Positive Constructed Formulas Preprocessing for Automatic Deduction

Artem Davydov, **Alexander Larionov**, Evgeny Cherkashin, Savely Arlyapov

`ethrik@gmail.com`, `bootfrost@zoho.com` `eugeneai@icc.ru`

Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences;

Irkutsk National Research Technical University,
Irkutsk, Russian Federation

ISDCT SB RAS, INRTU

31 May 2016
Opatija, Croatia

Introduction: Automated Theorem Proving

Automated theorem proving (ATP)

ATP is a part of artificial intelligence; it is based on methods of mathematical logic and realized as computer programs called provers (or solvers, or systems of ATP).

Theorem

«Theorem» describes a domain and a problem to be solved on some logical language (predicate language, clause language etc.).

Prover

A prover finds out whether some formula is a theorem.

Introduction: Application of provers

1. Solving of mathematical problems. There are examples of solving some open mathematical problems[1];
2. Software and hardware verification;
3. Program synthesis;
4. Expert systems;
5. Problem solving;
6. There are examples of the provers in areas of natural language processing, computer vision etc.

The most famous and efficient provers: Vampire, E, Prover9, Otter, SPASS, EQP, Isabelle, ACL2 etc.

[1] W.McCune. Solution of the Robbins problem;

Features of PCF

The PCF calculus is both **machine-oriented**, and **human-oriented**, naturally it was aimed at solving the problems of **control of dynamic systems**.
Features of PCFs follows:

1. unique inference rule and simple scheme of axioms;
2. modifyability of semantics (constructive, monotonic, temporal, etc.);
3. it is possible to construct intuitionistic inferences of some non-Horn formulas;
4. explicit usage of $\forall-$ and $\exists-$quantifiers;
5. scolemization procedure is not required.

The calculi of PCFs preserve heuristic structure of the original PC[2] presentation of the theorem to be proved.

[2] Predicate calculus.

Positively Constructed Formulae: an example

PC-Formulae:

root: $\forall: True$

base: $\exists v: A(v), B(e)$

questions:

$$\forall: True - \exists v: \underbrace{A(v), B(e)}_{\text{base}} - \begin{cases} \forall x,y: C(x), Q(y) - \exists: False \\ \forall x: P(v,x) - \exists: C(f(x)) \\ \forall x,y: A(x), B(y) - \exists z: P(x,z), Q(y) \end{cases}$$

Prolog-style:

```
a(v).
b(e).
```

```
p(X,g(X,Y)) :- a(X), b(Y).      q(Y) :- a(X), b(Y).
c(f(X)) :- p(v,X).
:- c(X), q(Y).
```

Base contains only ground terms (facts).

Logical inference is saturated base with facts as long as the base will not be contradiction.

Positively Constructed Formulae: Semantics

$$
\forall : T \left\{
\begin{array}{l}
\exists v : C(v), A(k) \left\{
\begin{array}{l}
\forall x : P(k,x) \{ \exists : C(x) \{ \forall z : A(z), P(x,z) \{ \exists u : D(z), Q(u) \\
\forall x : C(x) \left\{
\begin{array}{l}
\exists : D(x), P(x,k) \\
\exists z : P(x,z)
\end{array}
\right. \\
\dots \\
\forall x, y : D(x), A(y) \{ \exists : False
\end{array}
\right. \\
\exists u, v : D(k), C(e), P(h(u,v)) \{ \dots \\
\dots
\end{array}
\right.
$$

Deep subformula

Goal question

Disjunctive branching
(non-Horn)

Conjunctive branching

Research in the field

The research deals with adopting various nowadays techniques such as:

- data structures for formulae representation;
- sharing of data structures, garbage collection;
- efficient structure manipulation;
- term indexing (relations: inst/2, gen/2, unif/2, var/2);
- inference search process control and guiding;
- parallel implementations of the inference rule;
- testing on TPTP library of first–order theorems.

Application areas are (a little progress):

- Logical driven imitation modeling;
- Control synthesis;
- PCFs based programming language.

Formal part: PC to PCF conversion

source predicate calculus formula

$$F = \forall x \forall y (S(x, y) \leftrightarrow \forall z (I(z, x) \rightarrow I(z, y))).$$

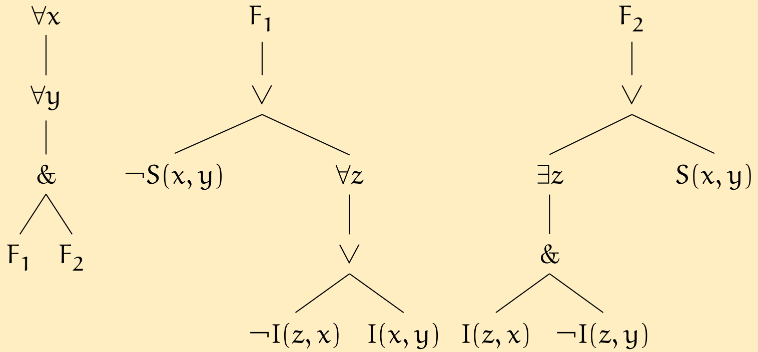
The transformations should preserve a heuristic structure of the original formula.

Connection $\leftrightarrow$ and $\rightarrow$ elimination

$$\begin{aligned} F &= \forall x \forall y (F_1 \& F_2); \\ F_1 &= \neg S(x, y) \vee \forall z (\neg I(z, x) \vee I(z, y)); \\ F_2 &= \neg \forall z (\neg I(z, x) \vee I(z, y)) \vee S(x, y) = \\ F_2' &= \exists z (I(z, x) \& \neg I(z, y)) \vee S(x, y). \end{aligned}$$

PC to PCF conversion

Translated formula



Conversion algorithm (FOL $\rightarrow$ PCF) $F^{\pi}(N)^{P}$

Input: Node N is the root of a tree T for F;
$P \in \{\forall, \exists\}$, $P = \forall$ by default.
Output: F^{π} is a PCF image of FOL F.

if $N = Qx \& P \neq Q$ **then** {Node N has children.}
  return $F^{\pi} = Qx : \emptyset((G^{\pi}_{N'})^{P})$
end if
if $N = \vee$ **then** {Node N has children.}
  return $F^{\pi} = \forall\emptyset((G^{\pi}_{N'_1})^{\exists}, \ldots, (G^{\pi}_{N'_k})^{\exists})$
end if
if $N = \&$ **then** {Node N has children.}
  return $F^{\pi} = \exists\emptyset((G^{\pi}_{N'_1})^{\forall}, \ldots, (G^{\pi}_{N'_k})^{\forall})$
end if
if $N = R$ **then** {R is an atom}
  return $F^{\pi} = \exists\emptyset : R$
end if
if $N = \neg R$ **then** {R is an atom}
  return $F^{\pi} = \forall\emptyset : R$
end if

Result of conversion for our example

$$\forall x\colon \emptyset - \exists\emptyset \;\text{——}\; \forall y\colon \emptyset - \exists\emptyset \left\langle \begin{array}{l} \forall\emptyset \left\langle \begin{array}{l} \exists\emptyset \;\text{——}\; \forall\colon S(x,y) \\ \exists\emptyset \;\text{——}\; \forall z\colon \emptyset \left\langle \begin{array}{l} \exists\emptyset \;\text{——}\; \forall\colon I(z,x) \\ \exists\colon I(x,y) \end{array}\right. \end{array}\right. \\ \forall\emptyset \left\langle \begin{array}{l} \exists z\colon \emptyset \left\langle \begin{array}{l} \forall\emptyset \;\text{——}\; \exists\colon I(z,x) \\ \forall\colon I(z,y) \end{array}\right. \\ \exists\colon S(x,y) \end{array}\right. \end{array}\right.$$

Now, the formula is to be reduced.

Theorem: Reduction rule

Input

$F_1 = Q_X : A$
- $P_Y : B$
  - $Q : C$
    - $P_{Z_1} : C_1$ — Φ_1
    - ...
    - $P_{Z_k} : C_k$ — Φ_k
  - Ψ
- Φ

Quantifiers $P, Q \in \{\forall, \exists\}$, $P \neq Q$, A, C, B are conjunctions, $C \subseteq B$, Φ, Ψ, Φ_i are sets of formulae. After conversion $F_2 \leftrightarrow F_1$.

Result

$F_2 = Q_X : A$
- $P_{Y,Z_1} : B, C_1$ — $\Phi_1 \cup \Psi$
- ...
- $P_{Y,Z_k} : B, C_k$ — $\Phi_k \cup \Psi$
- Φ

Reduction rule applied to the example

Reduced PCF

- $\forall x, y\colon \emptyset$
  - $\exists \emptyset$
    - $\forall z\colon S(x, y), I(z, x)$
      - $\exists\colon I(x, y)$
    - $\forall \emptyset$
      - $\exists z\colon I(z, x)$
        - $\forall\colon I(x, y)$
      - $\exists\colon S(x, y)$

There are two reduction options: eliminate $\exists\emptyset$ or $\forall\emptyset$, the latter applied.

The final result

- $\exists \emptyset$
  - $\forall x, y, z\colon S(x, y), I(z, x)$
    - $\exists\colon I(x, y)$
  - $\forall x, y\colon \emptyset$
    - $\exists z\colon I(z, x)$
      - $\forall\colon I(x, y)$
    - $\exists\colon S(x, y)$

Further development of the conversion engine

The algorithm and the software is being developed further.

- Adaptation to TPTP syntax;
- Conjunctive Normal Form (CNF) support;
- Reconstruct the heuristic structure of CNF;
- Implement it in Rust programming language as it
  - does not include a garbage collector in compiled code;
  - supports a strong memory control technique;
  - has reasonable compatibility with C and C++.

We already have a version of the algorithm for CNF having no existential variables.

Results of prover development

TPTP. www.tptp.org We developed a D-language version of prover, where implemented

1. memory data sharing;
2. two indexing techniques;
3. basic support of simple constraints and strategies;
4. connection to outer world via Python classes;
5. tested on all FOL theorems from TPTP library.

TPTP test results

Total number of solved problems

Complexity	Total number	Solved
0,0-0,03	192	181
0,04-0,20	435	373
0,21-0,32	128	79
0,33-0,49	115	44
0,5-0,67	223	21
0,68-0,92	72	6
0,93-1,0	56	0

Our rating is about 0,1–0,15.

Testing results

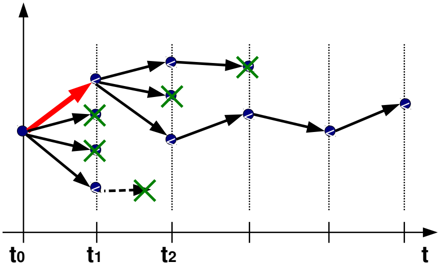
Results by domains

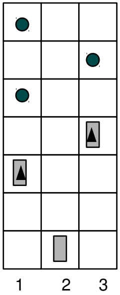
Domain	Total number	Solved
Geometry (GEO)	242	204
Management (MGT)	22	22
Syntax (SYN)	275	180
Semantic web (SWB)	25	22

Most complex solved (not checked)

Problem	Complexity	Time,s	Step count
LCL652+1.015	0,92	32,1	185 253
LCL656+1.020	0,92	1,24	845

Research target: Logical-dynamic imitation modeling





Time modeling: A(t_0) is state at the moment t_0 and t' is the next moment after t

$$\exists: A(t_0)\begin{cases}\forall t: T(t)\{\exists t': T(t'), N(t,t')\\ \Phi \\ \Psi\end{cases}$$

$$F \to \cancel{G}$$

The goal of control cannot be defined

1. Intuitionistic inference in non–Horn formalization.
2. Utilization of prover engine in forward-chaining inference.
3. Constraint satisfaction.

Conclusion

The software development of method of the PCF has a number of challenges, which are under research, but the progress we made shows that the inference method is meaningful to research further.

- Something different to Resolutional methods and Tabbling;
- It has direct technical origins (Theory of Control);
- Constructive inference of non–Horn clauses;
- Heuristic structure presentation.

Positive Constructed Formulas Preprocessing for Automatic Deduction

Artem Davydov, **Alexander Larionov**, Evgeny Cherkashin, Savely Arlyapov

`ethrik@gmail.com`, `bootfrost@zoho.com` `eugeneai@icc.ru`

Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences;

Irkutsk National Research Technical University,
Irkutsk, Russian Federation

ISDCT SB RAS, INRTU

31 May 2016
Opatija, Croatia