Reverse Polish Notation
SFT Simulation R&D meeting

Jonas Hahnfeld

November 29, 2022
Consider arithmetic expressions, e.g.

\[ 6 + (5 - 4) \times 3 \]
Consider arithmetic expressions, e.g.

$$6 + (5 - 4) \times 3$$

Operator precedence applies:
1. Parentheses
2. Multiplication
3. Addition
Consider arithmetic expressions, e.g.

\[ 6 + (5 - 4) \times 3 \]

Operator precedence applies:
1. Parentheses
2. Multiplication
3. Addition

One strategy: parse into tree
Consider arithmetic expressions, e.g.

\[ 6 + (5 - 4) \times 3 \]

Operator precedence applies:
1. Parentheses
2. Multiplication
3. Addition

One strategy: parse into tree
→ evaluate recursively
Consider arithmetic expressions, e.g.

$$6 + (5 - 4) \times 3$$

Operator precedence applies:
1. Parentheses
2. Multiplication
3. Addition

One strategy: parse into tree
\[ \rightarrow \text{evaluate recursively} \]
Consider arithmetic expressions, e.g.

\[ 6 + (5 - 4) \times 3 \]

Operator precedence applies:

1. Parentheses
2. Multiplication
3. Addition

One strategy: parse into tree
→ evaluate recursively
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 \ 5 \ 4 \ - \ 3 \ \times \ + \]
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 5 4 - 3 \times + \]

Evaluate expression using a stack
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 \ 5 \ 4 \ - \ 3 \ \times \ + \]

Evaluate expression using a stack
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 \ 5 \ 4 \ - \ 3 \ \times \ + \]

Evaluate expression using a stack
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 \ 5 \ 4 - 3 \times + \]

Evaluate expression using a stack

\[
\begin{array}{c}
6 \\
5 \\
4 \\
\end{array}
\]
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

$6 + (5 - 4) \times 3$

postfix notation:

$6 5 4 - 3 \times +$

Evaluate expression using a stack

```
6
→
1
```
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 \ 5 \ 4 \ - \ 3 \ \times \ + \]

Evaluate expression using a stack:

\[ \begin{array}{c}
\text{6} \\
\text{1} \\
\text{3}
\end{array} \]

\[ \rightarrow \]

Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

$$6 + (5 - 4) \times 3$$

postfix notation:

$$6 \ 5 \ 4 \ - \ 3 \ \times \ \ +$$

Evaluate expression using a stack
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 \ 5 \ 4 - 3 \times + \]

Evaluate expression using a stack

\[ \rightarrow \ 9 \]

\[ \]

\[ \]

\[ \]
Reverse Polish Notation
Stack-Based Evaluation of Arithmetic Expressions

infix notation:

\[ 6 + (5 - 4) \times 3 \]

postfix notation:

\[ 6 \ 5 \ 4 \ - \ 3 \ \times \ + \]

Evaluate expression using a stack

\[ \rightarrow 9 \]

\[ \ \ \ \ \ \ \ \ ]
Reverse Polish Notation
Analogy for Boolean CSG Trees

Concept caries over to boolean CSG trees

- Values: volumes / half-spaces of surfaces (or rather if a point is Inside)
- Operations: union, intersection, subtraction (or negation)
Reverse Polish Notation

Analogy for Boolean CSG Trees

Concept carries over to boolean CSG trees
  ▶ Values: volumes / half-spaces of surfaces (or rather if a point is \textit{Inside})
  ▶ Operations: union, intersection, subtraction (or negation)

Some optimizations not (easily) possible, such as short-circuiting
  ▶ When pushing an item, did not see yet the operator
Reverse Polish Notation

Analogy for Boolean CSG Trees

Concept carries over to boolean CSG trees

- Values: volumes / half-spaces of surfaces (or rather if a point is Inside)
- Operations: union, intersection, subtraction (or negation)

Some optimizations not (easily) possible, such as short-circuiting

- When pushing an item, did not see yet the operator

Could push only IDs and evaluate when required for an operator

- Works for some cases, but not for nested operators
- May instead be able to add “annotations” to skip part of the expression
Reverse Polish Notation makes it easy to evaluate Inside.
Further Considerations for Boolean CSG Trees

- Reverse Polish Notation makes it easy to evaluate Inside

- Distance and Safety are harder
  - May need to propagate through connected constituents of unions
Further Considerations for Boolean CSG Trees

▶ Reverse Polish Notation makes it easy to evaluate Inside

▶ Distance and Safety are harder
  ▶ May need to propagate through connected constituents of unions

▶ Approach based on solids:
  ▶ Model solids with bounded surfaces (cf VolumeShell)
  ▶ Apply boolean operations between solids, business as usual
Reverse Polish Notation makes it easy to evaluate Inside.

Distance and Safety are harder.
- May need to propagate through connected constituents of unions.

Approach based on solids:
- Model solids with bounded surfaces (cf VolumeShell).
- Apply boolean operations between solids, business as usual.

Approach based on surfaces:
- Generate all intersections with all surfaces, sort by distance.
- Check one by one if the intersection leaves the volume (using RPN).
- (implemented by Celeritas / ORANGE for complex = non-convex volumes).
Further Considerations for Boolean CSG Trees

Different approach for DistanceToIn and SafetyToIn

- Any Boolean expression has an equivalent Disjunctive Normal Form (DNF):
  - A (multi-)union of intersections: $\bigcup(\bigcap H_i)$
  - Maybe even possible to ensure convexity of $\bigcap H_i$?
  - Caveat: may be large, up to exponential in the number of variables / half-spaces

Possible approach for DistanceToIn and SafetyToIn:
- Entering the union if entering any of the constituents
- Check all intersections, return minimum distance / safety

No good idea for DistanceToOut and SafetyToOut

Similar concept of Conjunctive Normal Form (CNF):
- Exiting the intersection if exiting any of the constituents
- But what is a union of half-spaces?
Further Considerations for Boolean CSG Trees

Different approach for DistanceToIn and SafetyToIn

▷ Any Boolean expression has an equivalent Disjunctive Normal Form (DNF):
  ▷ A (multi-)union of intersections: $\bigcup(\bigcap H_i)$
  ▷ Maybe even possible to ensure convexity of $\bigcap H_i$?
  ▷ Caveat: may be large, up to exponential in the number of variables / half-spaces

▷ Possible approach for DistanceToIn and SafetyToIn:
  ▷ Entering the union if entering any of the constituents
  ▷ Check all intersections, return minimum distance / safety
Further Considerations for Boolean CSG Trees

Different approach for *DistanceToIn* and *SafetyToIn*

- Any Boolean expression has an equivalent Disjunctive Normal Form (DNF):
  - A (multi-)union of intersections: $\bigcup(\bigcap H_i)$
  - Maybe even possible to ensure convexity of $\bigcap H_i$?
  - Caveat: may be large, up to exponential in the number of variables / half-spaces

- Possible approach for *DistanceToIn* and *SafetyToIn*:
  - Entering the union if entering any of the constituents
  - Check all intersections, return minimum distance / safety

- No good idea for *DistanceToOut* and *SafetyToOut*
  - Similar concept of Conjunctive Normal Form (CNF): $\bigcap(\bigcup H_i)$
  - Exiting the intersection if exiting any of the constituents
  - But what is a union of half-spaces?