White Paper

vega: Theorem Solver To Find All Satisfiable Values

Tomori Nao (@K_atc)*

2020, March

1 About this document

This white paper describes concept and blueprint of *vega*.
vega is open source software and available on GitHub **TODO:** link.

2 Concept of vega

vega is yet another theorem solver and answers all values which satisfies given constraints on each variable as a model (in sat solver, this behavior is called as all sat solver).

For example, if there're variable $x \in \{a, b, c\}$ and constraints $x = a \vee x \neq c$, vega's solution is $x = \{a, b\}$. This means x is a or b. Note that general SMT solver (such as z3) 's solution $x = a$ does not covers all satisfiable values and such solvers requires much more time to obtain another solution.

2.1 Ground truth

We assume the following assumptions are enough to solve constraints.

- Solution space is finite.
- A model monotonically decreases by solving each constraint expression.
  - If solve constraint $A = A_1 \wedge A_2 \wedge \cdots \wedge A_n$ step by step: $\mathrm{model}(A_1) = M_1,\ \mathrm{model}(A_1 \wedge A_2) = M_2, \cdots,\ \mathrm{model}(A_1 \wedge \cdots \wedge A_n) = M_n$, models should satisfy $M_1(v) \supseteq M_2(v) \supseteq \cdots \supseteq M_n(v)$ on each variables in given constraint A.
- *Tighter* model is the one to be treated as solution of given constraint.
  - Tightness of model M satisfies constraint A is defined as equation 1.
    - Function model() (defined in section 4) solves constraints A and returns a model A as solution.
  - On constraint A, Model M_1 is tighter than M_2 if only if $\mathrm{Tightness}(M_1, A) > \mathrm{Tightness}(M_2, A)$.

$$\mathrm{Tightness}(M, A) = \max_X |A'| \text{ where } X = (A' \subseteq A \wedge \mathrm{model}(A') = \mathrm{model}(A)) \tag{1}$$

- Once a constraint is evaluated, cannot make it undone.
  - This means *vega* cannot adopt back track.
  - Semantics described in section 6 does not care back track.
  - This assumption is made to speed up solving and for simple to implement.

*shiftx1026@gmail.com, https://twitter.com/K_atc/

3 Use cases

3.1 Abstract Interpretation

TODO: What is abstract interpretaion

TODO: Example usage

4 Algorithms

Function model() solves constraint A and returns model M. Algorithm 1 is algorithm of model(). Function model1() and model2() are described in section 6.2 and 6.4, respectively。

Function `model(`A`)`
- **Input:** Constraint expression
- **Result:** Model
- $R_0, M_0, P \leftarrow \text{model1}(A)$
- $R, M \leftarrow \text{model2}(P, R_0, M_0)$
- **return** M

end

Algorithm 1: Getting a model M of constraints A

5 Notations and definitions

- $\mathcal{D}$ = Domain (All possible values)
- a = constant value ($a \in \mathcal{D}$)
- $\boldsymbol{a}$ = Set of values ($\boldsymbol{a} \subseteq \mathcal{D}$)
- S = Sort; Subset of domain ($S \subseteq \mathcal{D}$)
- V = Set of variables (Variables appeared in all constraints) ($V \subseteq \mathcal{D}$)
- v = Variable ($v \in V$)
- $R : v \to v^{'}$ = Variables equality. This is a directed graph of reference to denote eqality $v = v'$ and satisfies following assertions:
  - Requirement 1 (see equation 2): All nodes of directed graph R does not have more than 2 parents.
  - Requirement 2 (see equation 3): Directed graph R does not have closed loops among dirrefent nodes

$$\forall x, y(R(x) \neq R(y) \implies x \neq y) \tag{2}$$

$$\forall x, y(x \neq y \wedge R(x) = y \implies R(y) \neq x) \tag{3}$$

- $R.V$: Defined source of relations in R (i.e. all index of map R)
- $M = \bigcup_{v \in V} R^*(v) \mapsto \boldsymbol{a}$ = Model
- $M.V$: Variables defined in M
- $R^*(v) = \begin{cases} v & v \notin R.V \vee R(v) = v \\ R(v) & v \in R.V \wedge R(v) \neq v \end{cases}$

- $M(v) = \begin{cases} \boldsymbol{a} \cap M(R(v)) & v \mapsto \boldsymbol{a} \in M \land R(v) \in M.V \\ \mathcal{D} & v \notin M.V \end{cases}$

- $|M(v)|=$ Size (length) of $M(v)$

- $R_1 \circ R_2 = \bigwedge_{v \in M_1.V \cup M.V} \begin{cases} v \to R_1^*(v) & R_1^*(v) = R_2^*(v) \lor R_2^*(v) = v \\ R_1^*(v) \to R_2^*(v) & R_1^*(v) \neq R_2^*(v) \land R_2^*(v) \neq v \end{cases}$ (Denote equality $v = v'$ as $v \to v'$ for visibility)

  - $R_1^*(v) \neq R_2^*(v)$ and $R_2^*(v) \neq v$ are to meet requirement 1 and 2 of R, respectively.

- $M_1 \cap M_2 = \bigwedge_{v \in M_1.V \cup M_2.V} M_1(v) \cap M_2(v)$

5.1 Proof tree

To denote semantics of model function, we introduce following proof tree. $\Gamma \models \Delta$ means that Δ is true if assumption Γ is true. Each value of variable in lower side is determined by calcuation result of uppper side.

$$\frac{R,\ M,\ P \models A, R_0, M_0}{\mathrm{f}(A) = R, M, P}$$

Each meaning of variable is:

- R, M = same as above
- P = post-constraint (logical expression). If $P = \top$, this term is ommited (Used in chaper 6.2)
- A = given constraint (logical expression)
- R_0 = inital variables equality. If $R_0 = \top$, this term is ommitted (Used in section 6.4)
- M_0 = inital model. If $M_0 = \phi$, this term is ommited (Used in section 6.4)
- f = Defined function name goes here

6 Semantics

6.1 Satisfiability

M is $\begin{cases} sat & \text{iff } \bigwedge_{v \in V} |M(v)| > 0 \\ unsat & \text{iff } \bigvee_{v \in V} |M(v)| = 0 \end{cases}$

If M is sat, M should be evaluated as true. If M is unsat, M should be evaluated as false.

6.2 Eq, Not, And, Or: semantics without pre-condition

In this section, the following proof treee shows a definition of function model1 which introdues model M satisfies constraint A and variables equality R.

$$\frac{R,\ M,\ P \models A}{\mathrm{model1}(A) = R, M, P}$$

The following is rules of calcuation of $R,\ M,\ P \models A$. Notaion P^Xdescribes P is evaluated in X context. Notaion P^{any}(referred to as P for simplicity) describes P is evaluated in any context.

$$\frac{R = v \to v' \models \top}{R,\ \phi,\ \top \models \mathrm{Eq}^{\mathrm{And}}(v, v')}\ (\text{eq-and-v})$$

$$\frac{M = \{v \mapsto \{a\}\} \models \top}{\top,\ M,\ \top \models \mathrm{Eq}(v, a)}\ (\text{eq-a})$$

$$\frac{M = \mathcal{D} - \{a\} \models \top}{\top,\ M,\ \top \models \mathrm{Not}(\mathrm{Eq}(v, a))}\ (\text{not-eq-a})$$

$$\frac{}{\top,\ \phi,\ \{\mathrm{If}(A_1, A_2, A_3)\} \models \mathrm{If}(A_1, A_2, A_3)}\ (\text{and-if})$$

$$\frac{}{\top,\ \phi,\ \{\mathrm{Implies}(A_1, A_2)\} \models \mathrm{Implies}(A_1, A_2)}\ (\text{and-implies})$$

$$\frac{R_1,\ M_1, P_1 \models A_1^{\mathrm{And}} \qquad R_2,\ M_2, P_2 \models A_2^{\mathrm{And}} \qquad \cdots \qquad R_n,\ M_n, P_n \models A_n^{\mathrm{And}}}{R_1 \circ \cdots \circ R_n,\ M_1 \cap \cdots \cap M_n, P_1 \wedge \cdots \wedge P_n \models \mathrm{And}(A_1, A_2, \cdots, A_n)}\ (\text{and})$$

$$\frac{R_1,\ M_1,\ \top \models A_1^{\mathrm{Or}} \qquad R_2,\ M_2,\ \top \models A_2^{\mathrm{Or}} \qquad \cdots \qquad R_n,\ M_n,\ \top \models A_n^{\mathrm{Or}}}{\top,\ M_1 \cup \cdots \cup M_n,\ \top \models \mathrm{Or}(A_1, A_2, \cdots, A_n)}\ (\text{or})$$

TODO: De-morgan's Law and Not(Not()) pattern.

Note that any assumptions $\mathit{\Gamma}$ satisfies $\top$(i.e. True).

$$\frac{}{\Gamma \models \top}\ (\text{top})$$

6.3 Function check(): judging satifibility without side-effects

The following is definition of function check which returns satisfibility s of A under assumptions R, M.

$$\frac{}{\mathrm{check}(A, R, M) \to s}$$

The following is rules of calcution of function check.

$$\frac{\mathrm{check}(\neg A, R, M) \to s}{\neg\mathrm{check}(A, R, M) \to s}\ (\text{check-neg})$$

$$\frac{}{\mathrm{check}(\mathrm{Eq}(v, a), R, M) \to a \in M(v)}\ (\text{check-eq-a})$$

$$\frac{}{\mathrm{check}(\mathrm{Not}(\mathrm{Eq}(v, a)), R, M) \to a \notin M(v)}\ (\text{check-not-eq-a})$$

$$\frac{\mathrm{check}(P_n, R, M) \to s_1 \qquad \mathrm{check}(P_2, R, M) \to s_2 \qquad \cdots \qquad \mathrm{check}(P_n, R, M) \to s_n}{\mathrm{check}(\mathrm{And}(P_1, P_2, \cdots, P_n)) \to s_1 \wedge \cdots \wedge s_n}\ (\text{check-and})$$

$$\frac{\mathrm{check}(P_n, R, M) \to s_1 \qquad \mathrm{check}(P_2, R, M) \to s_2 \qquad \cdots \qquad \mathrm{check}(P_n, R, M) \to s_n}{\mathrm{check}(\mathrm{Or}(P_1, P_2, \cdots, P_n)) \to s_1 \vee \cdots \vee s_n}\ (\text{check-or})$$

6.4 If, Implies: semantics with pre-condition

The following is definition of function model2 which introduces model M and variables equality R satisfies constraint A and pre-requirement R_0, M_0, P.

$$\frac{R,\ M \models A,\ R_0,\ M_0}{\text{model2}(A, R_0, M_0) = R, M}$$

The following is rules of calcuation of $R,\ M \models A,\ R_0,\ M_0$. Note that rules defined in section 6.2 are applied to undefined rules such as case $A = \text{And}(\text{A}_1, \text{A}_2, \text{A}_3)$.

$$\frac{R_1,\ M_1 \models A_1,\ R_0,\ M_0 \qquad R_2,\ M_2 \models A_2,\ R_0,\ M_0}{R_0 \circ R_1 \circ R_2,\ M_0 \cap M_1 \cap M_2 \models \text{check}(A_1, R_0, M_0) = \top \Rightarrow A_2,\ R_0,\ M_0} \text{(if-check-true)}$$

$$\frac{}{\top,\ \phi \models \text{check}(A_1, R_0, M_0) = \bot \Rightarrow A_2,\ R_0,\ M_0} \text{(if-check-false)}$$

$$\frac{R_1,\ M_1 \models \text{check}(A_1, R_0, M_0) \Rightarrow A_2,\ R_0,\ M_0 \qquad R_2,\ M_2 \models \neg\text{check}(A_1, R_0, M_0) \Rightarrow A_3,\ R_0,\ M_0}{R_0 \circ R_1 \circ R_2,\ M_0 \cap M_1 \cap M_2 \models \text{If}(A_1, A_2, A_3),\ R_0,\ M_0} \text{(if)}$$

$$\frac{R_1,\ M_1 \models \text{check}(A_1, R_0, M_0) \Rightarrow A_2}{R_0 \circ R_1 \circ R_2,\ M_0 \cap M_1 \cap M_2 \models \text{Implies}(A_1, A_2),\ R_0,\ M_0} \text{(implies)}$$

7 Limitations

7.1 Limited expression(s)

As you can see in section 6, there are constraint expressions that cannot be handled such as Or(Implies($\bullet$)).

7.2 Reordering problem

Order of constraints to solve is important because vega does not support back track. We call this problem as *Reordering problem*. Future work is determistic tactic of ordering constraints to avoid false positive unsat.

vega introduces two tactics called Simple2 and WithReorder as function model() to mitigate this problem. Details are described section below.

Both of tactics cannot satisfy constraint contains potential conflict such as R(z) $\wedge$ Implies($P(x), Q(y)$)$\wedge$ Implies($R(z), \neg P(x)$) (where P, Q, R are constraints which does not have If() or Implies()) bacause they cannot determine whether $P(x)$ should be true or not .

7.2.1 Simple2 tactic

Algorithm 2 describes algorithm of Simple2 tactic.

Function `model(`A`)`
  Input: Constraint expression
  Result: Model
  $R_0, M_0, P \leftarrow \text{model1}(A)$
  $R_1, M_1 \leftarrow \text{model2}(P, R_0, M_0)$
  $R, M \leftarrow \text{model2}(P, R_1, M_1)$
  return M
end

Algorithm 2: Simple2 tactic

Simple2 tactic can solve $P(y,x), Q(z), Q'(y), R(z,y)$ where $x,y,z \in V$[1] which has possible 2 models [2]. WithReorder tactic cannot solve this because of solving with order Q, Q', P, R).

Simple2 tactic assumes solving post-constraints P repeatedly tightens/minimizes model without unsat.

7.2.2 WithReorder tactic

Algorithm 3 describes algorithm of WithReorder tactic. This tactics prioritize constraints which has solved variables.

```
Function model(A)
    Input: Constraint expression
    Result: Model
    R_0, M_0, P ← model1(A)
    visited_variables ← get_visited_variables(A)
    for e ∈ P do
        cond_variables ← getcondv(e)
        if cond_variables ∩ visited_variables = cond_variables then
            R, M ← model2(P, R_0, M_0)
            visited_variables ← visited_variables ∪ getv(e)
        else
            P ← P ∧ e
        end
    end
    return M
end
```

Algorithm 3: WithReorder tactic

TODO: Describe get_visited_variables, getcondv, getv

8 Benchmarks

TODO: 100 万個の制約、10 万個の変数を目安の上限に、時間・メモリーをプロット。

[1]This is case of predessor variable y is constrained by successor constraint R. An example is shown as below

$$\text{Implies}(y \neq a, x \neq b) \wedge z \neq b \wedge y \neq b \wedge \text{Implies}(z \neq b, y \neq a) \quad \text{where } x, y, z \in V, a, b \in \mathcal{D}$$

[2]there're 2 way of order P, Q, Q', R and Q, Q', P, P and latter makes tighter model

Revision history

- 2020/03/22: First version