贴心云衣柜
——基于机器学习和社交网络的服装搭配推荐

曹佳涵 [1)]　金亦凡 [1)]　李鑫烨 [1)]　李光耀 [1)]

[1)](南京大学计算机科学与技术系 南京 210093)

摘　要　随着物质生活水平的不断提高，服装搭配已经成为一个出门前必须进行的活动。它既是展现自我个性与风采的恰当途径，又有着十分重要的社交意义。在这一背景下，本项目开发了一款基于机器学习和社交网络的服装搭配推荐 APP——贴心云衣柜（SweetWard）。与市面上现有 APP 的差异之处在于，该 APP 不仅可以根据天气、场合等，从用户已有的衣物集当中为用户生成合理的穿衣搭配，还可以通过用户的穿衣历史信息和衣物搭配收藏，运用一种新兴的个性化推荐算法——协同过滤算法，向用户推荐社交网络中与该用户风格相近的用户以及他们的穿衣搭配收藏。文中对本 APP 的开发背景、工具及算法、实现过程、成品展示等进行了详细的阐述。

关键词　Android；服装搭配推荐；机器学习；协同过滤；社交网络

SweetWard
—Dressing recommendations based on machine learning and social networking

Jiahan Cao [1)]　Yifan Jin [1)]　Xinye Li [1)]　Guangyao Li [1)]

[1)](Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

Abstract　With the continuous improvement of material standard of living, dressing has become a necessary activity before going out. It not only is an appropriate way to show one's personality and style, but also has very important social significance. In this context, SweetWard, a dressing recommendation APP based on machine learning and social network, was developed in this project. What's different from existing apps on the market is that, this APP can not only generate reasonable clothing collocation for users from their existing clothing collection according to the weather, occasions and so on, but also recommend users with similar styles and their clothing collocation collection in social networks to the certain user based on his or her dressing history information and clothing collocation collection, applying a new personalized recommendation algorithm, collaborative filtering algorithm. In this paper, the development background, tools and algorithms, implementation process and finished product display of the APP are described in detail.

Key words　Android；dress recommendation；machine learning；collaborative filtering；social networking

1　引言

1.1. 项目开发背景

在物质生活水平不断提高的现代社会，人们对于服装搭配的要求也在一天天地提高：不仅要做到舒适得体，还要用不同的穿搭应对不同的场合。可以说，服装搭配已经成为了现代人出门之前必须要解决的一个问题。然而与此同时，随着生活节奏的进一步加快，思考如何搭配着装和选择购买新衣物的时间也愈加缺乏。在这种情况下，我们不难发现，一个智能的、个性化的，能应对上述诸多问题的穿衣搭配推荐 APP，必然能够吸引一个很大的用户群体，在 APP 市场中占有一席之地。

1.2. 需求分析

在项目开发的前期准备时，开发小组访问了 APP 商店，发现市场中已经存在很多有关穿衣搭配的 APP，但是这些 APP 的推荐功能较为基础，它们只是简单地处理了用户现有

的衣物的搭配问题，并没有推荐系统的处理。事实上，用户更加青睐于个性化的 APP，期望 APP 能够符合自己的品味。更进一层的需求是满足大多数人都具有的融入集体的心理，在社交网络中分享自己的风格，并找到同道中人。

以网易云音乐为例，该款 APP 的成功之处并不在于其与其他音乐 APP 相同或相似的音乐播放功能或庞大的音乐库，而是在于它特有的个性化音乐推荐功能，以及完善的社交功能，包括音视频分享功能和评论功能等。用户可以通过强大的推荐功能找到更多符合自己听歌风格的歌曲，丰富个人歌单；同时用户还能够评论、分享自己喜欢的歌曲，参与到社区互动，也可以找到听歌风格相近的用户集群。这种用户体验，是一个单纯的音乐播放器所无法提供的。

因此，在分析大多数用户的需求、并与市面上的同类型或同性质的 APP 进行横纵向比较之后，开发小组明确了开发方向，即开发一款这样的 APP：它既有普通穿衣搭配 APP 的功能（包括衣物管理系统、服装搭配收藏系统以及基于用户现有衣物和天气等客观信息的搭配推荐系统），又有类似于网易云音乐的个性化推荐系统和社交系统。用户能够收到 APP 为其提供的符合该用户个性的一套或多套穿衣搭配（内含的衣物单品可能不在用户现有衣物中），也能够参加社交活动，查看好友的衣物搭配，同时在社交网络中寻找到与其风格类似的其他用户。

1.3. 开发意义

从用户体验的角度分析，该 APP 能够提供当下市面上各种穿衣搭配 APP 所不能提供的一些用户体验。当用户想要在自己的衣柜中增添新衣物时，个性化的推荐功能可以更快地给用户指明方向；同时，在用户社群中找到风格类似的同道中人也能为用户带来极好的体验感，用户之间可以互相借鉴穿衣搭配，交流某种风格的穿搭心得等。

项目主要阶段以实现功能为主，在开发完成之后如果对整个项目进行优化，包括界面优化、性能优化等，那么这款 APP 成品必定能够占据一定市场，吸引一定量的用户集群。

1.4.

2. 概述

2.1. 项目功能简述

本项目开发的 APP 的核心功能有二：第一，根据天气等一些外在条件，在现有的衣物中为用户推荐一套适当的衣物搭配；第二，建立一个社交网络，根据该用户的穿衣风格，从社交网络中寻找与其风格相近的其他用户，并将这些用户或他们的衣物搭配收藏推荐给该用户。项目以这两个功能为核心进行架构。

除此之外，该 APP 还有一些优先级较低但同样十分必要的功能，例如天气、个人衣柜、服装搭配收藏、新衣物导入，以及一些社交网络的基本功能（好友系统、朋友圈等）。

2.2. 项目框架介绍

从功能列表中，我们能够大致将所有功能分为三个大类：演示、存储和计算。演示是 APP 最基础的功能，用户的各种个人信息、为用户推荐的各种信息以及其他种种用户会想看到的信息需要在手机界面上向用户演示；用户的个人信息，例如衣物收藏等，在每次启动 APP 时均需要进行加载，故需要选择一个合适的存储方式；最后，为用户推荐信息需要进行大量的计算。

鉴于用户对 APP 的轻量化需求，开发小组决定开设一台服务器，将所有计算工作以及部分存储工作转移至服务器，以达到轻量化的目标。用户集的信息存储在服务器中，本用户的一些信息（例如衣物收藏）则存储在本地；推荐衣物搭配及用户的计算过程放在服务器端，通过 Socket 架设接口传输推荐请求和推荐结果。客户端的主要任务则是演示。

2.3. NuSMV介绍

模型检验是一种自动验证有穷状态机是否满足规范的形式化方法。目前已有较多的模型检验工具，如 SPIN[7]、NuSMV。NuSMV 擅长同步系统的验证[8]，更适合于硬件描述语言的验证，本项目模型检验功能基于 NuSMV。

NuSMV 含义是 New Symbolic Model Verifier，是对第一代 SMV 的重新实现，加入了很多新的模型检测算法，采用 CUDD（Colorado University Decision Diagram）包提供高效的 BDD（Binary decision diagram，二叉决策图）操作，缓解了模型检验中状态爆炸的问题。

模型检验中通常使用线性时态逻辑（Linear-time Temporal Logic, LTL）和计算树逻辑（Computation Tree Logic, CTL）。NuSMV 对两种逻辑均支持。一个典型的用 smv 语言描述的有穷状态机的结构如下：

```
MODULE main
    VAR …
    ASSIGN …
MODULE submodule
    …
SPEC …
```

NuSMV 必须有一个 main 模块，main 模块可以对其他模块进行实例化，SPEC 是对该状态机的规约，即要验证的属性。变量有两大类，一类是 State Variables（用 VAR 表示，状态的值表示某个状态），另一类是 Input Variables（通过关系表示状态）。对状态机的描述有 ASSIGN、TRANS、INIT 等等语句。

3. 基于 PyVerilog 的语法树提取

3.1. 提取过程

语法树的提取是一个编译原理范畴内的工作，这是整个项目的第一步，也是将具体的 Verilog 代码抽象化，化为一个有层次数据结构进行存储的一步。一般来说语法树的提取需要进行词法分析，语法分析[9]。词法分析是将程序分割成一个个词素流，语法分析则是在词素的基础上将他们按照规定的产生式从终结符号生成非终结符号并组织成树。考虑到 Verilog 语法的复杂性所以我们使用了 PyVerilog[10]来辅助我们生成关于 Verilog 的语法树。PyVerilog 是一个专门用来将 Verilog 代码的语法层次化结构化输出的工具，非常便于进行 Verilog 代码的语法分析，有了这个工具，我们可以直接跳过词法分析的步骤，直接生成语法树。生成语法树的过程就是通过读入 PyVerilog 产生的语法结构然后根据不同的语法模块生成对应的语法节点，在特定的节点中写入必要的信息，然后将整个节点组织成一棵树。

3.2. 语法树结构

在程序中所采取的树的结构是子女-父母-兄弟链表树。即，每个节点有一根直接指向第一个子女节点的指针，有一根直接指向自己下一个兄弟节点的指针，（除根节点之外）有一根直接指向自己父节点的指针。之所以要有指向父节点，子节点和兄弟节点的指针，是因为考虑到在进行后续的语义分析，转换工作时，经常需要联合多个语法模块的信息来综合考虑。结合编译原理的知识，可以解释为：分析继承属性时需要父节点和兄弟节点的信息，分析综合属性时需要子节点的信息。但是因为每个节点可能的子女数并不确定，所以我们很难却硬性规定每个节点该有多少子女指针。解决的方法就是只配备一根仅仅指向第一位子节点的指针，如果从父节点想要访问其中的某个子节点，仅仅需要先根据子节点指针找到第一子节点，然后根据子节点的兄弟指针寻找之后的子女，直到找到目标子节点。这个过程也可以包装成为一个 API 方便使用。这种设计方式虽然在寻找靠后的子节点时效率较低，但是极大的简化了数据结构，便于后续的程序设计，考虑到实际中的 Verilog 语法树每个节点的子节点数目并不会太多，所以时间的开销也不会过大。

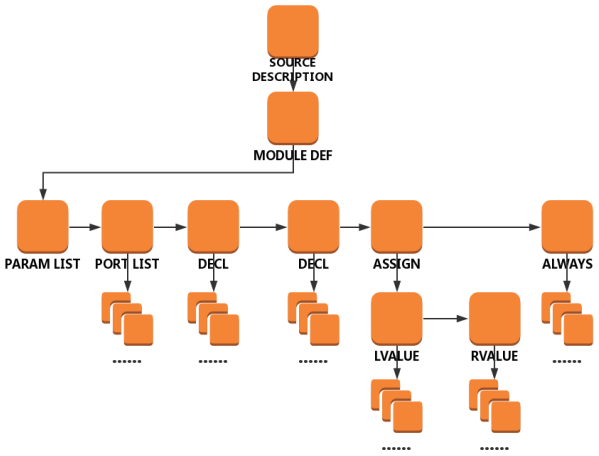

Figure 1 语法树结构示意图（省略了父指针）

上图是语法树的结构示意图（省略了父指针），可以看到根节点 SOURCE DESCRIPTION 节点直接指向自己的子女节点 MODULE DEF 节点，该节点又指向了第一个子女节点 PARAM LIST 节点，PARAM LIST 节点则指向了自己的后继兄弟节点。

3.3. 节点的设计

考虑到不同的语法模块所需的信息类型也不同，所以语法树节点 SyntaxNode 的设计必须满足多种类型的复用。采用的方法是将语法树节点进行分级，所有种类的节点都必须的信息直接存储在一级节点中，如果遇到特殊节点需要特殊信息，例如模块名，变量名，敏感列表类型等等，则将他们存放在二级节点中，一级节点有指针指向二级节点，这样就可以存储所有类型的语法信息了。

生成所有对应的语法节点并且将他们全部组织成语法树之后，为了方便之后的程序设计，这一部分又设计了一些常用的 API，例如获取一个表达式的操作数等。语法分析结束后可以选择打印语法树进行检查，语法树打印结果如下。

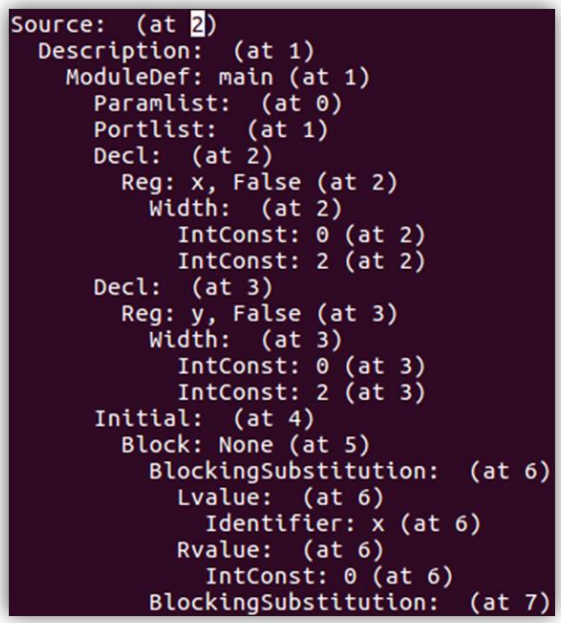

Figure 2 语法树打印结果

4. 状态机提取

4.1. 状态机数据结构

在建立抽象语法树后，需要基于抽象语法树构建Verilog代码的有穷状态机模型。在有穷状态机中，至少需要表达两个结构，state 和 condition，state 表示状态机的每个状态，condition 表示状态转移的条件。要表示每个 state 所对应的行为，并且要将 state 和 condition 通过指针连接起来。

condition 结构的基本信息包括：

a) be:该条件代表的基本表达式

b) st:该条件触发所达到的状态

state 结构包含的信息包括：

a) sn:该状态所代表的抽象语法树节点，是一个阻塞赋值或非阻塞赋值节点

b) be:该状态代表的基本表达式

c) cond:离开该状态的条件列表

d) id:该状态的标识。

4.2. 提取过程

4.2.1. 生成符号表

抽象语法树建立完成后，首先要建立所有变量的符号表，符号表中存储了变量名和变量的位宽。由于在生成condition、state 节点时需要生成该节点对应的基本表达式，而基本表达式需要利用变量的位宽，生成符号表至关重要。通过对抽象语法树的深度优先遍历将每个变量的信息插入符号表。

4.2.2. 提取所有 always 块

由于在产生一个信号时，所有产生该信号的条件都应放在一个 always 块内，所以每个变量的状态产生于一个 always 块内。首先要对抽象语法树进行深度优先遍历，找到所有 always 块的根节点。对每一个 always 语句块，找到该 always 块的所有变量并构建该变量的有穷状态机。

4.2.3. 建立有穷状态机

对每个变量，从该变量所在 always 语句块的根节点开始，进行深度优先遍历。在遍历过程中，对不同的节点类型进行不同处理。首先要处理敏感列表，对 posedge、negedge 等敏感列表条件，生成字符串插入全局敏感列表表中，为后续处理提供遍历。对 if、else、case 等节点，要建立 condition 节点，是通过该 condition 节点，到达了它指向的 state 状态节点，而这个状态节点就是 if、else 或 case 节点下执行的改变了该变量值的相关语句。对每个 condition 节点，将 if、else 或 case 语句的条件翻译为字符串存储到该节点对应的结构体中，翻译的过程是对条件表达式对应的抽象语法树节点进行递归处理。注意到 Verilog 代码中可能存在没有对应 else 节点的 if 节点，此时变量的值没有改变，但也属于一种可达状态，因此，要在有穷状态机中补充出这种可能的路径。对于改变了该变量值的阻塞赋值和非阻塞赋值节点，就要建立相应的 state 节点并链接到到达该状态的 condition 节点。对每个 state 节点，要记录该节点对应的阻塞或非阻塞赋值节点。这样，每个状态的内部信息以及状态间迁移的条件的过程就全部表示了。所有状态中，可以从初始状态迁移到达的都是可达状态，对模型的检验就是检验这些可达状态是否符合给定的规约。

5. NuSMV 代码生成

从 FSM 到 NuSMV 的转换涉及到许多细节处理。程序的整个处理流程可以类比于从源代码编译到目标代码，本环节将 FSM 中间层表示转换为 NuSMV 代码类比于汇编代码转换为机器码。

5.1. 基本表达式

NuSMV 支持绝大多数常见的操作符。操作数和操作符结合使用可产生基本表达式，基本表达式与次态表达式结合可产生复杂表达式。

处理基本表达式的转换首先要解决的问题是操作数的问题。Verilog 与 NuSMV 有两种截然不同的类型系统。Verilog 有 wire、reg、input、output 等，而 NuSMV 的 VAR 和 IVAR 两种变量声明类型可以表示 bool、array、word、integer。[11] 可见两者关联性很低。

NuSMV 对类型约束很严格。任何不同类型之间不允许隐式转换。integer 的值不能超出限定范围。我们尝试过 reg

变量用 bool 表示 0（False）和 1（True）的状态，一维数组用 integer 来表示范围。然而这两种类型不兼容，使用时必须使用类型转换符。且 integer 类型变量一旦超出范围不会自动进行模运算，这与寄存器级的逻辑表达不相符，且验证结果会因为死锁而总是为真。我们最终采用全部使用 word 来表达一个变量，而处理 input、output、reg 时采用方法有所不同。word 的麻烦之处在于两个操作数的类型和宽度必须匹配，若右边是整形常量，它就不符合类型系统，必须进行类型转换。

另一个转换是操作符的转换。Verilog 与 C 语言很相似，NuSMV 表示方式也很相似。但仍然存在不同之处。比如 NuSMV 没有逻辑与、逻辑或，只有按位与、按位或。这使得在分支语句中必须进行额外的检查。

完成相应工作后，将基本表达式语句的转换封装为一个函数，供多处使用。

5.2. 状态转移描述

状态的描述有初态和次态。初态使用 init 来约束，次态使用 next 来约束。如果不对变量的初态进行约束，则 NuSMV 验证时在第一状态直接给出规约的反例，即初始化一个不满足规约的状态集合，使得检测停止。如果不对次态进行约束，则次态可以在该变量的表示范围内任意变化，这个特性可以用来描述 Verilog 模块调用的 input 类型变量的参数，比如产生时钟。

对于初态的约束，若在 Verilog 源代码没有给出的明确的初始化值时，则初始化为 0，这与 Verilog 的硬件综合是相一致的。

对于次态的约束，依赖于 FSM。通过 FSM 结构取得相应的状态转移条件和转移执行动作。读这些条件和动作做相应分析，生成一个 case 语句，约束这个变量的次态变化。

5.3. 时序逻辑转换

时序逻辑在 Verilog 代码中主要表现在具有上升沿或下降沿的 always 语句块中。

上升沿和下降沿属于硬件特性，Verilog 是专门为硬件系统设计的描述语言，直接支持该特性。而 NuSMV 变量中没有一种是能描述这种性质的。这个问题在项目初期显得很难解决，主要原因在于我们使用 NuSMV 的 TRANS 语句进行转换。TRANS 是描述状态转移的语句之一，通过研究和测试，其对组合逻辑支持较好。而对于时序逻辑，如上升沿，很难加入到 TRANS 中。

我们设计了一种解决方式，将一个具有边沿触发属性的 Verilog 变量表示为 NuSMV 的两个变量，这两个变量的组合描述 Verilog 中该变量状态。举例说明，Verilog 的变量 a 被表示为 NuSMV 的当前状态变量 a 和上一状态变量 a#，a=0 且 a#=1 则可以表示下降沿。

为此，我们全面转移到 ASSIGN 语句描述的状态机，修改了前期的大量代码。但这个过程是值得的，ASSGIN 语句使得另外一些难以解决的问题有了解决方案。

5.4. 模块实例化

模块的概念在 Verilog 和 NuSMV 中很相似，均有实例名和模块名的调用，使得转换上更加自然。

NuSMV 有顶层模块的概念，相当于一个程序的入口。而 Verilog 描述的硬件，上电工作，代码可能存在一个顶层模块（只调用其他模块且没有被其他模块调用的属于顶层模块）。为了使得“程序入口”更加清晰，我们需要用户在规约中提供顶层模块名。这个顶层模块被一个生成的 main 模块实例化，main 模块中还会做其他的初始化工作。

另一个问题是 Verilog 参数可以乱序，而 NuSMV 不允许这样做，解决方式是实例化时对参数重新排序。

6. 工具的实现与检验

6.1. 整体框架

整个项目的整体框架如 Figure 3 所示，程序的输入包括需要检查的 Verilog 源代码以及用户提供的规约 SPEC。Verilog 源代码经过 PyVerilog 程序处理得到相应的 AST,然后经过 NuSMV Generator 程序生成 NuSMV 合法的状态机。与输入文件 SPEC 一起作为 NuSMV 的输入进行模型检测，得到最终的输出结果，并且产生反馈，对应到 Verilog 源代码上，提供可能出现错误区域的提示。

在此基础上，我们编写的图形化界面能够智能的将这些功能集成在一起。图形化界面支持用户从界面内写入修改或者从本地文件打开。通过这种方式获取项目的输入，然后利用系统调用的方法依次进行上述的处理过程，得到输出的结果与反馈。更新在图形化的界面上，使得用户能够更加清晰直观的得到模型检测的结果。

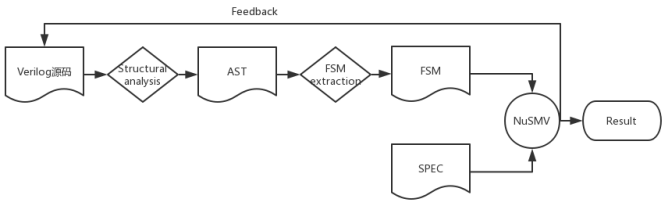

Figure 3 整体框架

6.2. 工具实现

上文对本项目的理论、实现技术大体框架等进行了详细说明。本节对我们实现的工具的具体效果进行展示。

代码的实现更加需要在细节上加以注意，如 Verilog 输

入后应该有预处理的过程，使得语法分析过程不需要考虑如宏定义头文件包含等等问题。程序运行时应该检测运行时环境，避免多个进程间通讯时共享资源分配问题。

Figure 4 是验证一个 Verilog 代码是否满足规约的图形化界面。最左边是输入 Verilog 代码的区域，中上方是输入用户规约的地方，中下方是验证规程，如果执行成功为绿色，否则为红色，出错的原因可能是 Verilog 语法错误、规约语法错误等等。最右边的框显示了验证的属性是否成立，成立为绿色，不成立及其反例路径为红色。

Figure 4 图形化界面

利用 FSM，我们可以对 Verilog 代码生成其状态转移图。Figure 5 是一个状态转移图的例子。每个变量都具有一个触发器，触发它进行状态迁移，状态迁移的条件为一条边，若边上没有信息则表明触发器信号使得该变量无条件执行相应动作。第三层及以下的每个节点表示执行动作。

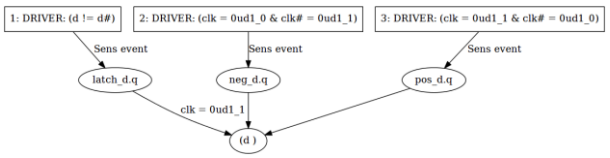

Figure 5

Figure 6 是反向路径追踪，根据反例在 FSM 中找出不满足条件的状态，定位在 Verilog 代码中，为用户给出提示，表示可能不满足规约的语句范围。

```
14          always @(posedge clk)
15          begin
16            if (cnt == 4 && yellow == 1)
17            begin
18              cnt <= 0;
19              yellow <= 0;
20              red <= 1;
21            end
22            else if (cnt == 30 && red == 1)
```

Figure 6

6.3. 基于案例的工作流程具体分析

本案例的 Verilog 代码见附录 A，规约见附录 B。

这段 Verilog 代码描述了一个交通灯系统，有红、绿、黄三种颜色交替变化。本处验证其交替变化的几种条件是否满足。代码第 16 行存在一个错误，黄灯的转移条件应为计数器为 3 的状态。

首先，对 Verilog 代码进行预处理，本文件没有宏定义和头文件包含。预处理输出结果。

对文件进行语法分析产生语法树并输出结果，如下：

Source: (at 1)

Description: (at 1)

ModuleDef: traffic_lights (at 1)

Paramlist: (at 0)

Portlist: (at 1)

Decl: (at 2)

Input: clk, False (at 2)

……

分析代码中的符号，产生符号表，以下为 NuSMV 的表示形式：

clk : unsigned word[1];

cnt : unsigned word[5];

green : unsigned word[1];

……

根据语法树生成状态机，下面为初始节点和一个状态：

traffic_lights.cnt: DRIVER: (clk = 0ud1_1 & clk# = 0ud1_0)

COND: (((cnt = (0ud5_4))& (yellow = (0ud1_1)))), BEHAVIOUR: (((0ud5_0)))

对规约文件进行预处理，提取相关信息等。

将 FSM 转换为 Graphviz 的输入文件，使用 dot 工具生成状态转移图。

根据 NuSMV 语法规则，将 FSM 生成 NuSMV 输入文件。合成输入文件和规约。

使用 NuSMV 进行验证，并得到验证结果。

分析验证结果，给出相应提示。

根据提示可以很容易找到相应错误，修改之后再次验证，此时规约全部通过。

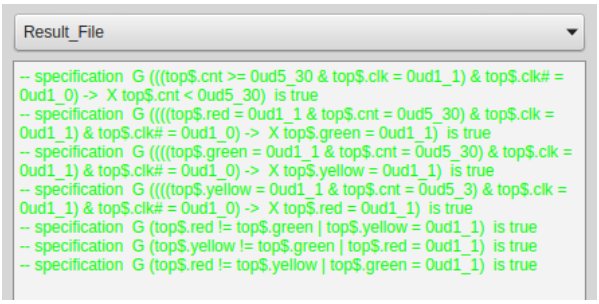

Figure 7 修改后规约全部通过

7. 总结

在本次项目实践中，我们对于模型化检验以及 Verilog 程序语言都有了更加深刻的了解，同时设计了一个能够输入为 Verilog 程序和对应规约，输出为以程序对应的状态机，NuSMV 代码，对规约检查的结果以及在原代码上给出的修改建议。转换程序实现了对本科“数字电路设计实验”课程涵盖的 Verilog 语法的大部分内容，并且以实验课程中学生自己编写的代码作为测试样例进行测试，在给出错误检查结果同时也能给出修改意见，极大的简化了在编写 Verilog 程序时调试错误的成本和代价，节约了大量的调试时间。在之后的后续推广过程中，我们会进一步增加转换程序所能够涵盖的 Verilog 程序的语法范围，进一步提高错误检查的效率以及错误修改建议的精度，并且在项目进一步成熟之后在本科生之中进行推广，希望能够最终在“数字电路设计实验”课程中发挥作用，作为辅助教学的工具，实质性的为本科教学创新和进步贡献力量。

致　谢　特别感谢项目指导老师——卜磊副教授自始至终对我们所给予的支持和鼓励！

参 考 文 献

[1] 赵丽芳. 基于 UPPAAL 和 UML 的实时系统形式化分析与应用[D]. 苏州大学,2008

[2] NuSMV: A New Symbolic Model Verifier A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri http://nusmv.fbk.eu/

[3] Automatic verification of knowledge and time with NuSMV A Lomuscio, C Pecheur, F Raimondi

[4] Model checking time Petri nets using NuSMV, A Bobbio, A Horvath

[5] Modeling freshness concept to overcome replay attack in kerberos protocol using nusmv S Adyanthaya, S Rukmangada

[6] 龙菲, 赵一帆. 基于FPGA的逻辑分析仪[J]. 硅谷, 2014(8):197-198.

[7] Spin - Formal Verification: http://spinroot.com/spin/whatispin.html

[8] 张军林. NuSMV模型验证器实现分析. 中山大学硕士学位论文. 2010-06-02

[9] Aho, Sethi, Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986. ISBN 0-201-10088-6

[10] Shinya Takamaeda-Yamazaki: Pyverilog: A Python-based Hardware Design Processing Toolkit for Verilog HDL, 11th International Symposium on Applied Reconfigurable Computing (ARC 2015) (Poster), Lecture Notes in Computer Science, Vol.9040/2015, pp.451-460, April 2015.

[11] Roberto Cavada, Alessandro Cimatti,. Charles Arthur Jochim, NuSMV 2.6 User Manual. https://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf

附录A

```
1   module traffic_lights();
2       input clk;
3       reg red;
4       reg green;
5       reg yellow;
6       reg [0:4]cnt;
7       initial
8       begin
9           red = 1;
10          green = 0;
11          yellow = 0;
12          cnt = 0;
13      end
14      always @(posedge clk)
15      begin
16          if (cnt == 4 && yellow == 1)
17          begin
18              cnt <= 0;
19              yellow <= 0;
20              red <= 1;
21          end
22          else if (cnt == 30 && red == 1)
23          begin
24              cnt <= 0;
25              green <= 1;
26              red <= 0;
27          end
28          else if (cnt == 30 && green == 1)
29          begin
30              cnt <= 0;
31              green <= 0;
32              yellow <= 1;
33          end
34          else
35          begin
36              cnt <= cnt + 1;
37          end
38      end
39  endmodule
```

附录B

```
--TOP traffic_lights
LTLSPEC
    G((top$.cnt >= 0ud5_30 & top$.clk = 0ud1_1 & top$.clk#
        = 0ud1_0) -> X(top$.cnt < 0ud5_30))
LTLSPEC
    G((top$.red = 0ud1_1 & top$.cnt = 0ud5_30 & top$.clk =
        0ud1_1 & top$.clk# = 0ud1_0) -> X(top$.green =
        0ud1_1))
LTLSPEC
    G((top$.green = 0ud1_1 & top$.cnt = 0ud5_30 & top$.clk
        = 0ud1_1 & top$.clk# = 0ud1_0) -> X(top$.yellow =
        0ud1_1))
LTLSPEC
    G((top$.yellow = 0ud1_1 & top$.cnt = 0ud5_3 & top$.clk
        = 0ud1_1 & top$.clk# = 0ud1_0) -> X(top$.red =
        0ud1_1))
LTLSPEC
    G(top$.red != top$.green | top$.yellow = 0ud1_1)
LTLSPEC
    G(top$.yellow != top$.green | top$.red = 0ud1_1)
LTLSPEC
    G(top$.red != top$.yellow | top$.green = 0ud1_1)
```