Nonclausal Proof Search for Dummies

Michael Färber

VINO 2017

Introduction

History

- 2015: Jens Otten mentions nonclausal proof search at PiWo'15
- 2016:
  - First competitive nonclausal ATP nanoCoP implemented
  - Presented at IJCAR'16
  - Q&A with Jens during PiWo'16
- 2017:
  - Reimplementation of nanoCoP in OCaml
  - Reconstruction of nonclausal proofs in HOL Light

Motivation

Problem

nanoCoP source code: “nicht ganz trivial” (Jens)

Goals

- Reimplement it in functional (OCaml) instead of relational language (Prolog) to make it faster
- Integrate it into interactive proof assistant (HOL Light) to certify proofs
- Helps others to understand the implementation (hopefully)

Nonclausal Proof Search

Clausal Proof Search

Clausal Matrix Example

$$M = \left[\begin{array}{cc} C_1 & \left[\begin{array}{c} L_1 \\ L_2 \end{array} \right] \end{array} \right]$$

Matrix Interpretation

The formula M' corresponding to M is:

$$M' = C_1' \wedge (L_1 \vee L_2)$$

The goal is to refute M', i.e. to show $M' \implies \bot$.

Extension

Given we know $\neg L_1$, then we can infer the extension clause $[L_2]$.

Nonclausal Proof Search

Nonclausal Matrix Example

$$M = \left[C_1 \quad \left[\quad \left[C_2 \quad \begin{bmatrix} L \\ M_2 \\ M_3 \end{bmatrix} \right] \right] \right] \quad M_1$$

Nonclausal Proof Search

Nonclausal Matrix Example

$$M = \left[\begin{array}{c} C_1 \\ \left[\begin{array}{c} M_1 \\ \left[\begin{array}{c} C_2 \\ \left[\begin{array}{c} L \\ M_2 \\ M_3 \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

Matrix Interpretation

$M' = C_1' \wedge (M_1' \vee (C_2' \wedge (L \vee M_2' \vee M_3')))$

Extension

Given we know $\neg L$, which extension clauses can we infer?

Nonclausal Proof Search

Nonclausal Matrix Example

$$M = \left[\begin{array}{c} C_1 \\ \left[\begin{array}{c} \\ \left[\begin{array}{cc} & M_1 \\ C_2 & \left[\begin{array}{c} L \\ M_2 \\ M_3 \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

Matrix Interpretation

$M' = C_1' \wedge (M_1' \vee (C_2' \wedge (L \vee M_2' \vee M_3')))$

Extension

Given we know $\neg L$, which extension clauses can we infer?

$$\left[\begin{array}{c} \\ \left[\begin{array}{cc} & M_1 \\ C_2 & \left[\begin{array}{c} M_2 \\ M_3 \end{array}\right] \end{array}\right] \end{array}\right] \text{ and potentially } \left[\begin{array}{c} M_2 \\ M_3 \end{array}\right].$$

Nonclausal Extension Clauses

- nanoCoP can generate extension clauses from a submatrix M if the current proof branch made an extension into M.
- Multiple possible extension clauses may need to be tried for completeness, because submatrices may have substituted variables.
- Smallest clauses are tried first for proof search efficiency.

Improvements

Empty Extension Clauses

$$M = \left[\begin{array}{cc} [\neg p] & \left[\begin{array}{c} \overset{\neg r}{} \\ [p] \quad [q] \end{array} \right] \end{array} \right]$$

The extension clauses for $\neg p$ are:

Empty Extension Clauses

$$M = \left[\begin{array}{cc} [\neg p] & \left[\begin{array}{c} \neg r \\ \left[\begin{array}{cc} [p] & [q] \end{array} \right] \end{array} \right] \end{array} \right]$$

The extension clauses for $\neg p$ are:

- In the original nanoCoP: $\left[\left[\left[\right] \right] \quad \neg r \right]$
- In my improved version: $\left[\neg r \right]$
- The improved version is more natural to reconstruct and more efficient.

Extension Clause Bug

$$M = \left[\; C_1 \quad \left[\; \left[\; C_2 \quad \begin{matrix} M_1 \\ \left[\begin{matrix} L \\ [C_3] \\ M_3 \end{matrix}\right] \end{matrix} \;\right] \;\right] \;\right]$$

What is the smallest possible extension clause for $\neg L$?

Extension Clause Bug

$$M = \left[C_1 \quad \left[\left[C_2 \quad \begin{array}{c} M_1 \\ \left[\begin{array}{c} L \\ [C_3] \\ M_3 \end{array} \right] \end{array} \right] \right] \right]$$

What is the smallest possible extension clause for $\neg L$?

- According to the original nanoCoP: C_3.
- If $[C_3]$ swapped with M_3, C_3 is not admissible any more as extension clause.
- Not yet proven to be unsound, but very fishy.
- Discovered by more reconstruction-friendly reimplementation.
- Fixed in my nanoCoP version; Jens is notified.

Extension Clause Order

$$M = \left[\, C_1 \quad \left[\, \left[\, C_2 \quad \left[\begin{array}{c} L \\ M_2 \\ M_3 \end{array}\right] \,\right] \quad M_1 \,\right] \,\right]$$

- Original extension clause for $\neg L$:

$$\left[\, \left[\, \left[\begin{array}{c} M_2 \\ M_3 \end{array}\right] \quad C_2 \,\right] \quad M_1 \,\right]$$

- Improved:

$$\left[\, \left[\, C_2 \quad \left[\begin{array}{c} M_2 \\ M_3 \end{array}\right] \,\right] \quad M_1 \,\right]$$

- Improved version is more natural for reconstruction
- How does it perform?

Implementation

Bug fixing

Methodology

- Run OCaml-nanoCoP on all TPTP problems
- Find smallest solved problem where number of inferences differs
- Produce proof search trace for OCaml and Prolog nanoCoP
- Compare traces and think very hard ...
- Fix bug
- Repeat

Experiment Management

- Keeping track of program versions & experiments
- "Which version of the prover with which options did I run to achieve the results in this folder?"
- Solution: `git tag` + `make`

Registering a new version

Makefile

```
cop-%: ../.git/refs/tags/cop-%
    git clone .. $@ && cd $@ && git checkout $@

%/nanocop.native: %
    make -C $< nanocop.native
```

Workflow

```
git tag cop-170829
```

Running an experiment

Makefile

```
BUSHY = $(shell find bushy/ -type f | sort -R)

o/b10s/nc170829: $(BUSHY:bushy/%=\
o/b10s/nc170829/%)
o/b10s/nc170829/%: cop-170829/nanocop.native bushy/%
    @mkdir -p "`dirname $@`"
    -/usr/bin/time -o "$@.time" timeout 10 $^ > "$@"
```

Workflow

- Write new rule in Makefile
- `make o/b10s/nc170829 -j$(CORES)`
- `make s/b10s/nc170829` to obtain set of solved problems

Advantages

- Experiment log
- Reproducible results
- Small dependencies (only GNU `make`)
- Parallelisation and resuming for free

Evaluation

Extension Clause Order

Table 1: nanoCoP with original and improved extension clause order. OCaml version, 10s timeout, single strategy (cut).

Dataset	Before	After	Change
MPTP2078 (Bushy)	491	529	+7.7%
TPTP 3.7.0 (FOF)	1235	1281	+3.7%

The Paramount Role of the Path

Table 2: leanCoP/nanoCoP with and without matrix reordering by paths. OCaml version, 10s timeout, single strategy. leanCoP run with Tseitin transformation.

Configuration	Without Paths	With Paths	Change
nanoCoP + cut	299	491	+64.2%
nanoCoP - cut	345	505	+42.6%
leanCoP + cut	548	612	+11.7%
leanCoP - cut	340	393	+15.6%

Observations

- Reordering by paths has much greater effect in nanoCoP
- Cut in nanoCoP has the opposite effect than in leanCoP

Conclusion

Conclusion

Material covered

- Introduction to nonclausal proof search
- Several improvements:
  - smaller extension clauses
  - fixed extension clause generation
  - improved extension clause order
- Experiment management
- Impact of matrix ordering by path and of extension clause order

Take-home message

Proof reconstruction can give valuable insights for proof search