finish all before the beginning of Fall semester 2016.
Thank you