|  |
| --- |
|  |
| 程序静态分析可视化 |
| 信息可视化项目文档 |

|  |
| --- |
| 学号：2017213854 姓名：赵越 班级：软硕172班  学号：2017213852 姓名：傅滢 班级：软硕172班  学号：2017213851 姓名：吴捷成 班级：软硕172班  2017-12-31 |

目录

[第一章 项目背景 2](#_Toc502432904)

[1.1 软件缺陷及其影响 2](#_Toc502432905)

[1.2 程序静态分析技术 2](#_Toc502432906)

[第二章 项目综述 2](#_Toc502432907)

[2.1 总体介绍 3](#_Toc502432908)

[2.2 点击源代码边进行展开 3](#_Toc502432909)

[2.3 上传文件 4](#_Toc502432910)

1. 项目背景
   1. 软件缺陷及其影响

随着计算机的发明以及计算机技术的快速发展和不断普及，软件已经成为当今社会不可缺少的一部分,深刻地改变着人类生产和生活方式。但与此同时，不可否认的事实是所有软件都存在缺陷。

软件缺陷是导致相关系统出错、失效、崩溃的潜在根源，特别在安全攸关的领域如银行、航天、航空等领域，仅仅一个软件缺陷就可能引起灾难性的后果，造成巨大的经济财产损失，甚至会威胁人们的生命安全。软件项目中存在的但未被发现的缺陷，就像是一个“定时炸弹”，无时不刻威胁着人们的生命财产安全。

软件缺陷能引起软件系统故障甚至失效从而导致严重的生命财产损失。而且，在软件项目生命周期的各个阶段，即使是针对同一个软件缺陷，修复成本也会大不相同。随着时间的推移，软件缺陷的修复费用是呈指数级增长的。因此，要尽可能早地发现缺陷并修复缺陷。能否尽早移除软件系统中存在的软件缺陷，是确保软件质量的关键所在。

* 1. 程序静态分析技术

程序静态分析技术是一种有效的寻找软件缺陷的技术，利用程序静态分析技术，能够不需要运行被分析程序而对程序进行检测的方法，具有如下的优点：

1) 尽早的检测出潜在的程序错误：静态分析能够在运行程序前，对代码进行分析，以更早的发现代码中的错误；有些支持模块化分析的技术，甚至不需要完整的程序就能对独立的模块进行分析和检测。

2) 覆盖率高：静态分析能够全面系统地探索到程序的所有可能执行情况，从而达到较高的覆盖率，可以非常有效的发现一些较为隐蔽的程序错误。

3) 适用性强，代价低：静态分析不需要运行程序，能够适用各种类型的代码，同时不会引入运行时的时间和空间代价。

目前许多程序静态分析工具被广泛应用在各个软件公司的开发过程中，但是对于静态分析的可视化却没有得到很好的支持，例如经典程序静态分析工具Coverity的可视化虽然能够显示出错原因，但是缺陷相关的文字信息不直观并且缺少程序运行时的环境信息，程序员无法从Coverity的可视化中立即获取程序为什么出错的原因。

1. 项目综述

本项目所做的工作是对C语言程序静态分析进行可视化。可视化数据来源为清华大学软件学院系统所Tsmart 程序静态分析工具得到的对被测C语言程序的分析结果，目的是将程序静态分析得到的信息（例如代码执行过程中内存的变化情况等）以直观直接的方式呈现给用户，方便用户快速了解程序执行过程中的各种环境信息，帮助用户进行代码审查，提高代码开发效率。

接下来的内容是本可视化项目的各个功能展示。

* 1. 总体介绍

如图 1所示，项目的总体界面分为三个主要的区域：

1. 源代码展示区

在界面的左上角是源代码展示区，此处展示被分析程序的源代码，代码经过高亮处理，且之后会随着用户点击发生颜色变化，区分当前展示的程序状态是执行到了那一条代码，具体效果见后图所示。

1. 程序运行展示区

界面的中间区域是程序运行展示区，是可视化的主体部分。可以看到程序运行展示区类似一张程序流程图，其中节点中存储的信息是当前环境的状态（例如内存信息），节点内的信息可以进一步展开，具体效果见后图所示；边上标注的是源代码信息，边的导向代表从上一个节点的状态到下一个的节点的状态的转换是通过执行了边上的代码而实现的。

1. 功能图标区

界面的右下角是功能图标区，使用形象化的图标。左边的向上的蓝色箭头图标代表上传，用户可以选择合适的.dot文件进行程序静态分析的信息可视化；右边的蓝色×图标代表删除，用户点击图标可以删除当前的图像。

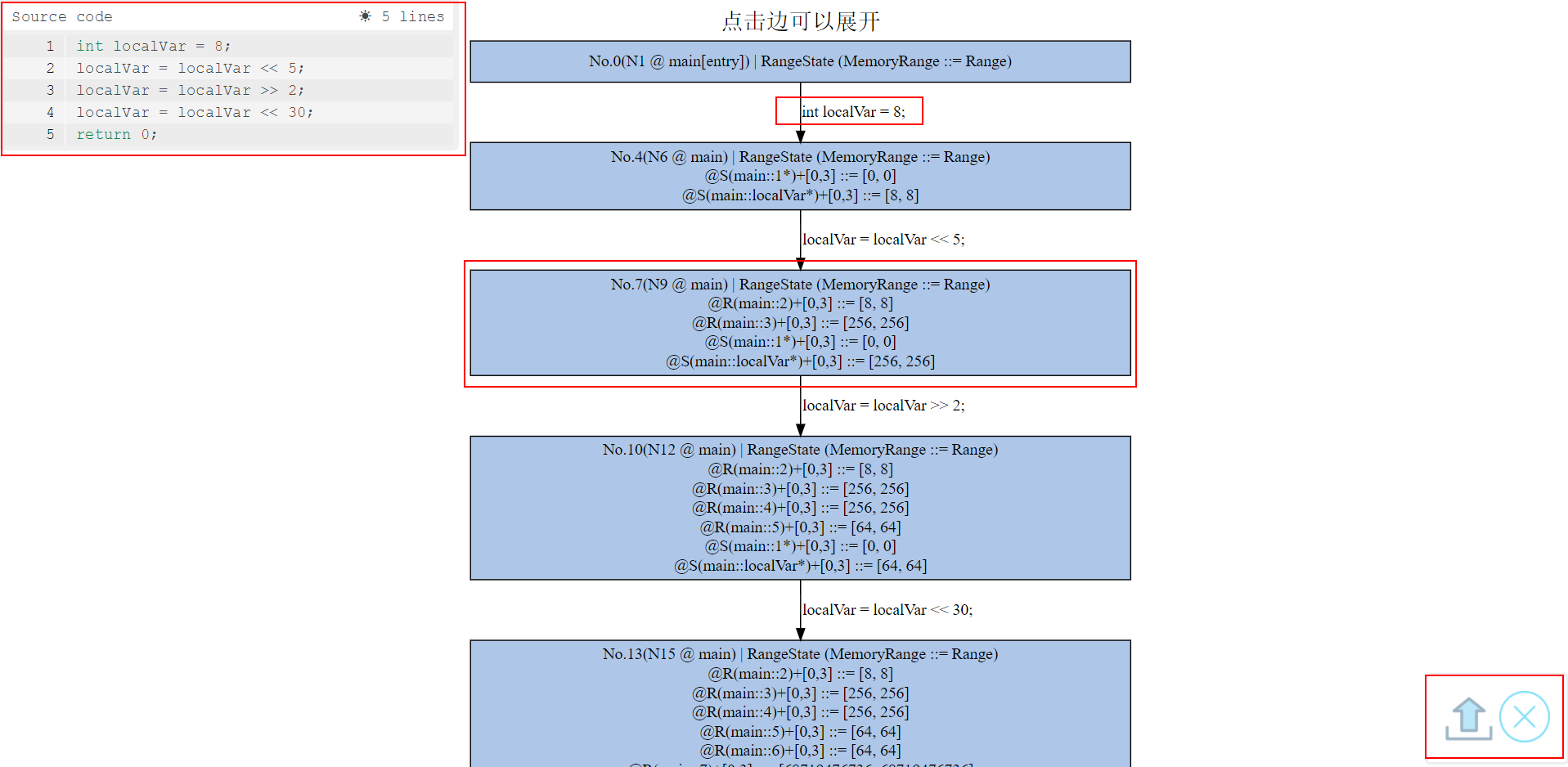


图 1-项目总体界面

* 1. 点击源代码边进行展开

点击蓝色图的边或者点击源代码展示的代码之后，可以将程序运行的结果进一步细化展示，如图 2所示。C语言程序在实际执行时不是直接执行源代码，而是机器执行编译之后的机器指令，为了更好地模拟显示代码的执行过程，本项目支持可视化显示LLVM代码执行后的环境信息，如图 2中右边红色图即为程序执行“int local Var=8”这句源代码时，实际执行的LLVM代码和程序环境状态的显示。可以注意到，当点开一条源代码的边时，左上角的代码显示区中的源代码会相应变红。

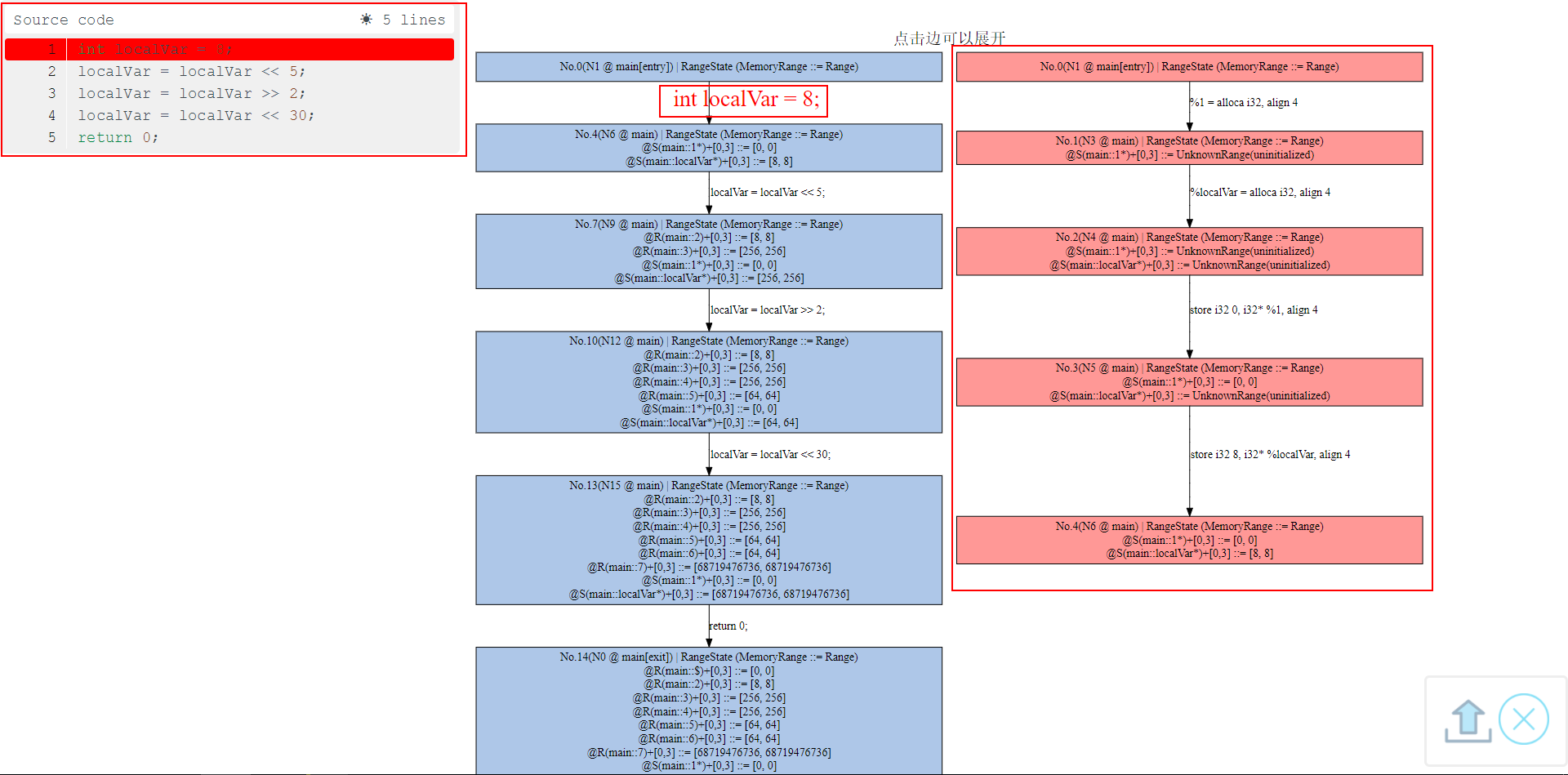


图 2-点击展开子状态

* 1. 上传文件

点击右下角功能图标区的上传图标后，用户可以自行选择合适的.dot文件进行上传，本项目可以将新文件可视化展示。

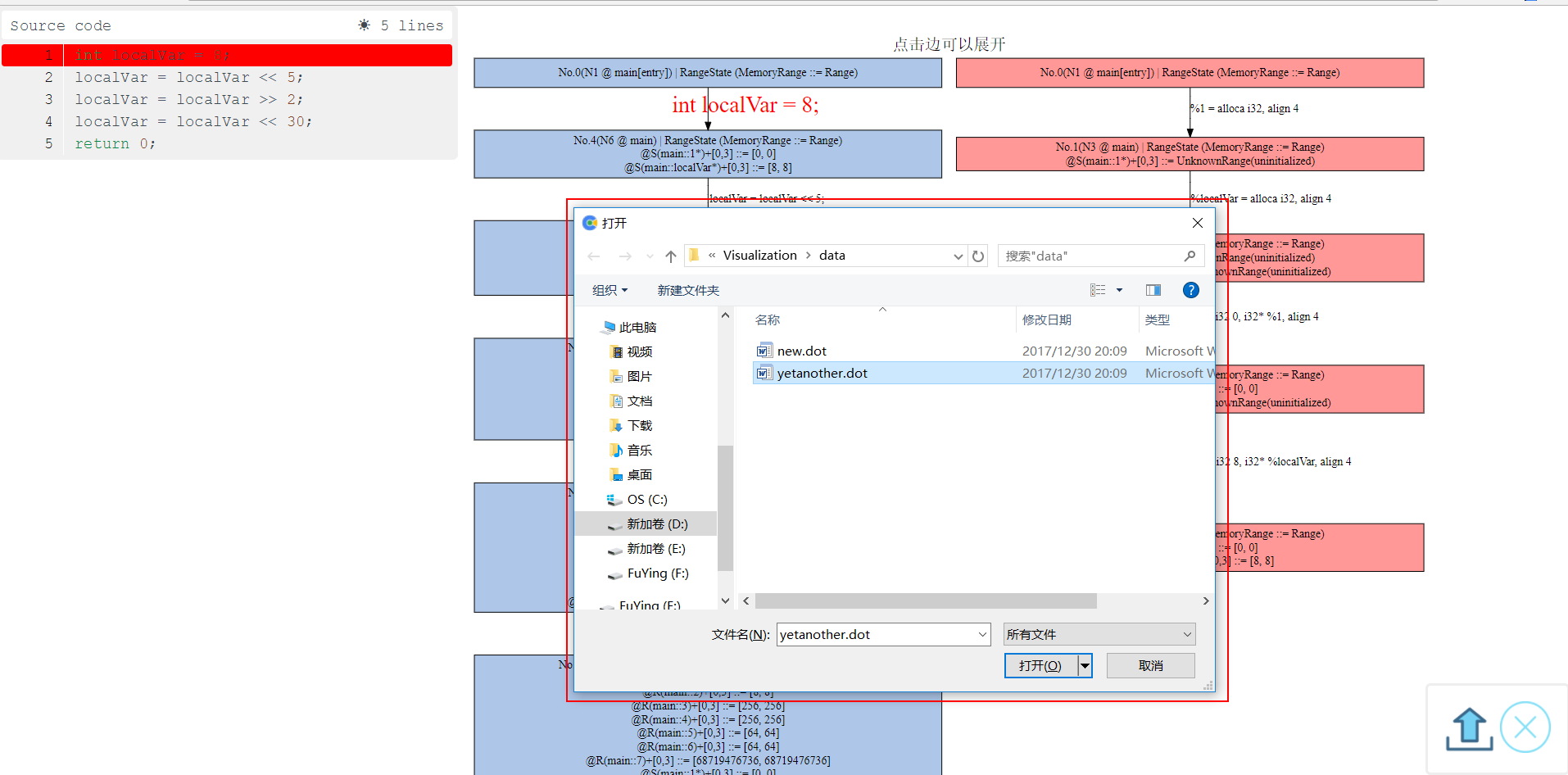


图 3-上传文件功能

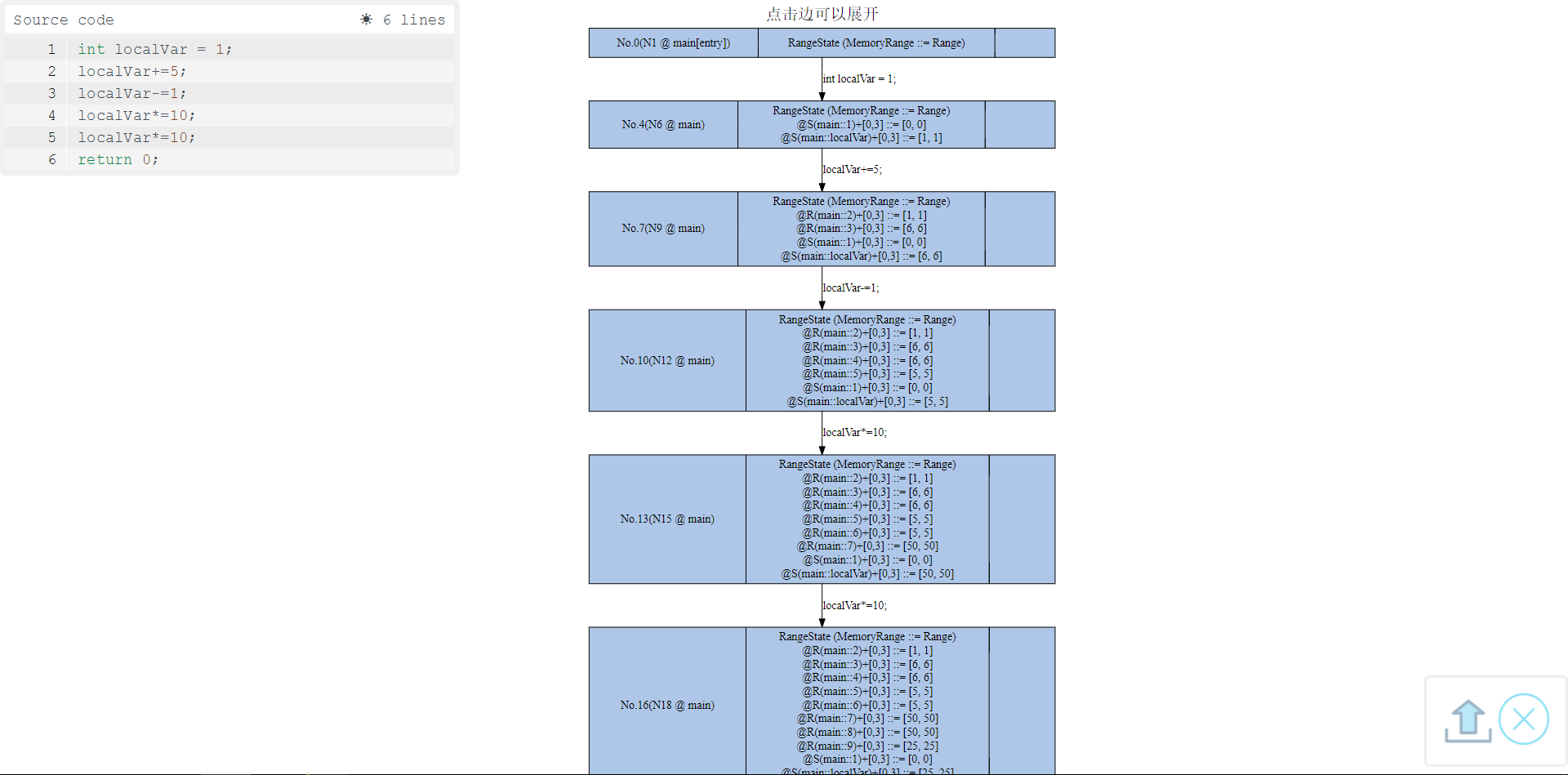


图 4-新文件上传后的显示效果

1. 数据输入
   1. 数据格式

本项目展示的实际上是被测程序的可达状态(reachability)，用抽象可达树(ARG, abstract reachability tree)表示，它是一个有向图，节点代表当前程序状态时程序内存情况，边代表执行一次跳转，通常为程序的一行代码或一次指令。

本项目依托的另一个理论是晶格理论(lattice theory)。举例来说，如下的一段代码，当变量a<5或者a≥5时，程序会有两种执行路径，导致程序执行到(2)时可能会有2种情况，当出现了许多这种分支语句时，程序总的执行状态呈现指数式爆炸，对程序分析不利。

if (a<5) {

... // (1)

}

... // (2)

晶格理论允许定义域(domain)的上界(top, ⊤)、下界(bottom, ⊥)以及在上下界之间的元素。通常，上界代表全集，下界代表空集。以整型数(Integer)为例，其上界为语言支持的最大值和最小值（c语言是[-2147483648, 2147483647]），下界为“未初始化”，元素为语言可表示的整型数值值/范围。使用晶格理论，在上面的例子中，若a<5，程序运行至(2)时的状态为[0,5]；若a≥5，程序运行至(2)状态为[-5,5]（一方包含另一方）,则两个状态可以合并，大大减少程序的总执行状态。

根据以上理论，我们需要一个可以展示有向图的方式以展示抽象可达树。DOT语言是一种文本图形描述语言。它提供了一种简单的描述图形的方法，并且可以为人类和计算机程序所理解。只需如下定义即可展示一个有向图。

digraph {

a -> b -> c;

b -> d;

}

本项目输入的.dot数据格式如下：

[

digraph {

graph [attributes]

node [style="filled" shape="box"]

a [label="xxx" id="a"]

b [label="xxx" id="b"]

c [label="xxx" id="c"]

...

a -> b [label="yyy" id="a -> b"];

b -> c [label="yyy" id="b -> c"];

...

}

],

[

a -> b {

d [label="xxx" id="d"]

e [label="xxx" id="e"]

f [label="xxx" id="f"]

...

d -> e [label="yyy" id="d -> e"];

e -> f [label="yyy" id="e -> f"];

...

}

]

...

digraph定义的是主状态（图2蓝色部分）的程序状态图。“a [label="xxx" id="a"]”定义节点，”label”属性定义节点内容（即蓝色方框内的文字）；”a -> b”定义一条由节点”a”到节点”b”的有向边，”label”属性定义边上的文字，这里为被测程序的每一行源代码。

除digraph之外定义的是子状态（图2红色部分）。这里面定义的节点与边为子状态的节点与边，但子状态的边为细化的llvm指令而非被测程序的源代码。

* 1. 生成过程

本项目输入的.dot文件是清华大学软件学院系统所Tsmart 程序静态分析工具对被测程序分析时输出的副产物，Tsmart在分析被测c语言程序时，先将被测程序编译为llvm的中间文件，目的是消除程序的复杂性，将其每一行源代码编译为多行原子指令，Tsmart分析的输入为每一行llvm指令，每一行指令执行完成后，计算得到的当前被测程序的内存表示情况输出到.dot文件中。