proof-tree-renderer

Render proof tree in bussproofs into html

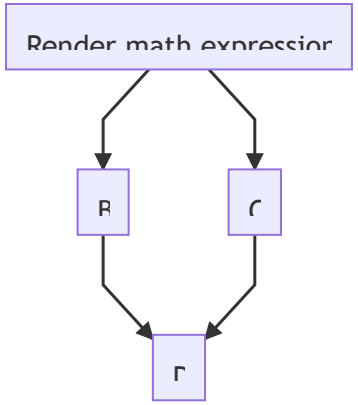
2024.07.06

sano

概要

KaTeX を使えば，HTML 上で数式のレンダリングができるが，bussproofs/LaTeX などを用いた証明木のレンダリングはできない．

本スクリプトは HTML 上の bussproofs を用いた証明木をレンダリングする．



要件

右ラベル前提とする.

概要

ラベルの配置とその分のスペースについては，CSS だけではなくて，JavaScript での動的な処理も必要.

処理の流れ

1. KaTeX を適用して，数式部分をレンダリングする.
2. p, div, li などの DOM 要素を取得する.
3. `\begin{prooftree}...\end{prooftree}` を切り出して DOM 要素のリストと開始位置の組を作る.
4. 証明木パートの LaTeX コードを解析して，LaTeX コマンドのリストを作る.
5. LaTeX コマンドのリストを構文解析して，証明木オブジェクトを生成する.
6. DocumentFragment 上で，証明木オブジェクトから証明木の DOM を構築．DOM の root は div.prooftree
7. 前後のテキストノードも含めて証明木 DOM をリアル DOM に反映させる.

1. KaTeX の適用

```
<!-- The loading of KaTeX is deferred to speed up page rendering -->
<script
  defer
  src="https://cdn.jsdelivr.net/npm/katex@0.16.10/dist/katex.min.js"
  integrity="sha384-hIoBPJpTUs74ddyc4bFZSM1TVlQDA60VBbJS0oA934VSz82sBx1X7kSx2ATBDIyd"
  crossorigin="anonymous"
></script>

<!-- To automatically render math in text elements, include the auto-render extension: -->
<script
  defer
  src="https://cdn.jsdelivr.net/npm/katex@0.16.10/dist/contrib/auto-render.min.js"
  integrity="sha384-43gviWU0YVjaDtb/GhzOouOXtZMP/7XUzwPTstBeZFe/+rCMvRwr4yROQP43s0Xk"
  crossorigin="anonymous"
></script>
<script>
  document.addEventListener("DOMContentLoaded", function () {
    renderMathInElement(document.body, {
      // customised options
      // • auto-render specific keys, e.g.:
      delimiters: [
        { left: "$$", right: "$$", display: true },
        { left: "$", right: "$", display: false },
      ],
      // • rendering keys, e.g.:
      throwOnError: false,
    });
  });
</script>
```

2. DOM Traversal

Marp

p 要素を取得する.

```
const nodeArray = Array.from(
  <HTMLCollectionOf<HTMLElement>>document.body.getElementsByTagName("P")
);
const nodes = nodeArray.filter((node) =>
  node.innerText.includes("\\begin{prooftree}")
);
```

3. DOM からの証明木パートの切り出し

Marp

`\begin{prooftree}...\end{prooftree}` を切り出して，証明木パートの DOM fragment を作る.

証明木をレンダリングしたら，その親の DOM 要素を再度リストに入れてやって，再度他に証明木がないか探索する.

```
interface PrtrFragment {
  // 証明木パートとその前後のテキストノードを含む，元からある DOM 要素のリスト.
  // 後でリアル DOM から削除するために持っておく.
  nodeList: HTMLElement[];

  // 証明木パートのみの DOM 要素のリスト.
  // コメントノードは除外しておく.
  prtrNodeList: HTMLElement[];

  // 証明木パートの前後のテキストノード.
  beforeTextNode: HTMLElement;
  afterTextNode: HTMLElement;
}
```

4. 証明木パートの LaTeX コードの解析

Marp

証明木パートの LaTeX コードを解析して，LaTeX コマンドのリストを作る.

```
const prtrObj: LtxCommands = parsePrtr(prtrNodeList); // Step 3.
```

LaTeX コマンドのリスト.

```
type LtxCommand =
  | { type: "AXC"; body: HTMLElement }
  | { type: "UIC"; body: HTMLElement }
  | { type: "BIC"; body: HTMLElement }
  | { type: "TIC"; body: HTMLElement }
  | { type: "QuaternaryInfC"; body: HTMLElement }
  | { type: "RightLabel"; body: HTMLElement };

type LtxCommands = LtxCommand[];
```

構文解析の要件： 以下のいずれかがゼロ個以上連続している.

5. LaTeX コマンドのリストを構文解析

LaTeX コマンドのリストを構文解析して，証明木オブジェクトを生成する.

```
type ProofTree =
  | { type: "Axiom"; axiom: string }
  | {
      type: "Sequent";
      premises: ProofTree[];
      rightlabel: HTMLElement;
      conclusion: HTMLElement;
    };

const prootTree: ProofTree = parseProofTree(ltxCommands);
```

リストを逆順にした後に，再帰降下法で構文解析する.

5. 証明木の DOM 構築

Marp

DocumentFragment 上で，証明木オブジェクトから証明木の DOM を構築.

```
const div = (label: string, children: HTMLElement[]): HTMLElement => {
  const newDiv = document.createElement("div");
  newDiv.classList.add("prtr-" + label);
  children.forEach(newDiv.appendChild);
  return newDiv;
};

const createPrtrDomHelper = (prtrDom: ProofTree): HTMLElement => {
  switch (prtrDom.type) {
    case "Axiom": {
      return div("axiom", [prtrDom.axiom]);
    }
    case "Sequent": {
      return div("sequent", [
        div("premises", prtrDom.premises.map(createPrtrDomHelper)),
        div("horizontal-rule", [div("right-label", [prtrDom.rightlabel])]),
        div("conclusion", [prtrDom.conclusion]),
      ]);
    }
  }
};

const createPrtrDom = (prtrDom: ProofTree): HTMLElement => {
  return div("prooftree", [createPrtrDomHelper(prtrDom)]);
};
```

6. リアル DOM への反映

前後のテキストノードも含めて証明木 DOM をリアル DOM に反映させる.

```
const fragment = new DocumentFragment();
fragment.append(beforeNode);
fragment.append(prtrDom);
fragment.append(afterNode);

beforeNode.parent.insertBefore(fragment, nodeList[0]);
nodeList.forEach((node: HTMLElement) => node.remove());
```

7. DOM 要素のサイズや配置のスタイル情報の更新

証明木 DOM を辿りながら要素の大きさの情報を取得し，サイズや配置のスタイル情報を更新していく．

```
// Step 6.
const applyStylesToPrtr = (prtrDom: HTMLElement) => {
  //
};

const applyStyles = () => {
  const prooftrees = Array.from(document.getElementsByClassName("prooftree"));
  prooftrees.forEach((pt) => applyStylesToPrtr(pt.children[0]! as HTMLElement)); // Step 6.
};
```

満たすべき要件：

- ラベルが干渉し合わない様な上手い配置にする必要がある．
- 推論の横線について，上下の数式の最大幅に合わせる必要がある．
  - 部分証明木全体の横幅ではない．

7.1. DOM 要素のサイズと横線の長さ計算アルゴリズム

```
-------    ----------
     P     Rhooooooo
     ---------------
           Theta
```

推論 D について，以下の値を再帰的に計算していく.

- ラベルを除いた横幅 $w(D)$
- ラベルを入れた横幅 $wl(D)$
- 結論の左側のマージン $ml(D)$
- 結論の右側のマージン $mr(D)$

結論の左右のマージンは左右で等しくない場合があることに注意.

推論 D は，前件 $P_i(i = 1, ..., n)$ と ラベル L と 結論 C からなるとする．前件の間は

7.2. レンダリングにおける CSS

Marp

ラベルは `position: absolute` にして，この幅は無視出来る様にする．

ラベル分のマージン $\max(0, wl(D) - w(D))$ は部分木の padding-right で確保する．

```
div.prtr-sequent#w7 {
  padding-right: max(0, wl(D) - w(D));
}
```

ラベルの配置はラベルの幅の分 $width(L)$ だけ右にずらす．

```
div.prtr-horizontal-rule > .prtr-right-label {
  right: -width(L);
}
```

横線の描画は，横線の幅 $hr(D)$ と，横線の左側の余り部分の長さ $m(D)$ を用いて行う．

```
div.prtr-sequent#w1 > div.prtr-horizontal-rule {
  width: hr(D);
```

7.3. コード

必要な値を計算し，スタイルを更新しながら，以下の値を返す関数を用いる．

- ラベルを除いた横幅 $w(D)$
- ラベルを入れた横幅 $wl(D)$
- 結論の左側のマージン $ml(D)$
- 結論の右側のマージン $mr(D)$

```
const applyStylesToPrtr = (node: HTMLElement): PrtrStyle => {
  if (node.classList.contains("prtr-axiom")) {
    const width = node.offsetWidth;
    return { w: width, wl: width, ml: 0, mr: 0 };
  } else if (node.classList.contains("prtr-sequent")) {
    const nodePremises = node.children[0] as HTMLElement;
    const nodeHR = node.children[1] as HTMLElement;
    const nodeConclusion = node.children[2] as HTMLElement;
    const premises = Array.prototype.slice.apply(nodePremises.children);
    if (premises.length === 0) {
      const width = nodeConclusion.offsetWidth;
      const labelWidth = nodeHR.children[0].offsetWidth;
      return { w: width, wl: labelWidth, ml: 0, mr: 0 };
    } else {
      const premisesStyles = premises.map(applyStylesToPrtr);
```

memo

前件が 0 個の推論と，推論でない公理は横線の有無で異なる.