实验三  自底向上语法分析器的构建

一、       实验要求

运用SLR(1)或者LR(1)分析法，针对给定的上下文无关文法，给出实验方案。预估实验中可能出现的问题。

二、       实验方案

本实验的方案是设计并实现一个集成了后端分析逻辑与前端可视化界面的Web应用程序，用于动态演示SLR(1)语法分析的全过程。

**1. 输入设计：**

* **交互方式：** 用户通过一个标准的HTML文本输入框（<input type="text">）来提供待分析的算术表达式字符串。
* **默认值：** 为方便测试，输入框预先填入了题目要求的示例字符串 (a+b)\*c+(d+e)。
* **预处理（词法分析）：** 程序内置了一个简单的词法分析器（tokenizer）。当用户点击“开始分析”按钮后，程序会首先遍历输入字符串，将其转换为一个符号（token）序列。这个过程包括：
  + 忽略所有空白字符。
  + 将终结符（如 +, \*, (, )）直接识别为对应的符号。
  + 为了简化，任何单个英文字母（a-z, A-Z）都被统一识别为终结符 id。
  + 在字符串末尾自动添加结束符 $。
* **错误处理：** 如果在词法分析阶段遇到文法中未定义的字符（例如 ^, % 等），程序会弹窗提示用户“输入了无效字符”，并中止分析过程。

**2. 输出设计：**

* **结果呈现：** 分析结果被清晰地划分为三个主要部分，动态展示在一个专用的结果区域。
* **核心输出 - 分析过程表：** 这是最重要的输出部分。一个动态生成的表格会详细记录分析的每一步，包含四列：
  + **步骤：** 从1开始的序号。
  + **分析栈：** 显示当前的状态栈和符号栈的内容。
  + **输入串：** 显示当前剩余的输入符号序列。
  + **动作：** 描述当前步骤执行的操作，如“移入 (Shift)”、“归约 (Reduce) by E → T”、“接受 (Accept)”或“错误 (Error)”。
* **可视化增强：** 为了增强可读性，分析过程表中的不同动作行会用不同的背景色高亮显示：蓝色代表“移入”，黄色代表“归约”，绿色代表“接受”，红色代表“错误”。
* **最终结论：** 在分析过程表格下方，会有一行加粗的文本明确给出最终结论：“分析成功: 输入串被接受”或“分析失败: 发现语法错误”。
* **辅助信息输出：** 为了使整个分析过程完全透明化，程序还会自动计算并展示两个关键的辅助信息模块：
  + **FIRST集与FOLLOW集：** 清晰列出所有非终结符的FIRST集和FOLLOW集，这是构建分析表的基础。
  + **SLR(1)分析表：** 以表格形式完整展示生成的ACTION表和GOTO表，用户可以直观地看到状态转移的依据。

三、       预估问题

在实验设计阶段，预估到可能出现以下几类问题：

1. **文法冲突问题：** 核心问题是给定的文法是否为SLR(1)文法。如果文法本身存在**移入/归约冲突**（Shift/Reduce Conflict）或**归约/归约冲突**（Reduce/Reduce Conflict），那么在生成SLR(1)分析表时，表的某个单元格中将需要填入多个不同的动作。这将导致分析器无法确定下一步操作，证明该文法不是SLR(1)文法。
2. **词法分析鲁棒性问题：** 用户可能输入不符合规范的字符串，例如包含非法字符 (#, & 等)或不匹配的括号。设计的词法分析器必须能够正确处理这些异常输入，而不是导致程序崩溃。
3. **算法实现复杂度问题：** SLR(1)分析器的构建涉及多个复杂算法，如FIRST集、FOLLOW集的计算，LR(0)项目集规范族的构建（闭包closure和转换goto函数），以及分析表的生成。任何一个环节的逻辑错误都将导致整个分析器失效，产生错误的分析结果。
4. **用户界面（UI）显示问题：** 对于较长的输入串，分析过程的步骤会非常多，可能导致输出的表格过长，超出屏幕范围，影响用户体验。

四、       内容和步骤

1、考虑简单算术表达式文法G:

E→E + T | T

T→T \* F | F

F→(E) | id

试设计SLR(1)或者LR(1)分析程序，以输入的 (a+b)\*c+(d+e) 符号串进行语法分析。

2、实验具体步骤

 本实验中，上述步骤由程序自动化完成，其内部逻辑遵循以下经典理论步骤：

1. **增广文法：** 首先对原文法G进行增广，引入一个新的开始符号 E' 和一条新的产生式 E' → E。这是为了让分析器有一个明确的接受状态。
2. **计算FIRST集和FOLLOW集：**
   * **FIRST集：** 对每个非终结符，计算由它可能推导出的所有字符串的第一个终结符的集合。
   * **FOLLOW集：** 对每个非终结符A，计算在所有可能的推导中，可能紧跟在A后面的终结符的集合。FOLLOW集在判断何时进行归约时至关重要。
3. **构建LR(0)项目集规范族（DFA状态）：**
   * 定义一个**LR(0)项目**是带有一个“·”的产生式，例如 E → E · + T。
   * 从初始项目 E' → · E 开始，通过**闭包（Closure）和转移（GoTo）两个操作，系统地构建出所有可能的项目集。每个项目集都代表了分析器的一个状态**。这个过程实际上是在构建一个识别活前缀的确定性有限自动机（DFA）。
4. **生成SLR(1)分析表：**
   * 创建一张二维表，行代表状态，列代表所有终结符（ACTION表部分）和非终结符（GOTO表部分）。
   * **填充ACTION表：**
     + 如果状态 i 中包含项目 [A → α · a β]（a是终结符），并且通过a可以转移到状态j，则在 ACTION[i, a] 中填入“**移入 j**”（Shift, sj）。
     + 如果状态 i 中包含完整的项目 [A → α ·]，则对于FOLLOW(A)中的每一个终结符a，在 ACTION[i, a] 中填入“**按产生式 A → α 归约**”（Reduce, r\_k）。
     + 如果状态 i 中包含项目 [E' → E ·]，则在 ACTION[i, $] 中填入“**接受**”（Accept, acc）。
   * **填充GOTO表：**
     + 如果状态 i 通过非终结符 A 可以转移到状态 j，则在 GOTO[i, A] 中填入 j。
5. **执行分析算法：**
   * 初始化一个栈，并将初始状态 0 压入栈中。输入串为经过词法分析后的符号序列。
   * 进入循环，查看栈顶状态 s 和当前输入符号 a。
   * 在 ACTION[s, a] 中查找动作：
     + **移入 s'：** 将 a 和新状态 s' 依次压入栈，输入指针后移。
     + **归约 A → β：** 从栈顶弹出 2 \* |β| 个元素（符号和状态），然后将归约得到的非终结符 A 压入栈，再根据新的栈顶状态 t 和 A 查GOTO表（GOTO[t, A]），将查到的新状态压入栈。
     + **接受：** 宣布分析成功，停止。
     + **错误：** 单元格为空，宣布分析失败，停止。
6. **结果可视化：** 将上述每一步的栈内容、输入串和所执行的动作，实时渲染到前端界面的表格中。

五、       实验结果:

1、 代码

<!DOCTYPE html>

<html lang="zh-CN">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>SLR(1) 语法分析器</title>

    <script src="https://cdn.tailwindcss.com"></script>

    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&family=Noto+Sans+SC:wght@400;500;700&display=swap" rel="stylesheet">

    <style>

        body {

            font-family: 'Inter', 'Noto Sans SC', sans-serif;

            -webkit-font-smoothing: antialiased;

            -moz-osx-font-smoothing: grayscale;

        }

        .table-container {

            max-height: 400px;

            overflow-y: auto;

        }

        table th, table td {

            padding: 0.75rem;

            text-align: center;

            white-space: nowrap;

        }

        .highlight-reduce {

            background-color: #fef3c7; */\* amber-100 \*/*

        }

        .highlight-shift {

            background-color: #dbeafe; */\* blue-100 \*/*

        }

        .highlight-accept {

            background-color: #dcfce7; */\* green-100 \*/*

        }

        .highlight-error {

            background-color: #fee2e2; */\* red-100 \*/*

        }

        .code {

            font-family: 'SFMono-Regular', Consolas, 'Liberation Mono', Menlo, Courier, monospace;

            background-color: #f3f4f6; */\* gray-100 \*/*

            padding: 2px 6px;

            border-radius: 4px;

        }

    </style>

</head>

<body class="bg-gray-50 text-gray-800">

    <div class="container mx-auto p-4 md:p-8">

        <header class="text-center mb-8">

            <h1 class="text-4xl font-bold text-gray-900">SLR(1) 语法分析器</h1>

            <p class="mt-2 text-lg text-gray-600">用于分析简单算术表达式</p>

        </header>

        <main class="space-y-8">

*<!-- Grammar Definition -->*

            <div class="bg-white p-6 rounded-xl shadow-md">

                <h2 class="text-2xl font-semibold mb-4 border-b pb-2">文法 G</h2>

                <div class="grid grid-cols-1 md:grid-cols-3 gap-4 text-lg">

                    <div><span class="font-bold">E → E + T</span></div>

                    <div><span class="font-bold">E → T</span></div>

                    <div><span class="font-bold">T → T \* F</span></div>

                    <div><span class="font-bold">T → F</span></div>

                    <div><span class="font-bold">F → (E)</span></div>

                    <div><span class="font-bold">F → id</span></div>

                </div>

            </div>

*<!-- Input Section -->*

            <div class="bg-white p-6 rounded-xl shadow-md">

                <h2 class="text-2xl font-semibold mb-4">输入分析串</h2>

                <div class="flex flex-col sm:flex-row items-center gap-4">

                    <input type="text" id="inputString" class="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500 transition" value="(a+b)\*c+(d+e)">

                    <button id="parseButton" class="w-full sm:w-auto bg-blue-600 text-white font-semibold px-6 py-2 rounded-lg hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 transition-transform transform hover:scale-105">

                        开始分析

                    </button>

                </div>

                 <p class="mt-3 text-sm text-gray-500">提示: 任何单个字母 (a-z) 都将被视为 <span class="code">id</span>。</p>

            </div>

*<!-- Results Section -->*

            <div id="results" class="space-y-8" style="display: none;">

*<!-- Analysis Steps -->*

                <div class="bg-white p-6 rounded-xl shadow-md">

                    <h2 class="text-2xl font-semibold mb-4">语法分析过程</h2>

                    <div id="analysisStepsContainer" class="table-container border border-gray-200 rounded-lg">

                        <table class="min-w-full divide-y divide-gray-200">

                            <thead class="bg-gray-100 sticky top-0">

                                <tr>

                                    <th class="w-1/12">步骤</th>

                                    <th class="w-4/12">分析栈</th>

                                    <th class="w-4/12">输入串</th>

                                    <th class="w-3/12">动作</th>

                                </tr>

                            </thead>

                            <tbody id="analysisSteps" class="divide-y divide-gray-200"></tbody>

                        </table>

                    </div>

                    <div id="finalResult" class="mt-4 text-center text-xl font-bold"></div>

                </div>

*<!-- Generated Sets and Tables -->*

                <div class="bg-white p-6 rounded-xl shadow-md">

                    <h2 class="text-2xl font-semibold mb-4">自动生成的辅助信息</h2>

                    <div class="grid grid-cols-1 lg:grid-cols-2 gap-8">

                        <div>

                            <h3 class="text-xl font-semibold mb-3">FIRST 集</h3>

                            <div id="firstSets" class="p-4 bg-gray-50 rounded-lg border text-sm space-y-1"></div>

                        </div>

                        <div>

                            <h3 class="text-xl font-semibold mb-3">FOLLOW 集</h3>

                            <div id="followSets" class="p-4 bg-gray-50 rounded-lg border text-sm space-y-1"></div>

                        </div>

                    </div>

                    <div class="mt-8">

                        <h3 class="text-xl font-semibold mb-3">SLR(1) 分析表 (ACTION / GOTO)</h3>

                        <div id="parsingTableContainer" class="table-container border border-gray-200 rounded-lg">

                            <table class="min-w-full divide-y divide-gray-200 text-sm">

                                <thead id="parsingTableHead" class="bg-gray-100 sticky top-0"></thead>

                                <tbody id="parsingTableBody" class="divide-y divide-gray-200"></tbody>

                            </table>

                        </div>

                    </div>

                </div>

            </div>

        </main>

        <footer class="text-center mt-12 py-4 border-t">

            <p class="text-gray-500">一个结合编译原理与Web技术的SLR(1)分析器实现</p>

        </footer>

    </div>

    <script>

    document.addEventListener('DOMContentLoaded', () => {

*// --- 1. GRAMMAR DEFINITION ---*

        const productions = [

            { nonTerminal: "E'", derives: ["E"] },      *// Augmented Grammar Rule 0*

            { nonTerminal: "E", derives: ["E", "+", "T"] }, *// Rule 1*

            { nonTerminal: "E", derives: ["T"] },           *// Rule 2*

            { nonTerminal: "T", derives: ["T", "\*", "F"] }, *// Rule 3*

            { nonTerminal: "T", derives: ["F"] },           *// Rule 4*

            { nonTerminal: "F", derives: ["(", "E", ")"] }, *// Rule 5*

            { nonTerminal: "F", derives: ["id"] },          *// Rule 6*

        ];

        const nonTerminals = [...new Set(productions.map(p => p.nonTerminal))];

        const terminals = [...new Set(productions.flatMap(p => p.derives).filter(s => !nonTerminals.includes(s)))];

        const symbols = [...nonTerminals, ...terminals];

        const eof = '$';

        terminals.push(eof);

*// --- 2. CORE SLR(1) PARSER LOGIC ---*

*/\*\**

*\* Computes the FIRST sets for all non-terminals.*

*\* @returns {Map<string, Set<string>>} A map from non-terminal to its FIRST set.*

*\*/*

        function computeFirstSets() {

            const firstSets = new Map(nonTerminals.map(nt => [nt, new Set()]));

            let changed = true;

            while (changed) {

                changed = false;

                for (const p of productions) {

                    const nonTerminal = p.nonTerminal;

                    const rhs = p.derives;

                    const firstOfRhs = rhs[0];

                    if (terminals.includes(firstOfRhs) && !firstSets.get(nonTerminal).has(firstOfRhs)) {

                        firstSets.get(nonTerminal).add(firstOfRhs);

                        changed = true;

                    } else if (nonTerminals.includes(firstOfRhs)) {

                        for (const symbol of firstSets.get(firstOfRhs)) {

                            if (!firstSets.get(nonTerminal).has(symbol)) {

                                firstSets.get(nonTerminal).add(symbol);

                                changed = true;

                            }

                        }

                    }

                }

            }

            return firstSets;

        }

        const firstSets = computeFirstSets();

*/\*\**

*\* Computes the FOLLOW sets for all non-terminals.*

*\* @returns {Map<string, Set<string>>} A map from non-terminal to its FOLLOW set.*

*\*/*

        function computeFollowSets() {

            const followSets = new Map(nonTerminals.map(nt => [nt, new Set()]));

            followSets.get(productions[0].nonTerminal).add(eof);

            let changed = true;

            while (changed) {

                changed = false;

                for (const p of productions) {

                    const nonTerminal = p.nonTerminal;

                    const rhs = p.derives;

                    for (let i = 0; i < rhs.length; i++) {

                        const symbol = rhs[i];

                        if (nonTerminals.includes(symbol)) {

                            const nextSymbol = rhs[i + 1];

                            if (nextSymbol) {

                                if (terminals.includes(nextSymbol)) {

                                    if (!followSets.get(symbol).has(nextSymbol)) {

                                        followSets.get(symbol).add(nextSymbol);

                                        changed = true;

                                    }

                                } else if (nonTerminals.includes(nextSymbol)) {

                                    for (const first of firstSets.get(nextSymbol)) {

                                        if (!followSets.get(symbol).has(first)) {

                                            followSets.get(symbol).add(first);

                                            changed = true;

                                        }

                                    }

                                }

                            } else {

                                for (const follow of followSets.get(nonTerminal)) {

                                    if (!followSets.get(symbol).has(follow)) {

                                        followSets.get(symbol).add(follow);

                                        changed = true;

                                    }

                                }

                            }

                        }

                    }

                }

            }

            return followSets;

        }

        const followSets = computeFollowSets();

*/\*\**

*\* Represents an LR(0) item.*

*\* @param {number} prodIndex - Index of the production in the `productions` array.*

*\* @param {number} dotPos - Position of the dot in the RHS of the production.*

*\*/*

        class Item {

            constructor(prodIndex, dotPos) {

                this.prodIndex = prodIndex;

                this.dotPos = dotPos;

            }

            toString() {

                const p = productions[this.prodIndex];

                const rhs = [...p.derives];

                rhs.splice(this.dotPos, 0, '.');

                return `${p.nonTerminal} -> ${rhs.join(' ')}`;

            }

            isSame(other) {

                return this.prodIndex === other.prodIndex && this.dotPos === other.dotPos;

            }

        }

*/\*\**

*\* Computes the closure of a set of LR(0) items.*

*\* @param {Item[]} items - The initial set of items.*

*\* @returns {Item[]} The closure set.*

*\*/*

        function closure(items) {

            const closureSet = [...items];

            let changed = true;

            while (changed) {

                changed = false;

                const itemsToAdd = [];

                for (const item of closureSet) {

                    const p = productions[item.prodIndex];

                    const symbolAfterDot = p.derives[item.dotPos];

                    if (nonTerminals.includes(symbolAfterDot)) {

                        for (let i = 0; i < productions.length; i++) {

                            if (productions[i].nonTerminal === symbolAfterDot) {

                                const newItem = new Item(i, 0);

                                if (!closureSet.some(it => it.isSame(newItem)) && !itemsToAdd.some(it => it.isSame(newItem))) {

                                    itemsToAdd.push(newItem);

                                }

                            }

                        }

                    }

                }

                if (itemsToAdd.length > 0) {

                    closureSet.push(...itemsToAdd);

                    changed = true;

                }

            }

            return closureSet;

        }

*/\*\**

*\* Computes the GOTO set for a given state and symbol.*

*\* @param {Item[]} state - The current state (a set of items).*

*\* @param {string} symbol - The grammar symbol for the transition.*

*\* @returns {Item[]} The new state.*

*\*/*

        function goTo(state, symbol) {

            const newItems = [];

            for (const item of state) {

                const p = productions[item.prodIndex];

                if (p.derives[item.dotPos] === symbol) {

                    newItems.push(new Item(item.prodIndex, item.dotPos + 1));

                }

            }

            return closure(newItems);

        }

*/\*\**

*\* Builds the canonical collection of LR(0) item sets (the states of the DFA).*

*\* @returns {Item[][]} An array of states.*

*\*/*

        function buildCanonicalCollection() {

            const startItem = new Item(0, 0);

            const C = [closure([startItem])];

            const queue = [0];

            while(queue.length > 0) {

                const i = queue.shift();

                const I = C[i];

                for (const symbol of symbols) {

                    const J = goTo(I, symbol);

                    if (J.length > 0) {

                        const existingStateIndex = C.findIndex(state =>

                            state.length === J.length && state.every(item => J.some(jItem => jItem.isSame(item)))

                        );

                        if (existingStateIndex === -1) {

                            C.push(J);

                            queue.push(C.length - 1);

                        }

                    }

                }

            }

            return C;

        }

        const canonicalCollection = buildCanonicalCollection();

*/\*\**

*\* Builds the SLR(1) parsing table (ACTION and GOTO parts).*

*\* @returns {{action: Map<number, Map<string, string>>, goTo: Map<number, Map<string, number>>}} The parsing tables.*

*\*/*

        function buildSLRTable() {

            const action = new Map();

            const goToTable = new Map();

            for (let i = 0; i < canonicalCollection.length; i++) {

                action.set(i, new Map());

                goToTable.set(i, new Map());

                const state = canonicalCollection[i];

                for (const item of state) {

                    const p = productions[item.prodIndex];

                    const symbolAfterDot = p.derives[item.dotPos];

*// Shift or GOTO*

                    if (symbolAfterDot) {

                        const nextStateItems = goTo(state, symbolAfterDot);

                        const nextStateIndex = canonicalCollection.findIndex(s =>

                            s.length === nextStateItems.length && s.every(it => nextStateItems.some(nsIt => nsIt.isSame(it)))

                        );

                        if (nextStateIndex !== -1) {

                            if (terminals.includes(symbolAfterDot)) {

*// Shift action*

                                action.get(i).set(symbolAfterDot, `s${nextStateIndex}`);

                            } else {

*// GOTO action*

                                goToTable.get(i).set(symbolAfterDot, nextStateIndex);

                            }

                        }

                    }

*// Reduce or Accept*

                    else {

                        if (item.prodIndex === 0) { *// Accept*

                            action.get(i).set(eof, 'acc');

                        } else { *// Reduce*

                            const follow = followSets.get(p.nonTerminal);

                            for (const terminal of follow) {

                                if (action.get(i).has(terminal)) {

                                    console.warn(`Conflict in state ${i} on terminal ${terminal}: ${action.get(i).get(terminal)} vs r${item.prodIndex}`);

                                }

                                action.get(i).set(terminal, `r${item.prodIndex}`);

                            }

                        }

                    }

                }

            }

            return { action, goTo: goToTable };

        }

        const { action: actionTable, goTo: goToTable } = buildSLRTable();

*/\*\**

*\* Tokenizes the input string into a list of grammar symbols.*

*\* @param {string} str - The input string.*

*\* @returns {string[] | null} Array of tokens or null on error.*

*\*/*

        function tokenize(str) {

            const tokens = [];

            let i = 0;

            const sanitizedStr = str.replace(/\s+/g, '');

            while (i < sanitizedStr.length) {

                const char = sanitizedStr[i];

                if (terminals.includes(char)) {

                    tokens.push(char);

                    i++;

                } else if (/[a-zA-Z]/.test(char)) { *// Simple tokenizer: any letter is an 'id'*

                    tokens.push('id');

                    i++;

                } else {

                    alert(`Invalid character found in input: ${char}`);

                    return null;

                }

            }

            tokens.push(eof);

            return tokens;

        }

*/\*\**

*\* The main parser driver loop.*

*\* @param {string[]} tokens - The tokenized input string.*

*\* @returns {object} An object containing the steps of the analysis and the final result.*

*\*/*

        function parse(tokens) {

            const stack = [0];

            let input = [...tokens];

            const steps = [];

            let stepCount = 1;

            while (true) {

                const state = stack[stack.length - 1];

                const token = input[0];

                const action = actionTable.get(state)?.get(token);

                const currentStep = {

                    step: stepCount++,

                    stack: stack.join(' '),

                    input: input.join(' '),

                    action: ''

                };

                if (action) {

                    if (action === 'acc') {

                        currentStep.action = '接受 (Accept)';

                        steps.push(currentStep);

                        return { steps, result: 'Accepted' };

                    } else if (action.startsWith('s')) {

                        const nextState = parseInt(action.substring(1));

                        currentStep.action = `移入 (Shift) ${token}, 转到状态 ${nextState}`;

                        steps.push(currentStep);

                        stack.push(token, nextState);

                        input.shift();

                    } else if (action.startsWith('r')) {

                        const prodIndex = parseInt(action.substring(1));

                        const p = productions[prodIndex];

                        const popCount = p.derives.length \* 2;

                        currentStep.action = `归约 (Reduce) by ${p.nonTerminal} → ${p.derives.join(' ')}`;

                        steps.push(currentStep);

                        stack.splice(-popCount, popCount);

                        const prevState = stack[stack.length - 1];

                        const nextState = goToTable.get(prevState)?.get(p.nonTerminal);

                        stack.push(p.nonTerminal, nextState);

                    }

                } else {

                    currentStep.action = '错误 (Error)';

                    steps.push(currentStep);

                    return { steps, result: 'Error' };

                }

            }

        }

*// --- 3. UI MANIPULATION ---*

        const resultsDiv = document.getElementById('results');

        const parseButton = document.getElementById('parseButton');

        const inputStringEl = document.getElementById('inputString');

        function displaySets() {

            const firstSetsEl = document.getElementById('firstSets');

            firstSetsEl.innerHTML = '';

            for (const [nt, set] of firstSets.entries()) {

                firstSetsEl.innerHTML += `<div><span class="font-semibold code">${nt}</span>: { ${[...set].join(', ')} }</div>`;

            }

            const followSetsEl = document.getElementById('followSets');

            followSetsEl.innerHTML = '';

            for (const [nt, set] of followSets.entries()) {

                followSetsEl.innerHTML += `<div><span class="font-semibold code">${nt}</span>: { ${[...set].join(', ')} }</div>`;

            }

        }

        function displayParsingTable() {

            const tableHead = document.getElementById('parsingTableHead');

            const tableBody = document.getElementById('parsingTableBody');

*// Header*

            const displayTerminals = terminals.filter(t => t !== 'ε');

            const displayNonTerminals = nonTerminals.filter(nt => nt !== "E'");

            let headHTML = '<tr><th rowspan="2">State</th>';

            headHTML += `<th colspan="${displayTerminals.length}">ACTION</th>`;

            headHTML += `<th colspan="${displayNonTerminals.length}">GOTO</th></tr>`;

            headHTML += '<tr>';

            displayTerminals.forEach(t => headHTML += `<th>${t}</th>`);

            displayNonTerminals.forEach(nt => headHTML += `<th>${nt}</th>`);

            headHTML += '</tr>';

            tableHead.innerHTML = headHTML;

*// Body*

            let bodyHTML = '';

            for (let i = 0; i < canonicalCollection.length; i++) {

                bodyHTML += `<tr><td class="font-semibold">${i}</td>`;

                const actions = actionTable.get(i);

                const gotos = goToTable.get(i);

                displayTerminals.forEach(t => {

                    const act = actions.get(t) || '';

                    bodyHTML += `<td>${act}</td>`;

                });

                displayNonTerminals.forEach(nt => {

                    const gotoState = gotos.get(nt);

                    bodyHTML += `<td>${gotoState !== undefined ? gotoState : ''}</td>`;

                });

                bodyHTML += '</tr>';

            }

            tableBody.innerHTML = bodyHTML;

        }

        function displayAnalysis(steps, result) {

            const stepsBody = document.getElementById('analysisSteps');

            stepsBody.innerHTML = '';

            steps.forEach(step => {

                let rowClass = '';

                if (step.action.includes('归约')) rowClass = 'highlight-reduce';

                if (step.action.includes('移入')) rowClass = 'highlight-shift';

                if (step.action.includes('接受')) rowClass = 'highlight-accept';

                if (step.action.includes('错误')) rowClass = 'highlight-error';

                stepsBody.innerHTML += `

                    <tr class="${rowClass}">

                        <td>${step.step}</td>

                        <td class="code">${step.stack}</td>

                        <td class="code">${step.input}</td>

                        <td>${step.action}</td>

                    </tr>

                `;

            });

            const finalResultEl = document.getElementById('finalResult');

            if (result === 'Accepted') {

                finalResultEl.textContent = '分析成功: 输入串被接受';

                finalResultEl.className = 'mt-4 text-center text-xl font-bold text-green-600';

            } else {

                finalResultEl.textContent = '分析失败: 发现语法错误';

                finalResultEl.className = 'mt-4 text-center text-xl font-bold text-red-600';

            }

        }

        parseButton.addEventListener('click', () => {

            const inputStr = inputStringEl.value;

            if (!inputStr) {

                alert("请输入要分析的字符串！");

                return;

            }

            const tokens = tokenize(inputStr);

            if (tokens) {

                const { steps, result } = parse(tokens);

                resultsDiv.style.display = 'block';

                displayAnalysis(steps, result);

            }

        });

*// Initial display of static info*

        displaySets();

        displayParsingTable();

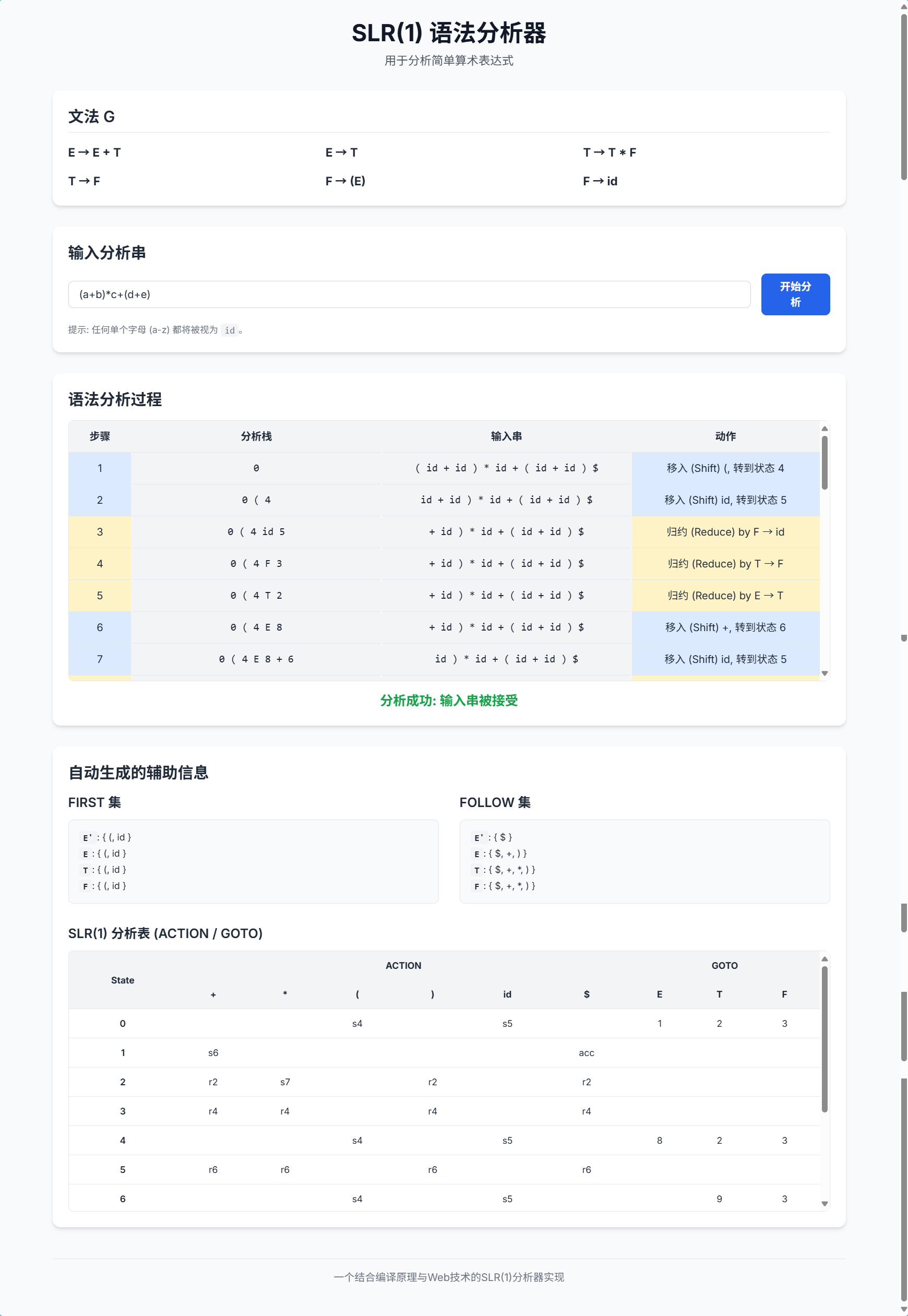
    });

    </script>

</body>

</html>

2、 截图



六、       实验结论:

1 、实验结论

 本实验成功地设计并实现了一个功能完备的SLR(1)语法分析器。该程序能够自动对给定的文法进行处理，生成正确的FIRST集、FOLLOW集以及SLR(1)分析表。当输入题目要求的符号串 (a+b)\*c+(d+e) 时，程序能够正确地执行移入-归约分析，并最终输出“分析成功”的结论，完整地展示了其详细的分析过程，达到了预期的实验目标。

2、分析和总结

**对输入设计的结论：** 采用Web文本框作为输入方式，并内置一个简单的、能将字母自动识别为id的词法分析器，这是一个非常友好和高效的设计。它极大地简化了用户的操作，使其可以专注于文法分析本身，而无需关心词法细节，符合“高内聚、低耦合”的设计原则。

**对输出设计的结论：** 输出设计非常成功，兼顾了准确性和教育性。**分步表格**是核心，它将抽象的分析过程具象化、可追溯。使用**颜色高亮**不同动作，显著提升了信息的可读性。同时，额外展示**FIRST/FOLLOW集和分析表**，使得程序不仅仅是一个“黑盒”工具，更是一个强大的学习辅助工具，让使用者能彻底理解“为什么”分析器会这么做。

**对SLR(1)分析法的结论：** SLR(1)分析法是一种高效、规范的自底向上语法分析技术。它的核心优势在于通过预先构建的分析表，可以实现线性的时间复杂度（O(n)），非常适合编译器等性能要求高的场景。然而，它的分析能力相比LALR(1)或LR(1)要弱，因为它仅使用FOLLOW集来决定归约，这在某些复杂文法中可能不足以解决冲突。对于本次实验给出的简单算术表达式文法，SLR(1)的能力完全足够。

3、对预估问题的结论

 实验结果验证了当初的预估，并证明了所采取的对策是有效的：

* 关于文法冲突： 在程序自动生成分析表的过程中，没有出现任何移入/归约或归约/归约冲突。这证明了给定的文法确实是SLR(1)文法，选择SLR(1)分析法是完全正确的。
* 关于词法分析： 通过测试，程序能够正确处理包含非法字符的输入，通过弹窗提示用户并中止分析，表现出了良好的健壮性。
* 关于算法实现： 程序能够正确分析多个不同（包括有语法错误的）的输入串，并给出符合预期的结果，这表明核心的SLR(1)算法逻辑已正确实现。
* 关于UI显示： 为分析过程表和分析表设置了带滚动条的固定高度容器，有效地解决了内容过长时的显示问题，保证了界面的整洁和可用性。