δ-Complete Decision Procedure and dReal

Damien Zufferey
MIT CSAIL

ARSBM 2016, 20 Sept 2016

Based on the work of Sicun Gao and Soonho Kong

Outline

- Interval constraints propagation (ICP)
  - Branch and Prune Algorithm
  - Completeness
  - dReal Example
- Adding ODEs
  - dReach Example
  - SMT encoding
- dReal Tricks

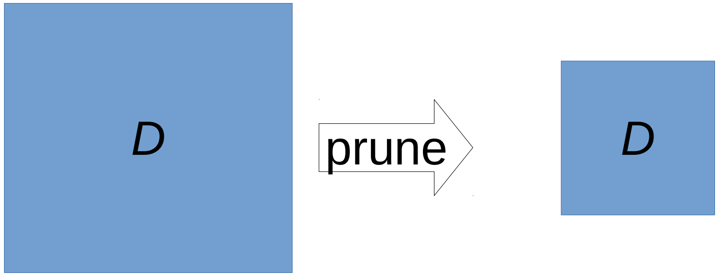
Interval Constraints Propagation

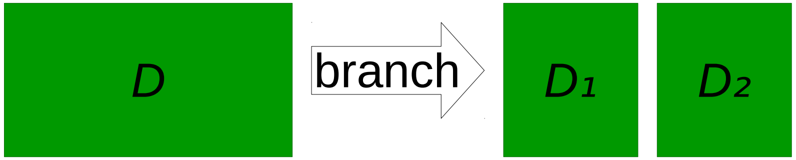
- Search for a solution using
  - Pruning: interval arithmetic to prune the search space.
  - Branching: when pruning is stuck, split the domain of a variable and continue recursively.
- Interval arithmetic on double precision numbers
  - Rounding errors taken into account
  - dReal uses IBEX and CAPD libraries
- Use $\delta>0$ to guarantee the termination

Branch and Prune ICP

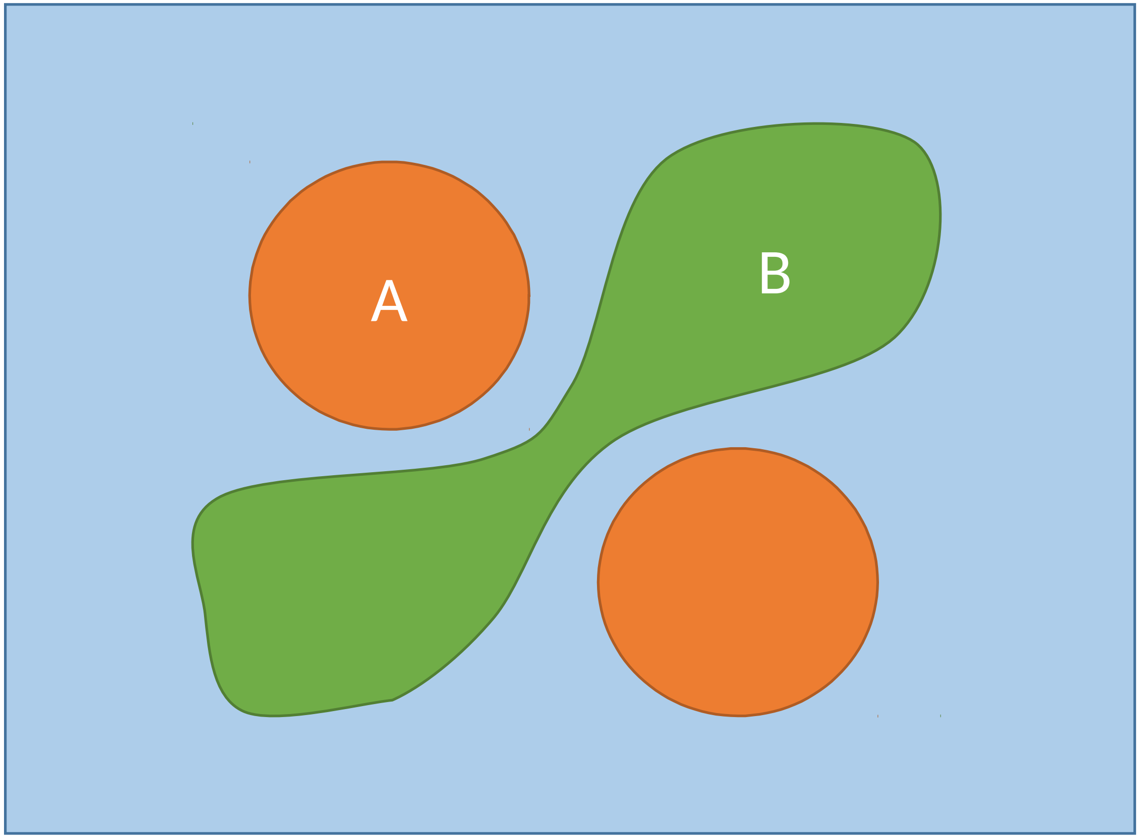
Algorithm 1 ICP($c_1, ..., c_m, \vec{D} = D_1 \times \cdots \times D_n, \delta$)

S.push($\vec{D}$)
while $S \neq \emptyset$ **do**
  $\vec{D} \leftarrow S$.pop()
  while $\exists 1 \leq i \leq m, \vec{D} \neq_\delta \text{Prune}(\vec{D}, c_i)$ **do**
    $\vec{D} \leftarrow \text{Prune}(\vec{D}, c_i)$
  end while
  if $\vec{D} \neq \emptyset$ **then**
    if $\exists 1 \leq i \leq n, |D_i| \geq \varepsilon$ **then** ▷ ε is some computable factor of δ
      $\{\vec{D}_1, \vec{D}_2\} \leftarrow \text{Branch}(\vec{D}, i)$
      S.push($\vec{D}_1$)
      S.push($\vec{D}_2$)
    else
      return sat
    end if
  end if
end while
return unsat



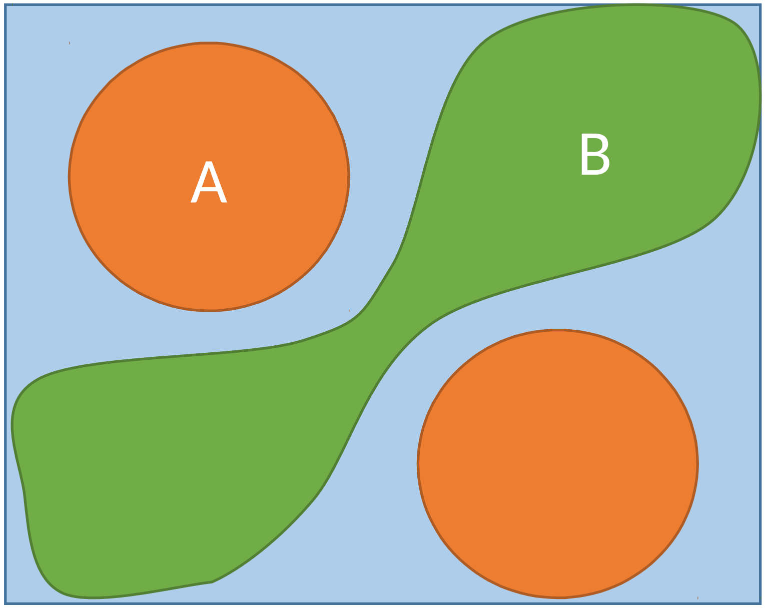


Branch-and-Prune Example



Branch-and-Prune Example

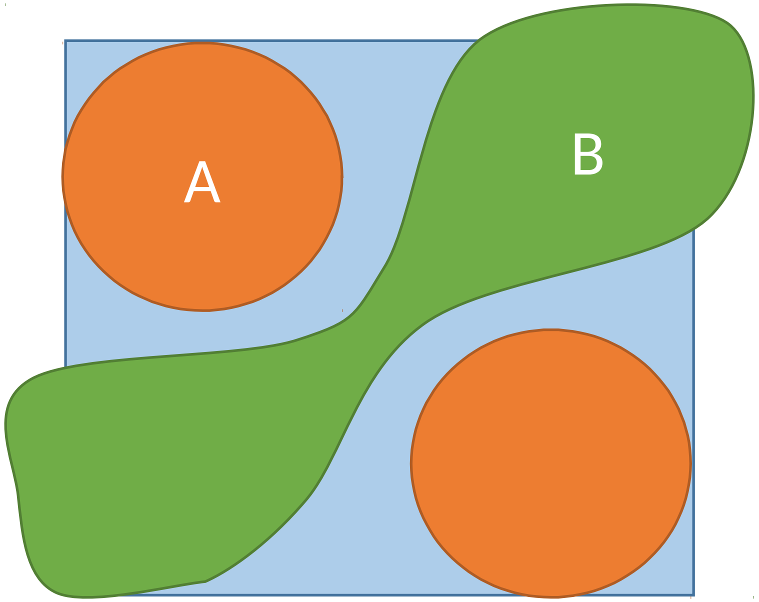
Prune by **B**



Branch-and-Prune Example

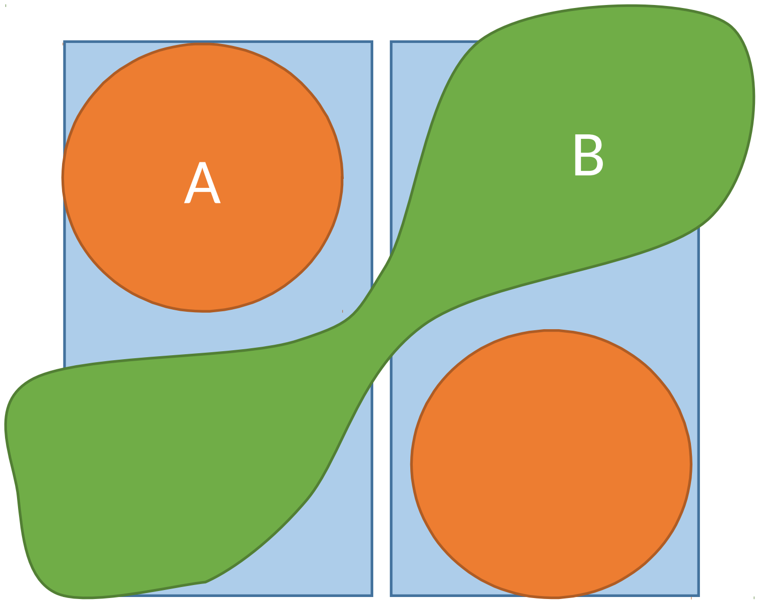
Prune by **B**

Prune by **A**



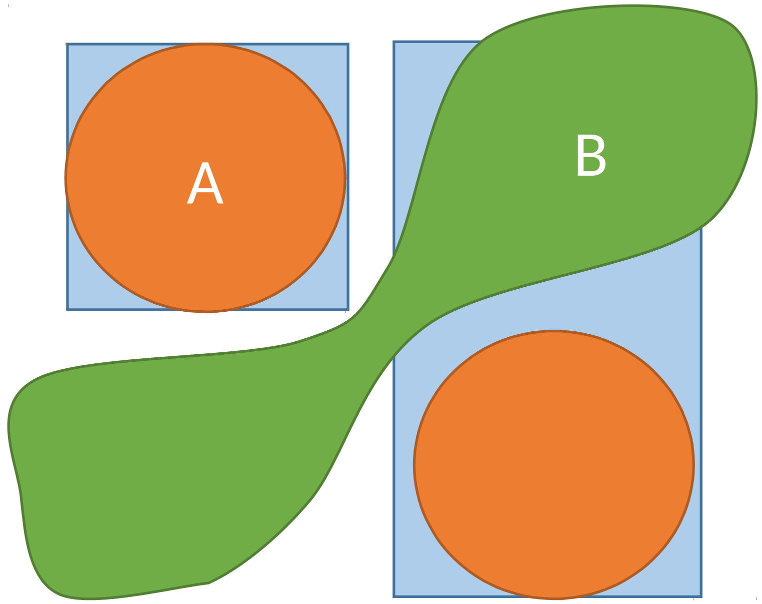
Branch-and-Prune Example

Prune by **B**
Prune by **A**
Branch



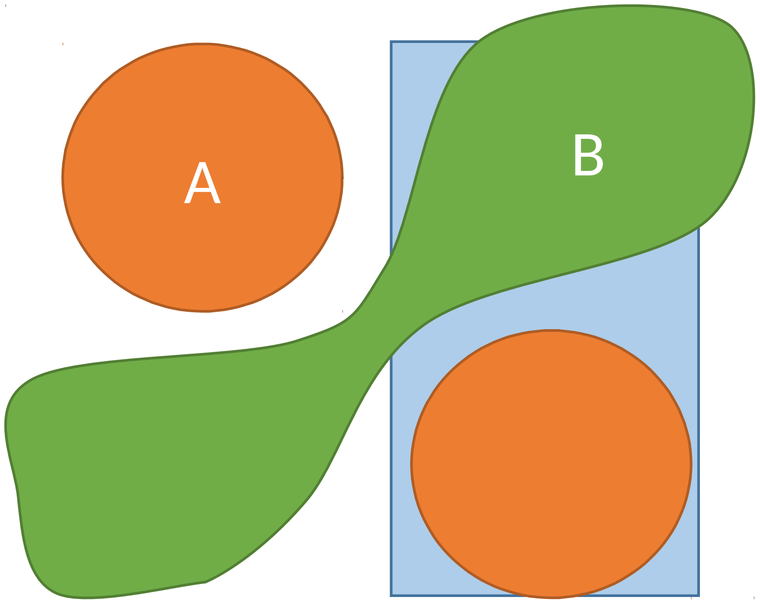
Branch-and-Prune Example

Prune by **B**
Prune by **A**
Branch
Prune by **A**



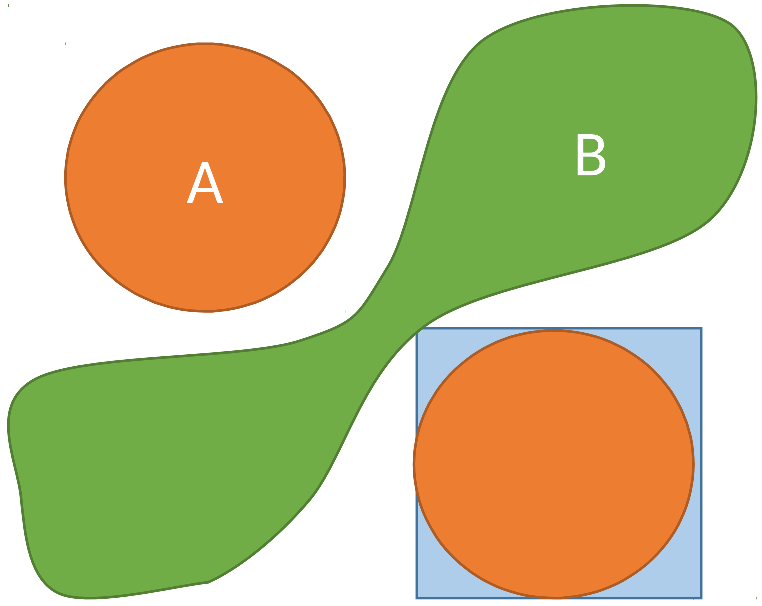
Branch-and-Prune Example

Prune by **B**
Prune by **A**
Branch
Prune by **A**
Prune by **B**



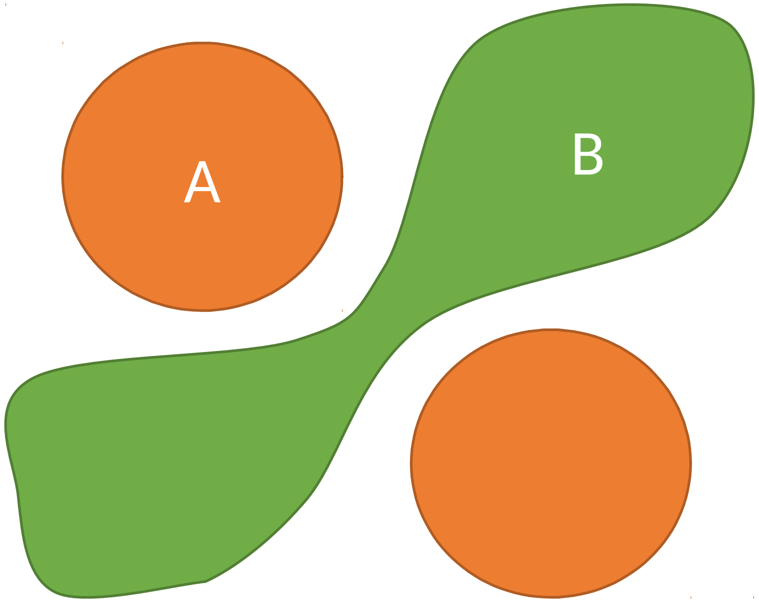
Branch-and-Prune Example

Prune by **B**
Prune by **A**
Branch
Prune by **A**
Prune by **B**
Prune by **A**



Branch-and-Prune Example

Prune by **B**
Prune by **A**
Branch
Prune by **A**
Prune by **B**
Prune by **A**
Prune by **B**



Completeness

- δ-satisfiability is NP (PSpace with ODE).
- Idea:
  - If we can *guess a small enough box* containing the solution, we can check it in polynomial time using interval arithmetic.
  - If the problem is unsatisfiable, we need to explore a potentially exponential number of small boxes and show that all of them are empty.
- Takeaway message:

  Nonlinear theories over the reals are *just* polynomially harder than SAT.

dReal

- Description: http://dreal.github.io/
- Getting the tool: https://github.com/dreal/dreal3
- GPL3 license
- Runs natively on Linux and Mac
- Runs on Windows via Docker

dReal Frontends

- SMT2

```
(set-logic QF_NRA)
(declare-fun x () Real)
(declare-fun y () Real)
(assert (< 2.4 x))
(assert (< x 2.6))
(assert (< -10.0 y))
(assert (< y 10.0))
(assert
    (and
        (= y (cos x))
    )
)
(check-sat)
(exit)
```

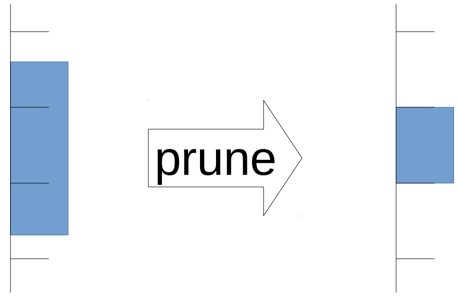
- dr

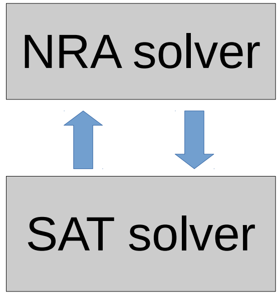
```
var:
    [2.4, 2.6] x;
    [-10, 10] y;
ctr:
    y = cos(x);
```

dReal Example

What We Support

- Types: Real, Int, Bool
  - Int are handled in the ICP by a special contractor.
  - Bool are handled before the ICP by a SAT solver.





- Functions:

  polynomials, trigonometric functions, logarithms, …

  (We will discuss very soon about the ODEs.)

ODEs and dReach

- dReal support ODEs directly in the SMT2 interface with a `QF_NRA_ODE` logic but the notation is non-standard.
- The dReach tool is much more user-friendly.
- dReach is a BMC that generates a dReal query from an hybrid automata

dReach Syntax

dReach Syntax

```
[0, 20] x;
[-9.8] g;
[-100, 100] v;
[0, 10] time;
```

dReach Syntax

```
[0, 20] x;
[-9.8] g;
[-100, 100] v;
[0, 10] time;


{ mode 1;
  invt:
        (v <= 0);
        (x >= 0);
  flow:
        d/dt[x] = v;
        d/dt[v] = g;
  jump:
        (x = 0) ==>
        @2 (and (x' = x)
                (v' = (0 - v)));
}

{ mode 2;
  invt:
        (v >= 0);
        (x >= 0);
  flow:
        d/dt[x] = v;
        d/dt[v] = g;
  jump:
        (v = 0) ==>
        @1 (and (x' = x)
                (v' = v));
}
```

dReach Syntax

```
[0, 20] x;
[-9.8] g;
[-100, 100] v;
[0, 10] time;


{ mode 1;
  invt:
        (v <= 0);
        (x >= 0);
  flow:
        d/dt[x] = v;
        d/dt[v] = g;
  jump:
        (x = 0) ==>
       @2 (and (x' = x)
                (v' = (0 - v)));
}
```

```
{ mode 2;
  invt:
        (v >= 0);
        (x >= 0);
  flow:
        d/dt[x] = v;
        d/dt[v] = g;
  jump:
        (v = 0) ==>
       @1 (and (x' = x)
                (v' = v));
}

init:
    @1  (and (x = 10) (v = 0));
goal:
    @2 (and (x = 1) (v >= 1));
```

dReach Example

SMT Encoding (1)

- Variables

```
(declare-fun mode_i () Real)
(declare-fun time_i () Real)
(declare-fun x_i_0 () Real)
(declare-fun x_i_t () Real)
(declare-fun v_i_0 () Real)
(declare-fun v_i_t () Real)
```

- Mode invariants

```
(assert (and
        (forall_t 1 [0 time_i] (>= x_i_t 0) (<= v_i_t 0))
        (forall_t 2 [0 time_i] (>= x_i_t 0) (>= v_i_t 0))
))
```

SMT Encoding (2)

- Flow declaration

```
(declare-fun x () Real)
(declare-fun v () Real)
(define-ode flow_1 (
        (= d/dt[x] v)
        (= d/dt[v] g) ))
(define-ode flow_2 (
        (= d/dt[x] v)
        (= d/dt[v] g) ))
```

- Jump conditions

```
(assert (or (and (= mode_i 1) (= mode_j 2) (= x_i_t 0)
                 (= x_j_0 x_i_t) (= v_j_0 (- v_i_t)))
            (and (= mode_i 2) (= mode_j 1) (= v_i_t 0)
                 (= x_j_0 x_i_t) (= v_j_0 v_i_t))))
```

SMT Encoding (3)

- Connecting the flows

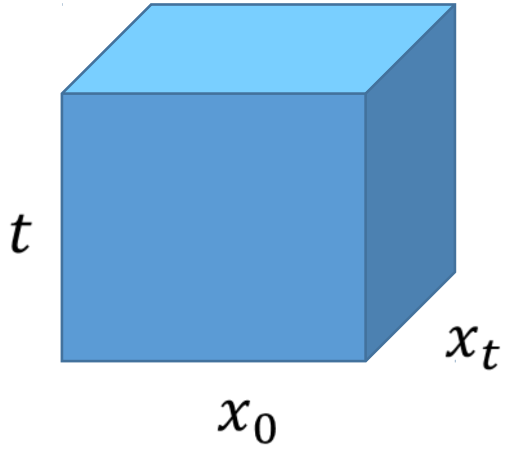
```
(assert (or
    (and (= mode_i 1)
         (= [x_i_t v_i_t] (integral 0. time_i [x_i_0 v_i_0] flow_1)))
    (and (= mode_i 2)
         (= [x_i_t v_i_t] (integral 0. time_i [x_i_0 v_i_0] flow_2)))
))
```

- Other elements
  - Initial and final conditions
  - Bounds for all the variables
  - ...

ODEs, dReal, and Completeness

$$x_t = x_0 + \int_0^t f(x)\,dx \ \wedge\ 0 \le t \le 2$$

is just a pruning operator over the domain



dReal Tricks

- Julia bindings, C API, etc.
- Precision (δ)
  - Option: `--precision 0.1`
  - In SMT file: `(set-option :precision 0.1)`
- Model Generation
  - Option: `--model`
- Polytope contractor
  - Option: `--polytope`
- Branching heuristics
  - Options: `--gradbranch`, `--scoring-icp`

What Comes Next

- More efficient search heuristics (!!!)
- ∃∀ formula
- More parallelism
- ...

Conclusion

- dReal is an SMT solver for nonlinear theories over the reals
- dReach is a bounded model checker for hybrid systems. dReach uses dReal as backend.
- If you have questions, contact us by email, open issues on github. Pull-requests on github are also welcome.