症状描述

使用 DFA::usefulf 函数对一个含有下沉状态的 DFA 进行移除下沉态操作，该 DFA 如下（样例来自《形式语言与自动机理论》(ISBN 978-7-302-31802-6) pg. 141 fig. 5-4)：

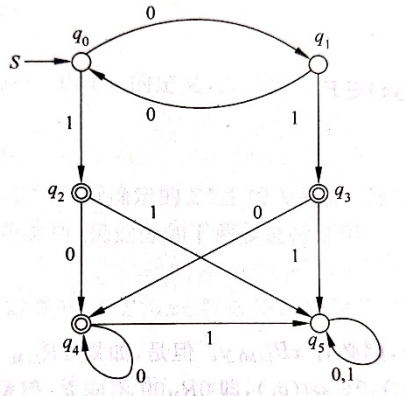
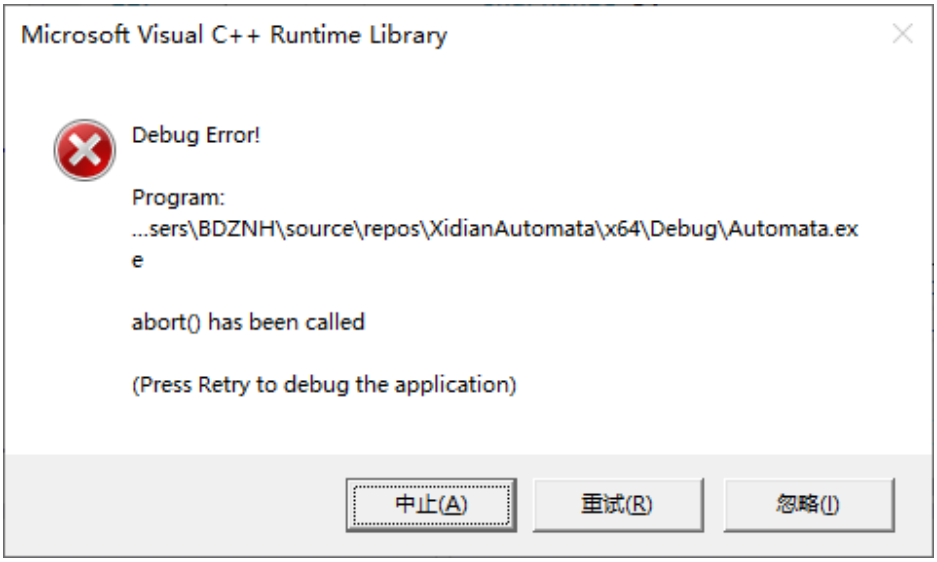


图 5-4　接受 $L=0^*10^*$ 的 DFA

涉及文件：
DFA.cpp(79), usefulf()。

usefulf()函数作用为去除自动机中的“not final-reachable”状态，如上的自动机中，q_5 状态即为符合要求状态，经过经过函数执行后，q5 状态被去除，则函数正常执行。

实际使用时，出现如下提示：



控制台报错：

到这里，可以确认 DFA::usefulf()函数未能完成其定义功能。

在 transimpl.cpp(79 行) 的 “assert(0 <= q);”处打断点，调试至此处，可以看到变量 q 的值为“-842150451”，十六进制值为“0xFFFFFFFF”，为常见的未初始化变量导致的错误，此处 q 为一个 State 变量，检查到一个值为负值的 State 变量时程序中止。观察函数 DFA::usefulf()的实现：

```
DFA_components ret;
State st;
for (st = 0; st < Q.size(); st++)
{
    // If this is a Usefulf State, carry it over by giving it a
name
    // in the new DFA.
    if (freachable.contains(st))
    {
        newnames.map(st) = ret.Q.allocate();
    }
}
```

这里将 Usefulf 的状态保存到 StateTo<State> 变量 newnames ，并且通过“ret.Q.allocate();”为这个状态命名一个新的状态名，作为新的自动机的状态名。再观察函数内构造新的自动机的实现代码的主要部分：

```
CRSet a;
for (State st = 0; st < Q.size(); st++)
{
    // If st is the representative, construct the transition.
    if (st == r.eq_class_representative(st))
    {
        State stprime(newnames.lookup(st));
        // What are st's out-transitions?
```

```
        CharRange b;
        a = T.out_labels(st);
        // The out-labels of any other element of [st]_r could have
        // been used instead. Some other choice may, indeed, lead
        // to a smaller DFA.This approach is used for simplicity.
        int it;
        // Iterate over the labels, constructing the transitions.
        for (it = 0; !a.iter_end(it); it++)
        {
            b = a.iterator(it);
            ret.T.add_transition(stprime, b,
newnames.lookup(T.transition_on_range(st, b)));
        }
        // st's eq. class may be final.
        if (F.contains(st)) ret.F.add(stprime);
    }
}
```

易知在引发程序中止的地方为“ret.T.add_transition(stprime, b, newnames.lookup(T.transition_on_range(st, b)));”其中 State 变量 stprime 为当前状态，CharRange 变量 b 为状态转移输入字符。

观察T.transition_on_range的实现代码（DtransRel.h 109行）：

```
// Compute the image of r, and CharRange it under *this.
inline State DTransRel::transition_on_range(const State r, const CharRange a) const
{
    assert(class_invariant());
    assert(0 <= r && r < domain());
    return(lookup(r).range_transition(a));
}
```

易知“T.transition_on_range(st, b)”将返回旧自动机的 st 状态经过字符 b 转移的目标状态。

观察newnames.lookup()的实现代码：

```
// The actual mapping function
// First, a const lookup operator.
template<class T>
inline const T& StateTo<T>::lookup(const State r) const
{
    assert(class_invariant());
    // First check that it's in bounds
    assert(0 <= r && r < domain());
    return(data[r]);
}
```

在本例中，此函数返回的是旧自动机的状态 r 所对应的新自动机的状态 data[r]。

经过调试发现，旧自动机的状态2经过字符“0”将转移到5状态，而在

```
DFA_components ret;
State st;
for (st = 0; st < Q.size(); st++)
{
    // If this is a Usefulf State, carry it over by giving it a
name
    // in the new DFA.
    if (freachable.contains(st))
    {
        newnames.map(st) = ret.Q.allocate();
    }
}
```

中并没有将状态 5 保存，所以函数newnames.lookup(T.transition_on_range(st, b))将返回一个未经初始化的值“-842150451(0xFFFFFFFF)”。而状态 5 是 sink 状态，去除自动机中的 5 状态也是DFA::usefulf()的作用，所以不应该将状态 5 保存到StateTo<State>变量 newnames 中。可以确定这里为一个bug。

修正方法

在 transimpl.cpp(79行) 的 “assert(0 <= q);”处可知，自动机要求 State 变量为自然数，在 State.h 中有这样的定义

```
// Invalid states mean something bad is about to happen.
const State Invalid = -1;
```

所以在保存新的自动机的状态名时，把未使用的状态全都标记为“Invalid”在构建新的自动机时，检查目标状态是否为一个有效状态即可。修正方法如下(红色为新增或修改部分)。

将

```
DFA_components ret;
State st;
for (st = 0; st < Q.size(); st++)
{
    // If this is a Usefulf State, carry it over by giving it a
name
    // in the new DFA.
    if (freachable.contains(st))
    {
        newnames.map(st) = ret.Q.allocate();
    }
}
```

更改为

```
DFA_components ret;
State st;
for (st = 0; st < Q.size(); st++)
{
```

```
    // If this is a Usefulf State, carry it over by giving it a name
    // in the new DFA.
    if (freachable.contains(st))
    {
        newnames.map(st) = ret.Q.allocate();
    }
    else
    {
        newnames.map(st) = Invalid;
    }
}
```

将 ret.T.add_transition(stprime, b, newnames.lookup(T.transition_on_range(st, b)));更改为

```
if(stprime != Invalid && newnames.lookup(T.transition_on_range(st,
b)) != Invalid)
{
    ret.T.add_transition(stprime, b,
newnames.lookup(T.transition_on_range(st, b)));
}
```

本例代码：

见附件