一种基于模型检验程序分析技术的前端工具研究

叶俊民 谢 茜 金 聪 李 明 张振方
(华中师范大学计算机科学系 武汉 430079)

摘 要 提出了一种用模型检验技术对程序进行分析的方法,其主要思想是将C/C++源代码转换为与控制流图等价的Kripke结构,用CTL公式描述待验证的源程序性质,使用NuSMV模型检验工具实施具体的程序分析。基于这一思想,设计并实现了一个自动将C/C++源代码转换为NuSMV输入的工具。所做的实验验证表明,该方法能够有效地对程序进行分析。

关键词 模型检验,程序分析,自动NuSMV输入工具

中图法分类号 TP311 **文献标识码** A

Research on a Front-end Tool for Program Analysis Based on Model Checking

YE Jun-min XEI Qian JIN Cong LI Ming ZHANG Zhen-fang
(Department of Computer Science, Central China Normal University, Wuhan 430079, China)

Abstract A model checking-based program analysis method was proposed. The main steps include translating C/C++ source code into Kripke structure that is equivalent to control flow graph, describing properties of source code in CTL formula and verifying the program using model checker NuSMV. Based on this idea, a tool which is used to automatically translate C/C++ source code into NuSMV input was designed and implemented. The experiments show that this approach is able to analyze the program effectively.

Keywords Model checking, Program analysis, Automatic NuSMV input tool

1 引言

目前,已经有学者提出了将模型检验技术应用到程序分析中以发现程序故障的思想[1,2]。我们在此基础上研究了基于模型检验思想的程序分析技术[5],其核心思想是结合NuSMV工具,通过遍历待检测系统模型的状态空间来测试该模型是否满足给定的性质。这一过程涉及对待检验系统进行形式化建模,使用时序逻辑公式描述期望的性质。本文将在文献[5]的基础上,研究基于模型检验程序分析技术的一种前端输入工具的实现。

2 研究基础

在标记图、标记树、性质树、抽象语法树(AST)、控制流程图、Kripke结构及其流程图和查询语言等概念[5]的基础上提出的基于模型检验技术进行程序分析的原理是,首先用GCC分析源代码生成的AST中间文件,并将该中间文件转化为一种基于XML表示的AST;从XML形式的AST中导出CFG;使用标记函数对AST和CFG进行标记,得到与控制流图等价的Kripke结构。其次,用CTL公式描述程序应该满足的性质。最后,将有限状态模型和CTL公式描述的程序性质转化为NuSMV的输入并实现对程序的分析。

下面描述具体步骤。

2.1 将C/C++源代码转换成控制流图的等价Kripke结构

2.1.1 从源代码生成具有XML结构的AST

(1)分析GCC生成的中间形式的AST

在GCC(v4.3.2)中,使用编译参数-fdump-original-original(g++),可以生成后缀名为original的抽象语法树AST中间文件。首先分析original文件格式,得到如图1所示的AST中间文件结点实例。

```
@1 function_decl  name:@2  mngl:@3  type:@4
                  srcp:sample.c:4  args:@5
                  extern  body:@6
```

图1 GCC的AST中间文件结点

(2)定义用于AST表示的XML框架

为了将中间形式的AST转换为XML格式,定义了表示AST的XML schema,具体如图2所示。

```
<xs:element name="node">
  <xs:complexType>
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="isInCfg" type="xs:bool" use="required"/>
  <xs:sequence>
    <xs:element name="type" type="xs:string"
```

到稿日期:2009-06-28 返修日期:2009-09-27 本文受武汉大学计算机软件工程国家重点实验室开放基金项目(编号:SKLSE20080705),湖北省自然科学基金(编号:2008CDB349),华中师范大学校基金(编号:2009A12),华中师范大学中央高校基本科研业务费项目(编号:CCNU09Y01009和CCNU09Y01013)资助。

叶俊民(1965-),男,博士,教授,主要研究兴趣是软件工程、高可信软件故障诊断,E-mail:jmye@mail.ccnu.edu.cn。

```
            minOccurs="0" maxOccurs="1" />
        <xs:element name="label" type="xs:string"
            minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="attr" type="xs:attr"
            minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="edge">
<xs :complexType>
    <xs:attribute name="id" type="xs:ID" use="optional"/>
    <xs : attribute name = "from" type = "xs: IDREF" use = "required" />
    <xs:attribute name="to" type="xs:IDREF" use="required" />
    <xs :attribute name="fromorder" type="xs:string" use="optional" />
    <xs :attribute name="isdirected" type="xs:bool" use="optional" />
    <xs :attribute name = "isInCfg" type = "xs: bool" use = "required"/>
    <xs :sequence>
        <xs:element name="type" type="xs:string"
                    minOccurs="0" maxOccurs="1" />
        <xs:element name="attr" type="xs:attr"
                    minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs :element name="attr"> <xs:complexType name="xs:attr">
    <xs:attribute name="id" type="xs:IDREF" use="optional" />
    <xs :attribute name = "name" type = "xs: NMTOKEN" use = "requried" />
    <xs :attribute name="kind" type="xs:NMTOKEN" use="optional" />
    <xs :sequence>
        <xs:element name="type" type="xs:string"
                    minOccurs="0" maxOccurs="1" />
        <xs:element name="attr" type="xs:attr"
                    minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
</xs:element>
```

图 2　AST 的 XML Schema

(3)生成 XML 形式的 AST

现在定义生成 XML 形式的 AST 转换规则如下：

Rule 1　如果是记录索引值，则每遇到一条新记录就建立一个新的 node 元素与之对应。其中，node 元素的 id 属性记录索引值，node 元素中的 isCFG 属性记录此结点是否是 CFG 结点，type 子元素纪录此结点的语句类型，label 子元素记录在此结点上成立的谓词。

Rule 2　如果记录与其他记录有程序分析相关的项，则建立 edge 元素。其中，edge 元素中的 from 性质记录边的起始结点索引，to 性质记录目的结点的索引。

Rule 3　如果记录中存在与其他记录不相关的项，则这些项由 attr 元素记录，作为 node 的子元素。

根据规则，基于 XML 形式的 AST 生成算法如图 3 所示。

```
算法功能：生成基于 XML 形式的 AST
输入：源代码
输出：基于自定义的 XML 结构的 AST
Begin
调用 GCC 处理源代码文件，生成 GCC 的 AST 中间文件；
for GCC 的 AST 中间文件的每一项记录 do
Begin
  1)如果是索引值，按照 Rule1 进行处理；
  2)如果是记录与其它记录相关的项，按照 Rule2 进行处理；
  3)如果是记录中与其它记录不相关的项，按照 Rule2 进行处理；
End
保存处理后的内容至文件；
End
```

图 3　源代码到抽象语法树的生成算法

因为该生成算法针对 GCC 的 AST 中间文件中的记录，所以其算法复杂度只与记录的个数有关。设 GCC 的 AST 中间文件的记录个数有 N 个，则此算法的时间复杂度为 $O(N)$。

2.1.2　通过 XML 形式的 AST 生成 CFG

控制流程图存储的是从 XML 形式的 AST 中抽出的与控制流和数据流相关的结点和边。因此，从 XML 形式的 AST 到控制流程图的转换算法可以简化为两步：(1)识别与控制流和数据流相关的结点；(2)标记被识别的结点为 CFG 结点。相关算法如图 4 所示。

```
算法功能：生成 CFG
输入：基于 XML 的 AST
输出：标识所有 CFG 结点
Begin
    初始化 CFG 结点性质的集合 P；
    for AST 中的每一个结点 node do
        Begin
          if node 的结点类型 ∈ P then 设置 node 的 isInCfg 属性为"true"
      end
End
```

图 4　CFG 的生成算法

因为生成 CFG 文件的算法操作针对的是 AST 中的结点，所以其算法复杂度只与 AST 中结点的个数有关。设 AST 有 N 个结点，那么此算法的时间复杂度为 $O(N)$。

2.1.3　获得 CFG 的等价 Kripke 结构

通过 2.1.1 节和 2.1.2 节得到的结点集合 L 和结点间的迁移关系集合 E，可得到满足 Kripke 结构的控制流程图的另外 3 个元素，即原子命题集合 AP、在结点上标记表达式类型的标记函数 μL 和在结点上标记是否满足的原子命题集合的标记函数 μE。

为了表述待分析的程序性质，定义如表 1 所列的原子命题。

表 1　原子命题

原子命题	含义
Declx	声明变量 x
assignedx	变量 x 赋值
Usedx	变量 x 的使用

mallocx	指针 x 分配内存
Freex	指针 x 释放内存
referencedx	声明引用 x
dereferencedx	对指针 x 解引用
End	程序结束

标记函数 μL 可将识别如图 1 所示的结点类型的工作与生成 AST 结点的工作同时进行。标记函数 μE 标记是否满足原子命题集合有两步：(1)对每一个原子命题，查询所有满足此原子命题的结点；(2)在查询到的结点上标记此原子命题。

本文将用到的所有满足原子命题结点的查询表达式的定义如表 2 所列。

表 2　查询表达式

命题	查询表达式
Declx	↓ * Var_decl[x]
Assignedx	↓ * Modif[Var_decl[x][attr="lhs"]] ∪ Init[Var_decl[x][attr="lhs"]]
Usedx	↓ *(Plus ∪ Minus ∪ Mult ∪ Postinc ∪ Preinc ∪ Postdec ∪ Predec ∪ …)[Var_decl[x]] ∪ ↓ * Modify[Var_decl[x][attr="rhs"]]
Mallocx	↓ * Modify[Var_decl[x][attr="lhs"] (function[malloc][attr="rhs"]]
Frreex	↓ * Call[Var_decl[x][attr="lhs"] ∪ function[free][attr="rhs"]]
Referencedx	↓ * addr[Var_decl[x]]
Dereferencedx	↓ * Indirect[Var_decl[x]]

使用标记函数对 CFG 进行标记，然后将其转化为等价的 Kripke 结构的算法，如图 5 所示。

```
算法功能：将 CFG 转化为等价 Kripke 结构
输入：抽象语法树 AST，控制流程图 CFG，原子命题集合 AP
输出：标记了原子命题的 CFG，即 CFG 的等价 Kripke 结构
Begin
  for 每一个原子命题集合中分原子命题 p do
    begin
      while(AST 中结点没有处理完) do
      查询满足 p 的结点 node;
      if(node (CFG) then 在 node 上标记 p;
      else
      begin
        AST 上回溯 node 的祖先 nodeP，直到 nodeP∈CFG;
        在 nodeP 上标记 p;
      end
    end while
end
end
```

图 5　将 CFG 转化为等价 Kripke 结构的算法

2.2　用 CTL 公式描述待检测的程序性质

本文定义的待分析的程序性质包括：(1)不存在未初始化变量；(2)内存分配和释放配对；(3)不存在野指针；(4)不存在变量未使用。待分析的程序性质如表 3 所列。

表 3　所有 CTL 公式描述的性质

性质	CTL 公式
不存在未初始化变量	AG(declx→A(→usedxUassignedx))
内存分配和释放配对	AG(mallocx→AF freex)
不存在变量未使用	AG(declx→AF usedx)
不存在野指针	AG(freex→AF→dereferencedx)

3　NuSMV 前端输入工具的实现

为了使用 NuSMV 工具，本文采用文献[4]中设计的从程序模型和待分析的程序性质到 NuSMV 工具输入的转换规则。NuSMV 工具的输入称为 NuSMV 输入，具体如下：

给定一个 C/C++程序 f，本文将相应的 Kripke 结构 $CFG(Lf,Ef,APf,\mu f)$ 转换成一个 NuSMV 模型，$NuSMV\ f=(varf,\Delta f,Deff,CTLf)$。其中，在 NuSMV 中，$varf$ 是一个 L 上的枚举变量，$l\in Lf$。$\Delta f=\{(l,Succ(l))\mid l\in Lf,\ Succ(l)=\{l'\mid(l,l')\in Ef\}\ l\}$；$Deff=\{define(p)=\{l\mid\mu f(l)=p,l\in Lf\}\mid p\in\Sigma f\}$；CTLf 是 CTL 性质符合 NuSMV 语法的语句集合。

Rule 4　若 $li\in Lf$，假设 li 的结点 id 是 i，且 $1\leqslant i\leqslant n$，则在 NuSMV 程序中定义 $varf$ 为一个 NuSMV 的枚举变量。该枚举变量的名字为 $location$，其取值范围是{$loc1,loc2,loc3,\cdots,locn$}。

Rule 5　若$(l_i,l_{m1}),(l_i,l_{m2}),\cdots,(l_i,l_{mj})\subset E_f$，则转换为 NuSMV case 语句中的一部分，用 $location=loci:\{locm1,locm2,\cdots,locmi\}$表示结点之间的迁移关系，其含义为从 $loci$ 到 $locm1,locm2,\cdots,locmi$ 存在迁移关系。

Rule 6　AP 中原子命题 p，在 CFG 中标记了此命题的位置 $l_{i1},l_{i2},\cdots,l_{in}$，define 区域写入 $p:=location\{loci.loci_2,\cdots,loc_{in}\}$，该式子的含义是对于原子命题 p，存在 $loci_1,loci_2,\cdots,loc_{in}$ 位置满足该原子命题。

Rule 7　将 CTL 公式中的符号转换为 NuSMV 公式中的对应符号。

为了实现上述相关过程，其核心就是将 C/C++源代码自动转换为 NuSMV 输入。下面将论述这一过程。

3.1　NuSMV 前端输入工具 F-N 的结构

图 6 为设计的 F-N 工具结构图。其中从 Input 层到接口层已由文献[5]实现；从接口层到 Output 层为本工具的主要实现目标。其中，枚举变量提取模块实现 CFG 中的 Lf 转换成 NuSMV 模型中的 $varf$ 的过程；迁移关系提取模块实现 CFG 中的 Ef 转换成 NuSMV 模型中的 Δf 的过程；原子命题提取模块确定待检测的程序性质的原子命题 p 在 CFG 中标记的位置，并将其转换为 NuSMV 公式中的对应符号。

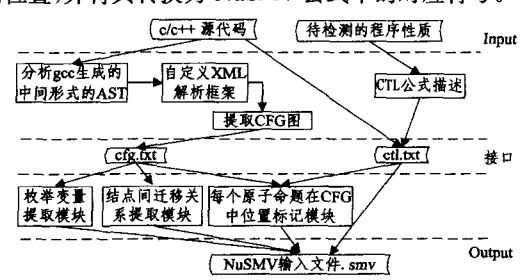

图 6　F-N 工具结构图

3.2　工具所用数据结构设计

F-N 工具定义了描述 Kripke 结构的 CFG 图，Node 类的对象结点的数据结构如下：

```
public class Node {
        private String no;
        private List(String) info;
        private String next;
}
```

上述数据结构表示中，no 用于唯一标识 Kripke 结构 CFG 中的 Lf。若 $li\in Lf$，li 的结点 id 是 i，则该结点的 no 在数值上等于 i。info 用于标识对应的 li 中包含的原子命题

p，且 $p \in APf$。next 用于标识与结点 li 存在迁移关系的结点。若 $li, lj \in Lf$，li 的结点 id 是 i；lj 的结点 id 是 j，如果 li, lj 存在迁移，相对 li 结点的 Node 中的 next 在数值上等于 j。此外，设计时用到的其他数据结构包括用于记录 NuSMV 输入的枚举变量集合 $location(n)$、用于记录结点迁移关系集合 $case(n)$、用于记录 ctl. txt 文件中所有原子命题内容的集合 $ctl(n)$以及标记每个原子命题 p 在 CFG 图中的位置的集合 $define(n)$。

3.3 主要算法

NuSMV 前端输入实现算法如图 7 所示。

```
算法功能：实现 NuSMV 前端的输入
输入：cfg. txt 文件和 ctl. txt 文件
输出：NuSMV 工具的前端输入文件
Begin
Step 1  初始化 NuSMV 的 location(n)，case(n)，ctl(n)和 define(n)集
        合为空；
Step 2  扫描 cfg. txt 文件
  for 对 cfg. txt 文件中每一个 Node 结点 do
  begin
    if(Node. no 不存在于 location 中) then 将此 Node. no 加入到 lo-
      cation 中；
  end
Step 3  标识 location 中每个 Node 结点的迁移关系
  for 对 location 中标识过的每个 Node 结点 loc(i)do
  begin
  while(Node. no =i) do 获取 i 结点的 next；
      if(Node. next ! =null)
      then 将此 Node. next 加入到该结点的 case(i)中；
      else 该结点的 case(i)=1；//表示结束
  end
Step 4  扫描 ctl. txt 文件，通过正则表达式获取所有变量的原子命
        题，将其存入到集合 ctl(n)中；
Step 5  标识每个原子命题在 CFG 中的位置；
  for 集合 ctl(n)中每个原子命题 ctl(i) do
  begin
      for 每一个 Node 结点 do
      if(Node. info 中包含了原子命题 ctl(i))
      then 将 Node. no 加入到该原子命题对应的 define(i)集合中；
      else 该结点的 define(i)=0；//表示该原子命题没出现；
end
End
```

图 7　NuSMV 前端输入实现算法

本生成算法是针对 cfg. txt 文件和 ctl. txt 文件进行的操作，其算法复杂度与 cfg. txt 文件的中记录的条数和 ctl. txt 文件中原子命题的个数有关。本文 cfg. txt 文件中记录的条数在数值上等于 CFG 图中的 Ef，即边的个数。假设记录的条数为 N，原子命题的个数为 M，那么此算法的时间复杂度为 $O(N(M+2))$。

4　实例分析

本文使用的例子程序如图 8 所示。

```
1 #include <stdio. h>
2 #include <stdlib. h>
3 #include <string. h>
4 int main()
5 {
6   char * a;
7   char * b;
8
9   a=(char * )malloc(20 * sizeof(char));
10  b=(char * )malloc(20 * sizeof(char));
11  strcpy(a,"Hello,world. \n");
12  printf("%s",a);
13  free(a);
14  printf("%c", * a);
15 }
```

图 8　实验用程序 sample. c

由 GCC 生成的 AST 中间文件如图 9 所示。

```
;; Function int main() (main)
;; enabled by -tree-original
@1   statement_list  0:@2          1:@3
@2   bind_expr      type:@4       vars:@5     body:@6
@3   return_expr    type:@4       expr:@7
@4   void_type      name:@8       algn:8
@5   var_decl       name:@9       type:@10    scpe:@11
                    srcp:sample. c:6          size:@12
                    algn:32       used:1
......
@23  integer_type   name:@35      size:@36    algn:8
                    prec:8        sign:signed min:@37
                    max:@38
```

图 9　部分 GCC 的 AST 中间文件

依据图 1 所生成的基于 XML 的 AST 部分结果如图 10 所示。

```
<? xml version="1. 0"?>
<ast>

<node id="1" isInCFG="false">
      <type name="statement_list"/>
</node>
<edge from="1" to="2" isInCFG="false"><type name= "state-
ment"/></edge>
<edge from="1" to="3" isInCFG="false"><type name= "state-
ment"/></edge>
.....
<node id="6" isInCFG="false">
      <type name="statement_list"/>
</node>
<edge from="6" to="13" isInCFG="false"><type name= "state-
ment"/></edge>
.....
<edge from="6" to="20" isInCFG="false"><type name= "state-
ment"/></edge>
.....
</node>
```

图 10　基于 XML 的 AST 部分结果

有了基于 XML 的 AST，就可以从该 AST 中识别出 CFG 结点，可视结果如图 11 所示。

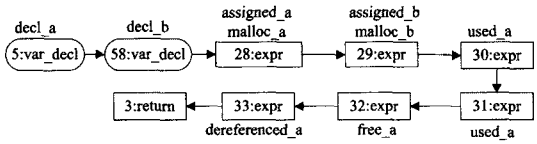

图 11　识别出的 CFG 结点

依据第 3 节介绍的将 Kripke 结构 CFG 图转换为文本格式的 cfg. txt 文件中的内容，结合定义的 CFG 图中结点的数据结构表示，将图 9 中的 Kripke 结构 CFG 图转换为图 12。

init state:5

no	info	next
05	decl_a	58
58	decl_b	28
28	assigned_a,malloc_a	29
29	assigned_b,malloc_b	30
30	used_a	31
31	used_a	32
32	free_a	33
33	indirect_a	3
3		

图 12　转换后的 cfg. txt 文件

结合 C/C++源代码，扫描出源代码中所有出现的变量，将其定义为待分析的程序性质，如表 4 所列。

表 4　CTL 公式描述的性质

性质	CTL 公式
不存在未初始化变量	$AG(decl_a \rightarrow A(\neg used_a\ U\ assigned_a))$
	$AG(decl_a \rightarrow A(\neg used_b\ U\ assigned_b))$
内存分配和释放配对	$AG(malloc_a \rightarrow AF\ free_a)$
	$AG(malloc_b \rightarrow AF\ free_b)$
不存在变量未使用	$AG(decl_a \rightarrow AF\ used_a)$
	$AG(decl_b \rightarrow AF\ used_b)$
不存在野指针	$AG(free_a \rightarrow \neg AF\ dereferenced_a)$
	$AG(free_b \rightarrow \neg AF\ dereferenced_b)$

将表 4 中第 2 列的 CTL 公式单独提出，生成文本格式的 ctl. txt 文件。

依照第 3 节定义的转换规则，结合 NuSMV 输入实现算法，设计了 NuSMV 前端输入工具，如图 13 所示。

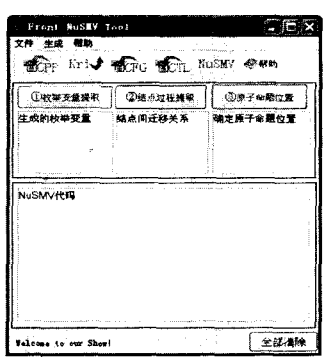

图 13　NuSMV 前端输入工具

分别运行 3 个提取功能过程后的结果如图 14 所示。

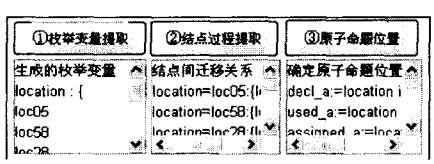

图 14　提取功能运行结果

运行“生成 NuSMV 文件”或点击“NuSMV”按钮后，可以将以上 3 个过程提取的内容合并成 NuSMV 输入，运行后结果如图 15 和图 16 所示。

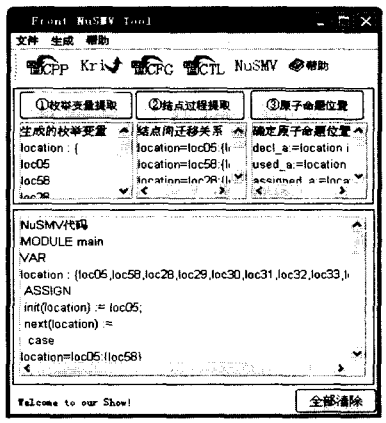

图 15　合并运行结果

```
MODULE main
 VAR
  location:{loc5,loc58,loc28,loc29,loc30,loc31,loc32,loc33,loc3};
 ASSIGN
init(location):=loc5;
  next(location):=
   case
    location=loc5:{loc58};
    ......
    location=loc33:{loc3};
    1       :location;
   esac;
 DEFINE
  decl_a:= location in {loc5};
  used_a:= location in {loc30,loc31};
  malloc_a:= location in {loc28};
  free_a:= location in {loc32};
  indirect_a:= location in {loc33};
  ......
SPEC AG (decl_a-> A [! used_a U assigned_a])
SPEC AG (decl_b-> A[! used_b U assigned_b])
......
```

图 16　转换后的 NuSMV 前端输入

运行上面的 NuSMV 程序得到的程序性质分析结论如图 17 所示。实验表明这个程序有 3 个错误：1)内存分配和释放不配对；2)存在变量未使用；3)存在使用野指针。

```
-specification AG (decl_a->A[! used_aUassigned_a]) is true
-specification AG (decl_b->A[! used_bUassigned_b]) is true
-specification AG (malloc_a->AF free_a) is true
-specification AG (malloc_b->AF free_b) is false
-specification AG (decl_a->AF used_a) is true
-specification AG (decl_b->AF used_b) is false
-specification AG (free_a->! (AF indirect_a)) is false
-specification AG (free_b->! (AF indirect_b)) is true
```

图 17　NuSMV 检测例子程序的所有性质的结果

结束语　本文介绍了基于模型检验进行软件分析的方法，并将其应用于实际程序，验证了方法的正确性。本方法的输

（下转第 174 页）

说明了负反馈使得系统接受解在一定程度上的退化；平直线说明在相应次数的迭代中没有更好的迭代最优解出现。

对比图 5 和图 6 可知，参数 q_0 越大，解的收敛曲线下降越陡，解的收敛速度越快，但所得最优解的质量下降。这是由于 q_0 是控制全局蚁概率搜索深度的参数，q_0 越大全局蚁对先前搜索经验的利用越多，越倾向于在同一区域附近搜索，造成解的质量下降。

6.4 实验三：至今最优 Ant-Agent

至今最优蚁是迭代最优的高一层主体，借助它可以分析整个系统的涌现及演化过程。图 7－图 10 表示在参数 $q_0=0.6$，$\rho=0.1, 0.2, 0.25, 0.3$ 时，至今最优解在 200 次迭代中的变化情况。图中曲线的严格递减说明整个系统有序地向着求出最优解的方向进化。单个主体能力有限，当它们协同工作时却涌现出了宏观有序的全局寻优智能行为，即群集智能。

对比图 7－图 10 可知，当 $\rho=0.25$ 时系统的演化最佳。在 $\rho=0.1, 0.2$ 时，由于每次相变(phase transition)前后都有较长的稳定期(stability)，因此系统要花较长时间才能到达最优状态。在 $\rho=0.3$ 时，在迭代前期，各稳定期较短，系统演化较快；但在从次优解 148 到最优解 141 的相变前后，出现了较长的稳定期，致使整个系统演化变缓。

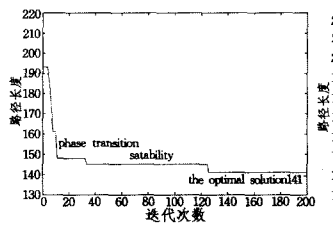

图 7 $\rho=0.1$ 时至今最优解的变化情况

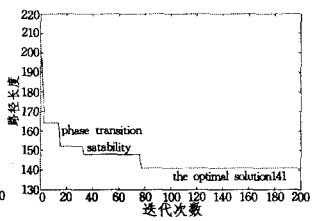

图 8 $\rho=0.2$ 时至今最优解的变化情况

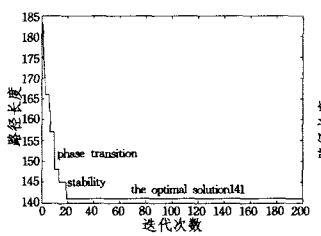

图 9 $\rho=0.25$ 时至今最优解的变化情况

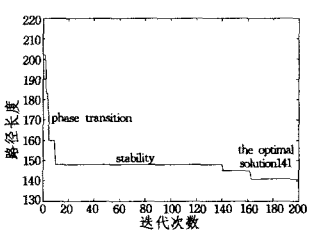

图 10 $\rho=0.3$ 时至今最优解的变化情况

结束语 仿真试验表明，本文研究的基于 ACS 的路径规划方法其速度快，具备了积极、实时的重要特性，十分适合移动机器人的路径规划。在 CAS 理论指导下，人工蚁群及其所处环境是一个没有全局控制的、基于个体的、不断发展和演化的系统；“整体大于部分之和”的增值是系统演化的主要动因。生成全局次优解和进一步加快系统演化速度是本文今后需要改进的地方。

参 考 文 献

[1] Dorigo M, Maniezzo V, Colorni A. The Ant System: optimization by a colony of cooperating agents[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 1996, 26(1): 1-13

[2] Dorigo M, Gambardella L M. Ant colony system: A cooperative learning approach to the traveling salesman problem[J]. IEEE Transactions on Evolutional Computation, 1997, 1(1): 53-66

[3] 席裕庚，张纯刚. 一类动态不确定环境下机器人的滚动路径规划[J]. 自动化学报，2002，28(2)：161- 175

[4] 蔡自兴，贺汉根，陈虹. 未知环境中移动机器人导航控制研究的若干问题[J]. 控制与决策，2002，17 (4)：385-390，464

[5] 张捍东，郑睿，岑豫皖. 移动机器人路径规划技术的现状与展望[J]. 系统仿真学报，2005，17(2)：439- 443

[6] 廖守亿，戴金海. 复杂适应系统及基于 Agent 的建模与仿真方法[J]. 系统仿真学报，2004，16(1)：114-116

[7] 王正中. 基于演化的复杂系统建模与仿真研究[J]. 系统仿真学报，2003，15(7)：907

[8] 朱庆保，张玉兰. 基于栅格法的机器人路径规划蚁群算法[J]. 机器人，2005，27(2)：132-136

[9] 樊晓平，罗熊. 复杂环境下基于蚁群优化算法的机器人路径规划[J]. 控制与决策，2004，19(2)：166- 169

[10] 董玉成，陈义华. 基于蚂蚁算法的移动机器人路径规划[J]. 重庆大学学报，2003，26(3)：49-51

[11] 叶小勇，雷勇，侯海军. 蚁群算法在全局最优路径寻优中的应用[J]. 系统仿真学报，2007，19(24)：5643-5647

[12] Tan G, He H. Global optimal path planning for mobile robot based on improved Dijkstra algorithm and ant system[J]. Journal CSUT (S1672-3104), 2006, 13(1): 80-86

[13] Zhang Chibin, Wang Xingsong. Complete Coverage Path Planning Based on Ant Colony Algorithm[C]// 15th International Conference on Mechatronics and Machine Vision in Practice. Auckland, New-Zealand, Dec. 2008: 357-361

[14] Cen Y, Song C. Path Planning Method for Mobile Robot Based on Ant Colony Optimization Algorithm[C]// 3rd IEEE Conference on Industrial Electronics and Applications. 2008: 298-301

[15] Tan Guan-zheng, He Huan. Ant Colony System Algorithm for Real-Time Globally Optimal Path Planning of Mobile Robots [J]. Acta Automatica Sinica, 2007, 33(3): 279-285

[16] 王俊峰. 基于改进蚁群算法的机器人路径规划[D]. 南京：南京师范大学，2005

[17] Dorigo M. 蚁群优化[M]. 张军，译. 北京：清华大学出版社，2007：90，39-41，76

[18] 段海滨. 蚁群算法原理及其应用[M]. 北京：科学出版社，2005：26

[19] de Berg M. 计算几何(第 2 版)[M]. 邓俊辉，译. 北京：清华大学出版社，2005：340

(上接第 122 页)

人只需给出源程序代码，就可以自动判断程序是否符合我们预定义的性质，降低了人们使用模型检验技术的难度。进一步研究包括优化本方法，扩充预定义性质，以达到分析更多程序性质的目标。

参 考 文 献

[1] Schmidt D A. Data flow analysis is model checking of abstract interpretations[C]// ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '98). New York, NY, USA: ACM Press, 1998

[2] Schmidt D A, Steffen B. Program analysis as model checking of abstract interpretations[C]// Intl. Symposium on Static Analysis (SAS '98). London, UK: Springer-Verlag, 1998: 351-380

[3] Benedikt M, Fan W, Kuper G M. Structural properties of xpath fragments[C]// ICDT '03: Proceedings of the 9th International Conference on Database Theory. London, UK: Springer-Verlag, 2002: 79-95

[4] Fehnker A, Huuck R, Jayet P, et al. Model Checking Software at Compile Time[C]// First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE '07). 2007: 45-56

[5] 叶俊民，张振方，王敬华，等. 基于模型检验技术的源程序分析研究[J]. 微电子学与计算机，2009(8)