Project Proposal: Efficient Neural Clause Selection[1]

Filip Bártek Martin Suda

Czech Technical University in Prague

September 8, 2022

[1] This work is supported by the Czech Science Foundation project no. 20-06390Y (JUNIOR grant), and the Grant Agency of the Czech Technical University in Prague, grant no. SGS20/215/OHK3/3T/37.

Context

Saturation-based automated theorem proving for first-order logic

Context

Saturation-based automated theorem proving for first-order logic

Context

Saturation-based automated theorem proving for first-order logic

Saturation-based theorem proving

Two sets of clauses:

- Passive
- Active

Saturation loop:

1. Select clause C from Passive
2. Perform all inferences between C and all clauses in Active
    - Add the newly inferred clauses to Passive
3. Move C from Passive to Active

Saturation-based theorem proving

Two sets of clauses:

- Passive
- Active

Saturation loop:

1. Select clause C from Passive
2. Perform all inferences between C and all clauses in Active
    - Add the newly inferred clauses to Passive
3. Move C from Passive to Active

Saturation-based theorem proving

Two sets of clauses:

- Passive
- Active

Saturation loop:

1. Select clause C from Passive
2. Perform all inferences between C and all clauses in Active
    - Add the newly inferred clauses to Passive
3. Move C from Passive to Active

Saturation-based theorem proving

Two sets of clauses:

- Passive
- Active

Saturation loop:

1. Select clause C from Passive
2. Perform all inferences between C and all clauses in Active
    - Add the newly inferred clauses to Passive
3. Move C from Passive to Active

Saturation-based theorem proving

Two sets of clauses:

- Passive
- Active

Saturation loop:

1. Select clause C from Passive
2. Perform all inferences between C and all clauses in Active
    - Add the newly inferred clauses to Passive
3. Move C from Passive to Active

Saturation-based theorem proving

Two sets of clauses:

- Passive
- Active

Saturation loop:

1. Select clause C from Passive – **Which one?**
2. Perform all inferences between C and all clauses in Active
    - Add the newly inferred clauses to Passive
3. Move C from Passive to Active

Clause weight

$w($`product(X0,X1,multiply(X0,X1))`$)$

Clause weight

$$w(\texttt{product(X0,X1,multiply(X0,X1))}) = 6$$

Clause weight

$$
\begin{aligned}
&w(\texttt{product(X0,X1,multiply(X0,X1))}) \\
=&4w_X + w_{\text{product}} + w_{\text{multiply}}
\end{aligned}
$$

Clause weight

$$\begin{aligned} & w(\texttt{product(X0,X1,multiply(X0,X1))}) \\ =& 4w_X + w_{\mathrm{product}} + w_{\mathrm{multiply}} \\ =& \begin{bmatrix} 4 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_X & w_{\mathrm{product}} & w_{\mathrm{multiply}} \end{bmatrix} \end{aligned}$$

Input problem

Problem GRP003-2 from TPTP:

```
cnf(left_identity,axiom,
    ( product(identity,X,X) )).

cnf(left_inverse,axiom,
    ( product(inverse(X),X,identity) )).

cnf(total_function1,axiom,
    ( product(X,Y,multiply(X,Y)) )).

cnf(total_function2,axiom,
    ( ~ product(X,Y,Z)
    | ~ product(X,Y,W)
    | equalish(Z,W) )).
```

```
cnf(associativity1,axiom,
    ( ~ product(X,Y,U)
    | ~ product(Y,Z,V)
    | ~ product(U,Z,W)
    | product(X,V,W) )).

cnf(associativity2,axiom,
    ( ~ product(X,Y,U)
    | ~ product(Y,Z,V)
    | ~ product(X,V,W)
    | product(U,Z,W) )).

cnf(product_substitution3,axiom,
    ( ~ equalish(X,Y)
    | ~ product(W,Z,X)
    | product(W,Z,Y) )).

cnf(prove_right_identity,negated_conjecture,
    ( ~ product(a,identity,a) )).
```

- Predicates: `product`, `equalish`
- Functions: `identity`, `inverse`, `multiply`, `a`

Default weights

```
% Instruction limit reached!
% ------------------------------
[...]
% Termination reason: Unknown
% Termination phase: Saturation

% Memory used [KB]: 37611
% Time elapsed: 256.608 s
% Instructions burned: 1000003 (million)
% ------------------------------
----  Runtime statistics ----
clauses created: 185175
clauses deleted: 70528
-----------------------------

% ------------------------------
```

Custom weights

```
% Refutation found. Thanks to Tanya!
% SZS status Unsatisfiable for GRP003-2
% ------------------------------
[...]
% Termination reason: Refutation

% Memory used [KB]: 639
% Time elapsed: 0.035 s
% Instructions burned: 47 (million)
% ------------------------------
----  Runtime statistics ----
clauses created: 44
clauses deleted: 11
------------------------------
% ------------------------------
```

Variables: 0.1
`multiply`: 10
`equalish`: 10

Custom weights

Variables: 0.1
multiply: 10
equalish: 10

```
% Refutation found. Thanks to Tanya!
% SZS status Unsatisfiable for GRP003-2
% -------------------------------
[...]
% Termination reason: Refutation

% Memory used [KB]: 639
% Time elapsed: 0.035 s
% Instructions burned: 47 (million)
% -------------------------------
----  Runtime statistics  ----
clauses created: 44
clauses deleted: 11
------------------------------
% -------------------------------
```

Clauses activated in proof search

```
1. product(identity,X0,X0) [input]
2. product(inverse(X0),X0,identity) [input]
8. ~product(a,identity,a) [input]
5. ~product(X4,X2,X3) | product(X0,X5,X3) | ~product(X1,X2,X5) | ~product(X0,X1,X4) [input]
9. ~product(X3,X2,X1) | product(X0,X1,X2) | ~product(X0,X3,identity) [resolution 5,1]
11. ~product(X0,identity,identity) | product(X0,X1,X1) [resolution 9,1]
14. product(inverse(identity),X1,X1) [resolution 11,2]
6. ~product(X1,X2,X5) | ~product(X0,X5,X3) | product(X4,X2,X3) | ~product(X0,X1,X4) [input]
18. ~product(X0,identity,X3) | product(X3,X1,X2) | ~product(X0,X1,X2) [resolution 6,1]
12. ~product(X2,inverse(X3),identity) | product(X2,identity,X3) [resolution 9,2]
24. product(inverse(inverse(X0)),identity,X0) [resolution 12,2]
25. product(inverse(inverse(identity)),X0,X0) [resolution 24,11]
26. ~product(inverse(inverse(X1)),X2,X3) | product(X1,X2,X3) [resolution 24,18]
37. product(X2,identity,X2) [resolution 26,24]
```

Clauses activated in proof search

```
1. product(identity,X0,X0) [input]
2. product(inverse(X0),X0,identity) [input]
8. ~product(a,identity,a) [input]
5. ~product(X4,X2,X3) | product(X0,X5,X3) | ~product(X1,X2,X5) | ~product(X0,X1,X4) [input]
9. ~product(X3,X2,X1) | product(X0,X1,X2) | ~product(X0,X3,identity) [resolution 5,1]
11. ~product(X0,identity,identity) | product(X0,X1,X1) [resolution 9,1]
14. product(inverse(identity),X1,X1) [resolution 11,2]
6. ~product(X1,X2,X5) | ~product(X0,X5,X3) | product(X4,X2,X3) | ~product(X0,X1,X4) [input]
18. ~product(X0,identity,X3) | product(X3,X1,X2) | ~product(X0,X1,X2) [resolution 6,1]
12. ~product(X2,inverse(X3),identity) | product(X2,identity,X3) [resolution 9,2]
24. product(inverse(inverse(X0)),identity,X0) [resolution 12,2]
25. product(inverse(inverse(identity)),X0,X0) [resolution 24,11]
26. ~product(inverse(inverse(X1)),X2,X3) | product(X1,X2,X3) [resolution 24,18]
37. product(X2,identity,X2) [resolution 26,24]
```

Goal

Overall goal: Improve performance of saturation-based ATPs

Our approach:

- Automatically assign weights to symbols
  - Clause weight = symbol weights · symbol occurrence counts
- Use a GNN to generate the symbol weights
  - GNN only runs in preprocessing

Goal

Overall goal: Improve performance of saturation-based ATPs

Our approach:

- Automatically assign weights to symbols
  - Clause weight = symbol weights · symbol occurrence counts
- Use a GNN to generate the symbol weights
  - GNN only runs in preprocessing

Goal

Overall goal: Improve performance of saturation-based ATPs

Our approach:

- Automatically assign weights to symbols
  - Clause weight $=$ symbol weights $\cdot$ symbol occurrence counts
- Use a GNN to generate the symbol weights
  - GNN only runs in preprocessing

Goal

Overall goal: Improve performance of saturation-based ATPs

Our approach:

- Automatically assign weights to symbols
  - Clause weight $=$ symbol weights $\cdot$ symbol occurrence counts
- Use a GNN to generate the symbol weights
  - GNN only runs in preprocessing

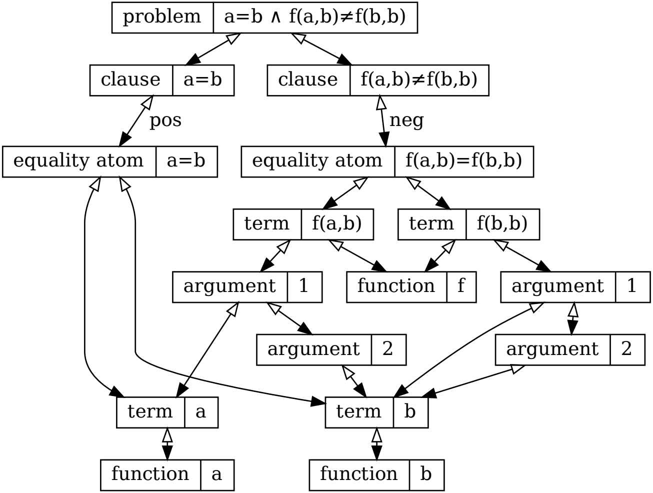
Goal

Overall goal: Improve performance of saturation-based ATPs

Our approach:

- Automatically assign weights to symbols
    - Clause weight = symbol weights · symbol occurrence counts
- Use a GNN to generate the symbol weights
    - GNN only runs in preprocessing

Graph neural network



Training data

```
1. product(identity,X0,X0) [input]
2. product(inverse(X0),X0,identity) [input]
8. ~product(a,identity,a) [input]
5. ~product(X4,X2,X3) | product(X0,X5,X3) | ~product(X1,X2,X5) | ~product(X0,X1,X4) [input
9. ~product(X3,X2,X1) | product(X0,X1,X2) | ~product(X0,X3,identity) [resolution 5,1]
11. ~product(X0,identity,identity) | product(X0,X1,X1) [resolution 9,1]
14. product(inverse(identity),X1,X1) [resolution 11,2]
6. ~product(X1,X2,X5) | ~product(X0,X5,X3) | product(X4,X2,X3) | ~product(X0,X1,X4) [input
18. ~product(X0,identity,X3) | product(X3,X1,X2) | ~product(X0,X1,X2) [resolution 6,1]
12. ~product(X2,inverse(X3),identity) | product(X2,identity,X3) [resolution 9,2]
24. product(inverse(inverse(X0)),identity,X0) [resolution 12,2]
25. product(inverse(inverse(identity)),X0,X0) [resolution 24,11]
26. ~product(inverse(inverse(X1)),X2,X3) | product(X1,X2,X3) [resolution 24,18]
37. product(X2,identity,X2) [resolution 26,24]
```

Symbol weight recommender

- Input: First-order logic (FOL) problem with signature Σ
- Output:
  - Single weight for the variables
  - Weight for each symbol $s \in \Sigma$
  - Each weight is a real number
- Architecture:
  - Graph neural network (GNN)
    - Signature-agnostic
    - Output: Symbol weights
  - Loss on a clause: Binary cross-entropy on clause weight
    - Clause weight $=$ symbol weights $\cdot$ symbol occurrence counts

Results

Model	Problems solved[2]	
	T=0	Best
Baseline	232	232

[2] Training problems solved out of 244

Results

Model	Problems solved[2]			Accuracy	
	T=0	T=1000	Best	T=0	T=1000
Baseline	232		**232**		
GNN	92	132	150	0.51	0.78

[2] Training problems solved out of 244

Results

Model	Problems solved[2]			Accuracy	
	T=0	T=1000	Best	T=0	T=1000
Baseline	232		**232**		
GNN	92	132	150	0.51	0.78
GNN+	204	196	204	0.50	0.75

[2] Training problems solved out of 244

Next steps

- Improve loss: Learn from pairs of clauses
- Additional features for clause weight:
  - Occurrence counts: variables, equalities, inequalities, positive and negative literals
  - Number of bound variables
  - Runtime: derivation depth and size, age
- Training/evaluation loop
  - Extract negative samples from failed proof attempts

Next steps

- Improve loss: Learn from pairs of clauses
- Additional features for clause weight:
  - Occurrence counts: variables, equalities, inequalities, positive and negative literals
  - Number of bound variables
  - Runtime: derivation depth and size, age
- Training/evaluation loop
  - Extract negative samples from failed proof attempts

Thank you!

Configuration

Configuration:

- Automated theorem prover (ATP): Vampire
  - Saturation algorithm: DISCOUNT
  - AVATAR: off
  - Age-weight ratio: 1:1
  - Instruction limit: 5×10^{10} instructions
- Dataset: 500 training FOL problems from TPTP 7.5.0

Simplifications:

- Variable weight is common for all input problems
- Equality weight is hard-coded to 1

Explosive proof search on GRP011-4

```
[SA] active: 5. b != d [input] {a:0,w:3,wCS:-0.963572,nSel:1,goal:1,thAx:0,allAx
     :1,thDist:-1}
[SA] active: 1. multiply(multiply(X0,X1),X2) = multiply(X0,multiply(X1,X2)) [
     input] {a:0,w:11,wCS:0.28454,nSel:1,thAx:0,allAx:1,thDist:-1}
[...]
[SA] active: 767. multiply(X99,multiply(X100,multiply(X101,multiply(X102,
     multiply(X103,multiply(X89,multiply(X90,multiply(X91,multiply(X92,X98))))))
     ))) = multiply(inverse(multiply(X104,multiply(X105,multiply(X106,multiply(
     inverse(multiply(X89,multiply(X90,multiply(X91,multiply(X92,inverse(
     multiply(X93,multiply(X94,multiply(X95,multiply(X96,X97)))))))))),inverse(
     multiply(X99,multiply(X100,multiply(X101,multiply(X102,X103)))))))))),
     multiply(X104,multiply(X105,multiply(X106,multiply(X93,multiply(X94,
     multiply(X95,multiply(X96,multiply(X97,X98))))))))) [superposition 409,409]
      {a:7,w:75,wCS:-1.60515,nSel:1,thAx:0,allAx:24,thDist:-24}
[SA] active: 18. c = multiply(inverse(d),multiply(b,c)) [superposition 11,4] {a
     :2,w:8,wCS:2.862,nSel:1,thAx:0,allAx:3,thDist:-3}
[SA] active: 1446. multiply(X197,multiply(X198,multiply(X199,multiply(X200,
     multiply(X201,multiply(X202,multiply(X203,multiply(X204,multiply(X205,X206)
     )))))))) = multiply(inverse(multiply(X207,multiply(X208,multiply(X209,
     multiply(inverse(multiply(X202,multiply(X203,multiply(X204,multiply(X205,
     inverse(multiply(X210,multiply(X211,multiply(X212,multiply(X191,multiply(
     X192,multiply(X193,multiply(X194,multiply(X195,multiply(X182,multiply(X183,
     multiply(X184,multiply(X185,X196)))))))))))))))))),inverse(multiply(X197,
     multiply(X198,multiply(X199,multiply(X200,X201)))))))))),multiply(X207,
     multiply(X208,multiply(X209,multiply(X210,multiply(X211,multiply(X212,
     multiply(X191,multiply(X192,multiply(X193,multiply(X194,multiply(X195,
     multiply(X182,multiply(X183,multiply(X184,multiply(X185,multiply(X196,X206)
     )))))))))))))))) [forward demodulation 1445,767] {a:8,w:107,wCS:-4.56565,
     nSel:1,thAx:0,allAx:48,thDist:-48}
[SA] active: 2342. multiply(X319,multiply(X320,multiply(X321,multiply(X322,
     multiply(X323,multiply(X324,multiply(X325,multiply(X326,multiply(X327,
     multiply(X293,multiply(X294,multiply(X295,multiply(X300,multiply(X301,
     multiply(X302,multiply(X303,multiply(X304,multiply(X305,multiply(X306,
```

Tools

- TensorFlow
- Deep Graph Library
- Vampire