Matching Algorithms

Forum Teoretycznej Informatyki 2015

Warszawa, 31 stycznia 2015

Hans de Nivelle

Uniwersytet Wrocławski

Context

Theorem proving: We try to find logical proofs automatically. Frequently used calculi:

First order logic $\Rightarrow$ Resolution, Paramodulation, Superposition.

Propositional Logic $\Rightarrow$ Davis Putnam Loveland Logeman algorithm, enhanced with conflict learning.

In 2005-2006, I developed a calculus called geometric resolution. It is good at finding finite models, and reasonably good at finding proofs.

Context (2)

Since apprimately 2010, I try to extend geometric resolution to classical logic enhanced with partial functions. (Wie deelt door nul is een ...).

Assume that $X = 1,\ Y = 1$: It follows that
$X^2 - Y^2 = X^2 - X.Y \Rightarrow (X - Y)(X + Y) = (X - Y)X \Rightarrow$
$X + Y = X \Rightarrow 1 + 1 = 1.\ \ (QED)$

```
template< typename A >
void misbehave( )
{ std::list<A> lst1, lst2;
   if( lst1.front( ) == list2.front( ))
      std::cout << "first elements equal!\n";
   else
      std::cout << "first elements differ!";
}
```

Partial Classical Logic

In 2010-2012, I developed an enhancement of classical logic, with supports partial functions.

I am totally not going to speak about it, but it is a nice logic:

- Theorem Proving for Classical Logic with Partial Functions by Reduction to Kleene Logic. Journal of Logic and Computation, 2014.
- Classical Logic with Partial Functions, Journal of Automated Reasoning, 2011.

(Don't read the 2011 paper, because the logic has evolved)

Geometric Resolution

The theorem proving method in the 2014 paper is a 3-valued adaptation of:

- Geometric Resolution: A Proof Procedure Based on Finite Model search, (coauthored by Jia Meng), International Joint Conference on Automated Reasoning, 2006.

Geometric resolution has been implemented in a theorem prover, which was (creatively) called **Geo**, and written in C^{++}.

I am adapting this prover to 3-valued logic.

I will not speak about geometric resolution, but only about its most costly operation, which is matching of Horn clauses into interpretations.

So we finally arrived at our main topic!

Matching

We want to know whether a Horn clause, e.g.
$p(X, Y),\ q(Y, Z),\ r(Z, T) \rightarrow \bot$ rejects a certain interpretation, e.g.

$$p(0,0), p(0,1), p(1,1), p(1,2), \quad q(0,0), q(1,0), q(2,1), \quad r(1,0), r(1,1).$$

In order to do this, define:

Definition: Assume a set of atoms Φ, whose arguments are variables, and a set of atoms M, whose arguments are constants.

Does there exist a substitution Θ, s.t. for every $A \in \Phi$, $A\Theta \in M$?

Geo tries to construct a satisfying interpretation of a set of Horn clauses by backtracking. Matching is the key operation.

Unfortunately:

theorem: Matching is very NP-complete.

The proof is easy, by reduction from SAT.

Naive Matching

```
match( size_t i, const std::vector<varatom> & horn,
       const std::vector<groundatom > & interpretation,
       subst& theta )
{
   if( i == horn. size( ))
     solution !
   else
   { for( const groundatom& at : interpretation )
      { if( matching horn[i] into at is
              consistent with theta )
         {  match( i+1, horn, interpretation,
                   theta + { A[i], at } );
         }
} } }
```

```
Adding lemma: 3035 : c2(V1) /\ f1(V1,V1,V2) /\ f1(V2,V1,V3)
/\ f1(V4,V1,V2) /\ f1(V5,V1,V2) /\ f1(V6,V1,V2) /\ f1(V7,V1,V3)
/\ f1(V8,V1,V2) /\ f1(V9,V1,V2) /\ f1(V10,V1,V3) /\ f1(V11,V10,V2
/\ f1(V12,V1,V13) /\ f1(V14,V1,V13) /\ f1(V15,V12,V16) /\
f1(V16,V4,V15) /\ f1(V17,V14,V16) /\ f1(V18,V1,V19) /\
f1(V16,V18,V20) /\ f4(V20,V21) /\ f1(V17,V21,V22) /\
f1(V23,V7,V19) /\ f1(V24,V1,V25) /\ f1(V1,V25,V26) /\
f1(V27,V24,V28) /\ f1(V27,V26,V29) /\ f1(V23,V30,V29) /\
f1(V31,V1,V28) /\ f4(V32,V30) /\ f1(V34,V5,V33) /\
f1(V34,V2,V35) /\ f1(V33,V0,V22) /\ f4(V36,V0) /\
f1(V37,V1,V38) /\ f1(V1,V38,V39) /\ f1(V40,V39,V35) /\
f1(V40,V37,V41) /\ f4(V41,V42) /\ f1(V36,V42,V43) /\
f1(V44,V6,V43) /\ f4(V44,V45) /\ f4(V46,V45) /\ f4(V47,V45) /\
f1(V47,V46,V48) /\ f1(V49,V48,V22) /\ f1(V49,V31,V50) /\
f1(V50,V51,V28) /\ f1(V52,V1,V53) /\ f1(V1,V53,V54) /\
f1(V55,V54,V1) /\ f1(V55,V52,V32) /\ f1(V56,V1,V57) /\
```

```
f1(V58,V1,V57) /\ f1(V59,V58,V22) /\ f1(V60,V56,V61) /\
f1(V62,V8,V60) /\ f1(V61,V63,V51) /\ f1(V62,V63,V64) /\
f1(V65,V1,V66) /\ f1(V1,V66,V67) /\ f1(V68,V67,V64) /\
f1(V68,V65,V69) /\ f4(V69,V70) /\ f1(V59,V70,V71) /\
f1(V72,V9,V71) /\ f4(V72,V73) /\ f1(V74,V1,V75) /\
f1(V1,V75,V76) /\ f1(V77,V76,V1) /\ f1(V77,V74,V78) /\
f4(V78,V79) /\ f1(V11,V79,V80) /\ f1(V80,V2,V81) /\
V81 != V73 -> FALSE
```

```
   matchings_attempted = 113200000004
   matchings_succeeded = 11099545936
```

Definition: A substclause is an object of form

$$(V_1, \ldots, V_n) := (e_{1,1}, \ldots, e_{1,n}) \mid \cdots \mid (e_{m,1}, \ldots, e_{m,n}).$$

A model of a set of substclauses is a substitution Θ that is consistent with all substclauses.

$$\begin{cases} (X, Y) & := & (0,0) \mid (0,1) \mid (1,1) \mid (1,2) \\ (Y, Z) & := & (0,0) \mid (1,0) \mid (2,1) \\ (Z, T) & := & (1,0) \mid (1,1) \end{cases}$$

This corresponds to matching $p(X,Y),\ q(Y,Z),\ r(Z,T)$ into

$$p(0,0), p(0,1), p(1,1), p(1,2), \quad q(0,0), q(1,0), q(2,1), \quad r(1,0), r(1,1).$$

Algorithm (Simple)

1. Start with $\Theta = \emptyset$.
2. Pick an unchecked substclause

$$\overline{V} := \overline{E}_1 \mid \cdots \mid \overline{E}_m.$$

Split $\overline{E}_1, \ldots, \overline{E}_m$ into those that are in conflict with Θ, and those that are not:

$$\overline{F}_1, \ldots, \overline{F}_p, \qquad \overline{G}_1, \ldots, \overline{G}_q.$$

If $q = 0$, then fail.

Otherwise, backtrack through all $\overline{G}_j$: For each $\overline{G}_j$, add the missing assignments to Θ and continue search.

Learning from Conflicts (Borrowed from DPLL)

Definition A lemma is an object of form $\Sigma \to \bot$.

A lemma $\Sigma \to \bot$ is correct if there exists no substitution Θ, s.t. $\Sigma \subseteq \Theta$ and Θ is a solution of the matching problem.

The search algorithm maintains a set of lemmas. Whenever it fails to obtain a solution, it creates a lemma that is applicable on the present substitution.

Lemmas are usually much smaller than the current substitution.

Efficiency results from the fact that lemmas are often reused.

Checking for reusability can be efficiently implemented by watched assignments.

Learning from Conflicts

Assume that we have backtracked through all $\overline{G}_1, \ldots, \overline{G}_q$. Assume that for checking of $\overline{G}_j$, we used extended substitution Θ_j. By recursion, we have a lemma $\Sigma_j \to \bot$, that is applicable on Θ_j. If one of the $\Sigma_j \to \bot$ is applicable on Θ, than return without doing anything.

Otherwise, find minimal subset of Θ that conflicts with all of $\overline{F}_1, \ldots, \overline{F}_p$. Call this set Θ'. (It is not unique.)

Create lemma $\Theta' \cup (\Sigma'_1 \cap \Theta) \cup \cdots \cup (\Sigma'_q \cap \Theta) \to \bot$.

This algorithm is currently used in Geo. Due this algorithm, Geo is a moderately strong theorem prover, instead of totally useless.

Example

$$\begin{cases} (X,Y) & := & (0,0) \mid (0,1) \mid (1,1) \mid (1,2) \\ (Y,Z) & := & (0,0) \mid (1,0) \mid (2,1) \\ (Z,T) & := & (1,0) \mid (1,1) \end{cases}$$

Assume that $\Theta = \{X := 0,\ Y := 0\}$. Consider the second substclause. Assignments $(Y,Z) := (1,0) \mid (2,1)$ are in conflict with Θ. Assignment $(Y,Z) := (0,0)$ is possible.

This results in $\Theta^* = \{X := 0,\ Y := 0,\ Z := 0\}$. Consider the last substclause. Its both assignments are in conflict with Θ^*. The assignment $Z := 0$ is a minimal subset of Θ^* that conflics both of them. Hence we learn $\{Z := 0\} \to \bot$.

Back to previous level: $Y := 0$ is minimal subset of Θ that is in conflict with $(Y,Z) := (1,0) \mid (2,1)$. We learn $\{Y := 0\} \to \bot$.

Filtering

Consider problem again:

$$\begin{cases} (X,Y) & := & (0,0) \mid (0,1) \mid (1,1) \mid (1,2) \\ (Y,Z) & := & (0,0) \mid (1,0) \mid (2,1) \\ (Z,T) & := & (1,0) \mid (1,1) \end{cases}$$

By staring long at the last two substclauses, one sees that only $(Y,Z) := (2,1)$ is possible.

After that, by looking at the first two substclauses, one sees that only $(X,Y) := (1,2)$ is possible.

Problems solved without backtracking!

Filtering (2)

Pick two substclauses $\overline{X} := \overline{E}_1 \mid \cdots \mid \overline{E}_m$ and $\overline{Y} := \overline{F}_1 \mid \cdots \mid \overline{F}_n$ that share a variable. If one of $\overline{X} := \overline{E}_i$ is in conflict with all $\overline{Y} := \overline{F}_1, \ldots, \overline{F}_n$, then remove $\overline{E}_i$ from the clause.

Keep on doing this, until nothing changes.

This filter rejects $90 - 99\%$ of instances a priori, (and is polynomial).

Same could be done with triples, quadruples, etc. I didn't try it (yet). It may be effective.

Unfortunately, the limit is the naive algorithm.

Squeezing

I didn't tell the whole story yet.

We do want to find some matching, we want the best matching.

The learning algorithm cannot efficiently find the best matching.

In order to find the best method, use a technique called sqeezing.

Squeezing (2)

Assume that every atom B in the interpretation has a cost $\#B$.

Assume that Φ can be matched into M through substitution Θ.

Write $M = M_1 \cup M_2$, where initially $M_2 = \emptyset$.

Let B be the most expensive atom in M_1, for which there is an $A \in \Phi$, s.t. $A\Theta = B$.

Let $M_1' = M_1 \backslash \{C \mid \#C \geq \#B\}$.

Call the matching algorithm to match Φ into $M_1' \cup M_2$.

If a matching Θ' is found, restart with (M_1', M_2) and Θ'.

If no matching is found, restart with $(M_1',\ (M_2 \cup \{B\}\ )$ and Θ.

Continue until M_1 is empty.

Conclusion and Future Work

Develop methods that mix filtering and backtracking. (At this moment, filtering is only a preprocessing technique. If it is so succesful, why not make it part of the algorithm after every backtracking choice?)

There are many ways to do this. Resolution? Resolution with Splitting? Splitting with Learning? Informatyka is an empirical science, more like fizyka than matematyka.

Find filtering techniques that work on groups of lemmas.

Find many-one matching methods?