dReal & dReach
THE DELTA DECISION TOOLS

Soonho Kong / Sicun Gao / Edmund Clarke

2013/11/22, CMACS/AVACS Workshop
http://dreal.cs.cmu.edu
SMT Problem

- Complexity results of **non-linear** arithmetic over the **reals**
  - **Decidable** if $\phi$ only contains **polynomials** [Tarski51]
  - **Undecidable** if $\phi$ contains trigonometric functions
- **Real-world problems** contain **complex nonlinear functions** (sin, exp, ODEs)
Decision Problem

Standard Form

\[ \phi := \exists^1 x \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k} f_{ij}(x) = 0 \]
δ-decision Problem

\[ \phi^\delta := \exists^I x \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k} |f_{ij}(x)| \leq \delta \]
\( \delta \)-decision Problem

- **UNSAT**: \( \phi^\delta \) is unsatisfiable
- **\( \delta \)-SAT**: \( \phi^\delta \) is satisfiable
- **Decidable** [LICS'12, IJCAR'12]
  - NP-complete: \( \mathcal{F} = \{ +, \times, \exp, \sin, \ldots \} \)
  - PSPACE-complete: \( \mathcal{F} = \{ \text{ODEs with P-computable rhs} \} \)
\( \delta \)-decision Problem

\[ \phi^\delta : UNSAT \implies \phi : UNSAT \]
$\delta$-decision Problem

$\phi^\delta : SAT \implies \phi : SAT \lor \phi : UNSAT$
$\delta$-decision Problem

$\phi^\delta : SAT \implies \phi : SAT \lor \phi : UNSAT$
δ-decision Problem

May find a smaller $\delta' < \delta$ such that

$\phi^{\delta'} : UNSAT \implies \phi : UNSAT$
$\delta$-decision Problem

$\phi^{\delta'} : SAT$ with a **reasonably small** $\delta'$

may indicate a **robustness** problem of the system in verification.
"Small perturbation on the system may **violate** safety properties"
dReal

- **δ-complete** SMT solver
- Can handle various **nonlinear real functions** such as sin, cos, tan, arcsin, arccos, arctan, log, exp, . . .
- Can handle **ODEs** (Ordinary Differential Equations)
- **Open-source**: [http://dreal.cs.cmu.edu](http://dreal.cs.cmu.edu)
Design of **dReal**

General **DPLL(T)** Framework

- SAT Solver: provide Boolean abstraction
- Theory Solver **T**: check \( T \)-satisfiability of assignment.
Design of dReal

**DPLL(ICP) Framework**
- **ICP** (Interval Constraint Propagation) solver
- Uses "**Branch & Prune**" algorithm
ICP: Pruning

\[ \exists x, y \in [0.5, 1.0] : y = \sin(x) \land y = \tan(x) \]

ANSWER: UNSAT
ICP: Pruning

\[ \exists x, y \in [0.5, 1.0] : y = \sin(x) \wedge y = \tan(x) \]
∃x, y ∈ [0.5, 1.0] : y = \sin(x) \land y = \tan(x)

\[ y' = y \cap \sin(x) \]
\[ = [0.5, 1.0] \cap \sin([0.524, 1.0]) \]
\[ = [0.5, 1.0] \cap [0.5, 0.841] \]
\[ = [0.5, 0.841] \]
ICP: Pruning

$$\exists x, y \in [0.5, 1.0] : y = \sin(x) \land y = \tan(x)$$

$$x' = x \cap \sin^{-1}(y)$$

$$= [0.5, 1.0] \cap \sin^{-1}([0.5, 1.0])$$

$$= [0.5, 1.0] \cap [0.524, 1.570]$$

$$= [0.524, 1.0]$$
ICP: Pruning

\[ \exists x, y \in [0.5, 1.0] : y = \sin(x) \land y = \tan(x) \]

\[ x : [0.524, 1.0], y : [0.5, 0.841] \]
ICP: Pruning

\[ \exists x, y \in [0.5, 1.0] : y = \sin(x) \land y = \tan(x) \]

\[ x' = x \cap \tan^{-1}(y) \]
\[ = [0.524, 1] \cap \tan^{-1}([0.5, 0.841]) \]
\[ = [0.524, 1] \cap [0.546, 1.117] \]
\[ = [0.546, 1.0] \]
ICP: Pruning

\[ \exists x, y \in [0.5, 1.0] : y = \sin(x) \land y = \tan(x) \]

\[ y' = y \cap \tan(x) \]
\[ = [0.5, 0.841] \cap \tan([0.546, 1.0]) \]
\[ = [0.5, 0.841] \cap [0.5, 1.0] \]
\[ = [0.5, 0.785] \]
ICP: Pruning

\[ \exists x, y \in [0.5, 1.0] : y = \sin(x) \land y = \tan(x) \]

\[ x : [0.524, 1.0], y : [0.5, 0.785] \]
ICP: Pruning

\[ \exists x, y \in [0.5, 1.0] : y = \sin(x) \land y = \tan(x) \]

ANSWER: UNSAT
ICP: Branching

- Divide the search space and try each one
- **Stop** when the size of box is smaller than $\varepsilon$:
  \[ |B| < \varepsilon \]
ICP: Branching

Graph for $\sin(x)$, $\arctan(x)$

Answer: SAT
ICP: Branching

\[ \exists x \in [0.5, 2.0], y \in [0.0, 2.0] : y = \sin(x) \land y = \tan(x) \]

\[ \epsilon = 0.001 \]

ANSWER: \( \delta \text{-SAT} \)
ICP: Branching

\( \exists x \in [0.5, 2.0], y \in [0.0, 2.0] : y = \sin(x) \land y = \tan(x) \)

\( \epsilon = 0.001 \)

**ANSWER:** \( \delta\text{-SAT}, x = [1.556, 1.557], y = [1.000, 1.000] \)
ICP: Branching

Input:
\[ \sin(x) = \tan^{-1}(x) \]
\( \tan^{-1}(x) \) is the inverse tangent function

Alternate form:
\[ \frac{1}{2} i e^{-i x} - \frac{1}{2} i e^{i x} = \frac{1}{2} i \log(1 - i x) - \frac{1}{2} i \log(1 + i x) \]
\( \log(x) \) is the natural logarithm

Integer solution:
\[ x = 0 \]

Numerical solution:
\[ x \approx \pm 1.55708581552472\ldots \]
Algorithm 1: Theory Solving in DPLL(ICP)

\textbf{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 \[ B_0^0 = I_1^0 \times \cdots \times I_n^0, \] box stack \( S = \emptyset \), and precision \( \delta \in \mathbb{Q}^+ \).

\textbf{output}: \( \delta \)-sat, or unsat with learned conflict clauses.

1. \( S.\text{push}(B_0) \);
2. \textbf{while} \( S \neq \emptyset \) \textbf{do}
3. \hspace{1em} \( B \leftarrow S.\text{pop}() \);
4. \hspace{1em} \textbf{while} \exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i) \textbf{ do}
5. \hspace{2em} \text{//Pruning without branching, used as the assert() function.}
6. \hspace{2.5em} 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} \textbf{if} \( B \neq \emptyset \) \textbf{then}
10. \hspace{2em} \textbf{if} \exists 1 \leq i \leq n, |I_i| \geq \epsilon \textbf{ then}
11. \hspace{3em} \{B_1, B_2\} \leftarrow \text{Branch}(B, i); \text{//Splitting on the intervals}
12. \hspace{3em} S.\text{push}({B_1, B_2});
13. \hspace{2em} \textbf{else}
14. \hspace{3em} \text{return} \( \delta \)-sat; \text{//Complete check() is successful.}
15. \hspace{2em} \textbf{end}
16. \hspace{1em} \textbf{end}
17. \textbf{return} \text{unsat};
**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 \( B^0 = I_1^0 \times \cdots \times I_n^0 \), box stack \( S = \emptyset \), and precision \( \delta \in \mathbb{Q}^+ \).

**output**: \( \delta \)-sat, or unsat with learned conflict clauses.

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

\textbf{input:} A conjunction of theory atoms, seen as constraints, \(c_1(x_1, \ldots, x_n), \ldots, c_m(x_1, \ldots, x_n)\), the initial interval bounds on all variables \(B^0 = I_1^0 \times \cdots \times I_n^0\), box stack \(S = \emptyset\), and precision \(\delta \in \mathbb{Q}^+\).

\textbf{output:} \(\delta\)-sat, or unsat with learned conflict clauses.

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

\textbf{return} unsat:
ICP in dReal

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 \(B^0 = I^0_1 \times \cdots \times I^0_n\), box stack \(S = \emptyset\), and precision \(\delta \in \mathbb{Q}^+\).

output: \(\delta\)-sat, or unsat with learned conflict clauses.

1. \(S\).push\((B_0)\);
2. \textbf{while} \(S \neq \emptyset\) \textbf{do}
3. \hspace{1em} \(B \leftarrow S\).pop();
4. \hspace{2em} \textbf{while} \(\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)\) \textbf{do}
5. \hspace{3em} //Pruning without branching, used as the assert() function.
6. \hspace{4em} \(B \leftarrow \text{Prune}(B, c_i)\);
7. \hspace{2em} \textbf{end}
8. \hspace{1em} //The \(\epsilon\) below is computed from \(\delta\) and the Lipschitz constants of
9. \hspace{1em} functions beforehand.
10. \hspace{2em} \textbf{if} \(B \neq \emptyset\) \textbf{then}
11. \hspace{3em} \textbf{if} \(\exists 1 \leq i \leq n, |I_i| \geq \epsilon\) \textbf{then}
12. \hspace{4em} \{B_1, B_2\} \leftarrow \text{Branch}(B, i); //Splitting on the intervals
13. \hspace{5em} S\).push\((\{B_1, B_2\})\);
14. \hspace{3em} \textbf{else}
15. \hspace{4em} \textbf{return} \(\delta\)-sat; //Complete check() is successful.
16. \hspace{3em} \textbf{end}
17. \hspace{1em} \textbf{end}
18. \textbf{return} unsat;
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 \( B^0 = I_1^0 \times \cdots \times I_n^0 \), box stack \( S = \emptyset \), and precision \( \delta \in \mathbb{Q}^+ \).

**output**: \( \delta \)-sat, or unsat with learned conflict clauses.

1. \( S \).push\((B_0)\);
2. while \( S \neq \emptyset \) do
3.  \( B \leftarrow S \).pop();
4.  while \( \exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i) \) do
5.     //Pruning without branching, used as the assert() function.
6.     \( B \leftarrow \text{Prune}(B, c_i) \);
7.  end
8. //The \( \varepsilon \) below is computed from \( \delta \) and the Lipschitz constants of functions beforehand.
9. if \( B \neq \emptyset \) then
10.    if \( \exists 1 \leq i \leq n, |I_i| \geq \varepsilon \) then
11.       \{\( B_1, B_2 \)\} \leftarrow \text{Branch}(B, i); //Splitting on the intervals
12.       S.push({\( B_1, B_2 \)});
13.    else
14.       return \( \delta \)-sat; //Complete check() is successful.
15. end
16. end
17. end
18. return unsat;
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 \( B^0 = I_1^0 \times \cdots \times I_n^0 \), box stack \( S = \emptyset \), and precision \( \delta \in \mathbb{Q}^+ \).

*output*: \( \delta \)-sat, or unsat with learned conflict clauses.

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

An ODE system

\[
\frac{d\vec{x}}{dt} = \vec{f}(\vec{x}, t)
\]

when put in Picard–Lindelöf form:

\[
\vec{x}_t = \vec{x}_0 + \int_0^t \vec{f}(\vec{x}, s)ds
\]

is seen as a **constraint** between \(\vec{x}_0, \vec{x}_t,\) and \(t\).
ODE Pruning
Starting with big intervals for

\[ \vec{x}_t, \vec{x}_0, t \]

use the ODE constraints to find smaller intervals for them.
Forward Pruning (on $X_t$)
Forward Pruning (on $X_t$)
Forward Pruning (on $X_t$)
Forward Pruning (on $X_t$)
Forward Pruning (on $X_t$)
Backward Pruning (on $X_0$)
Backward Pruning (on $X_0$)
Backward Pruning (on $X_0$)
Time Pruning (on $T$)
Time Pruning (on $T$)
Time Pruning (on $T$)
Pruning with Invariant
Pruning with Invariant
Pruning with Invariant
Pruning with Invariant
dReach

- Tool for safety verification of **hybrid systems**.
- Handle general hybrid systems with nonlinear differential equations and complex discrete mode-changes.
- Performs bounded delta-complete reachability analysis and uses dReal as a computation engine.
## Experimental Results: Hybrid System Benchmark

<table>
<thead>
<tr>
<th>Problem</th>
<th># of Mode</th>
<th># of Unrolling Depth</th>
<th># of ODEs</th>
<th># of variables</th>
<th>Eps</th>
<th>Result</th>
<th>Time</th>
<th>Size of Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atrial Filbrillation</td>
<td>4</td>
<td>3</td>
<td>20</td>
<td>44</td>
<td>0.001</td>
<td>SAT</td>
<td>43.10</td>
<td>90K</td>
</tr>
<tr>
<td>Atrial Filbrillation</td>
<td>8</td>
<td>7</td>
<td>40</td>
<td>88</td>
<td>0.001</td>
<td>SAT</td>
<td>698.86</td>
<td>20M</td>
</tr>
<tr>
<td>Atrial Filbrillation</td>
<td>8</td>
<td>23</td>
<td>120</td>
<td>246</td>
<td>0.001</td>
<td>SAT</td>
<td>4528.13</td>
<td>59M</td>
</tr>
<tr>
<td>Atrial Filbrillation</td>
<td>8</td>
<td>31</td>
<td>160</td>
<td>352</td>
<td>0.001</td>
<td>SAT</td>
<td>8485.99</td>
<td>78M</td>
</tr>
<tr>
<td>Atrial Filbrillation</td>
<td>8</td>
<td>47</td>
<td>240</td>
<td>528</td>
<td>0.001</td>
<td>SAT</td>
<td>15740.41</td>
<td>117M</td>
</tr>
<tr>
<td>Atrial Filbrillation</td>
<td>8</td>
<td>55</td>
<td>280</td>
<td>616</td>
<td>0.001</td>
<td>SAT</td>
<td>19989.59</td>
<td>137M</td>
</tr>
<tr>
<td>Prostate Cancer</td>
<td>2</td>
<td>2</td>
<td>15</td>
<td>36</td>
<td>0.005</td>
<td>SAT</td>
<td>345.84</td>
<td>3.1M</td>
</tr>
<tr>
<td>Prostate Cancer</td>
<td>2</td>
<td>2</td>
<td>15</td>
<td>36</td>
<td>0.002</td>
<td>SAT</td>
<td>362.84</td>
<td>3.1M</td>
</tr>
<tr>
<td>Electronic Oscillator</td>
<td>3</td>
<td>2</td>
<td>18</td>
<td>42</td>
<td>0.01</td>
<td>SAT</td>
<td>52.93</td>
<td>998K</td>
</tr>
<tr>
<td>Electronic Oscillator</td>
<td>3</td>
<td>2</td>
<td>18</td>
<td>42</td>
<td>0.001</td>
<td>SAT</td>
<td>57.67</td>
<td>847K</td>
</tr>
<tr>
<td>Electronic Oscillator</td>
<td>3</td>
<td>11</td>
<td>72</td>
<td>168</td>
<td>0.01</td>
<td>UNSAT</td>
<td>7.75</td>
<td>--</td>
</tr>
<tr>
<td>Bouncing Ball</td>
<td>2</td>
<td>10</td>
<td>22</td>
<td>66</td>
<td>0.01</td>
<td>SAT</td>
<td>0.25</td>
<td>123K</td>
</tr>
<tr>
<td>Bouncing Ball</td>
<td>2</td>
<td>20</td>
<td>42</td>
<td>126</td>
<td>0.01</td>
<td>SAT</td>
<td>0.57</td>
<td>171K</td>
</tr>
<tr>
<td>Bouncing Ball</td>
<td>2</td>
<td>20</td>
<td>42</td>
<td>126</td>
<td>0.001</td>
<td>SAT</td>
<td>2.21</td>
<td>168K</td>
</tr>
<tr>
<td>Bouncing Ball</td>
<td>2</td>
<td>40</td>
<td>82</td>
<td>246</td>
<td>0.01</td>
<td>UNSAT</td>
<td>0.27</td>
<td>----</td>
</tr>
<tr>
<td>Bouncing Ball</td>
<td>2</td>
<td>40</td>
<td>82</td>
<td>246</td>
<td>0.001</td>
<td>UNSAT</td>
<td>0.26</td>
<td>----</td>
</tr>
<tr>
<td>Decay Model</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td>24</td>
<td>0.1</td>
<td>SAT</td>
<td>30.84</td>
<td>72K</td>
</tr>
</tbody>
</table>
Experimental Results: Hybrid System Benchmark

Bouncing Ball

Click and drag above to zoom / pan the data
Experimental Results: Hybrid System Benchmark

- Including **Atrial Filbrillation** Model: (R. Groso et al, "From Cardiac Cells to Genetic Regulatory Networks", CAV'11)
Experimental Results: Hybrid System Benchmark
dReach Demo
Conclusion

- **dReal** is an δ-complete SMT solver
- **dReach** is a tool for safety verification of hybrid systems.
- Support nonlinear real functions such as sin, cos, tan, arcsin, arccos, arctan, log, exp, ...
- Handle ODEs (Ordinary Differential Equations)
- Based on DPLL(I) framework
- **Scalable** with our experiments
- **Open-source**: available at http://dreal.cs.cmu.edu