

Model-Based Testing in the Key of C#

Harry Robinson
Michael Corning
Microsoft

Reasonable Questions

- What is a model?
- Why use model-based testing?
- How can we use C# for model-based testing?

What is a Model?

- A model is a description of a system.
- Models are simpler than the systems they describe.
- Models help us understand and predict the behavior.

Why Use Model-Based Testing?

- Better understanding of the application
- Generate zillions (or scrillions) of tests
- Enables more agile testing

Let's test a simple app

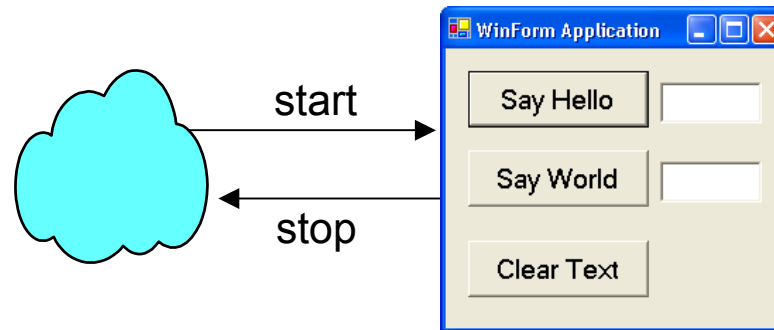


First, manual testing ...

Next, serial scripts ...

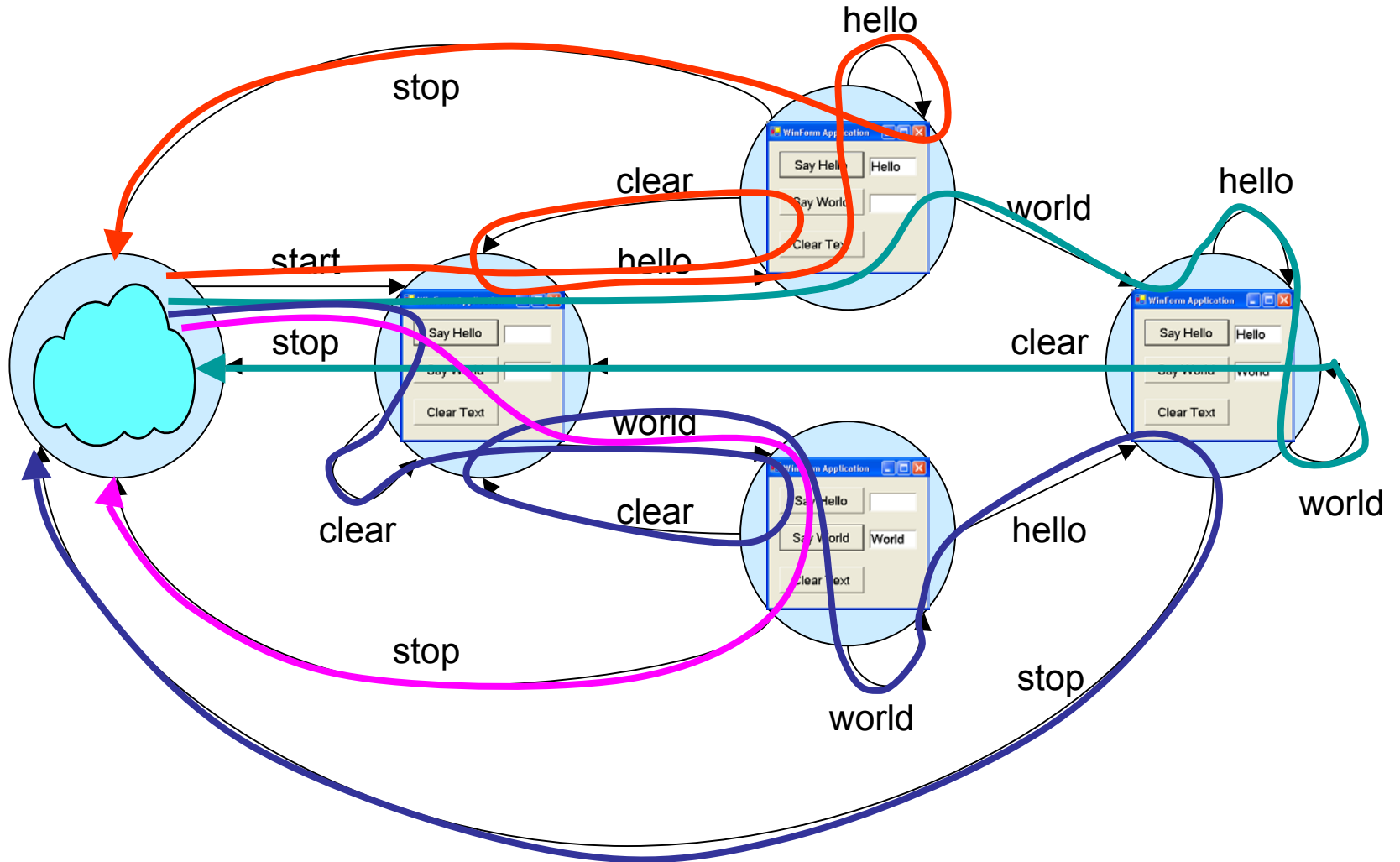


Finally, modeling



First, on the whiteboard ...

Hand-crafted traversals



These are the hand-generated traversals

Test 1

1. start
2. hello
3. clear
4. hello
5. hello
6. stop

Test 2

1. start
2. hello
3. world
4. hello
5. world
6. clear
7. stop

Test 3

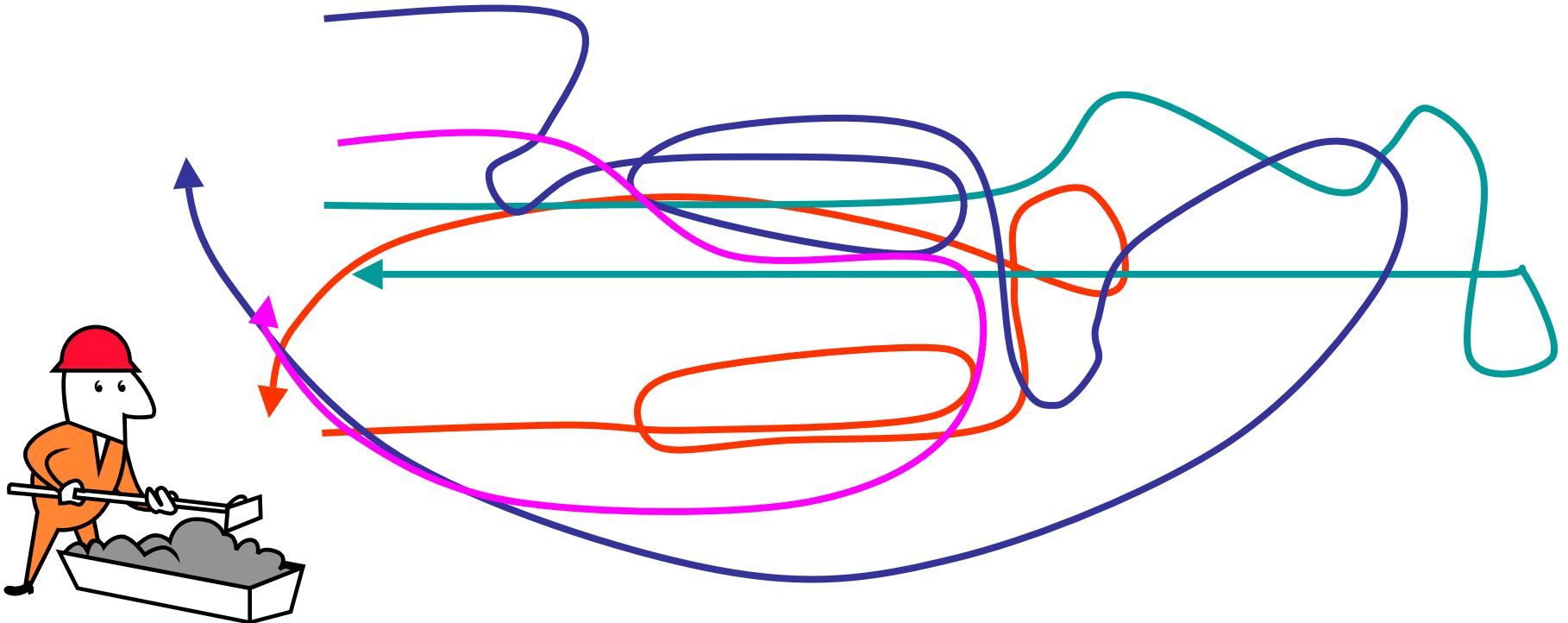
1. start
2. clear
3. world
4. clear
5. world
6. world
7. hello
8. stop

Test 4

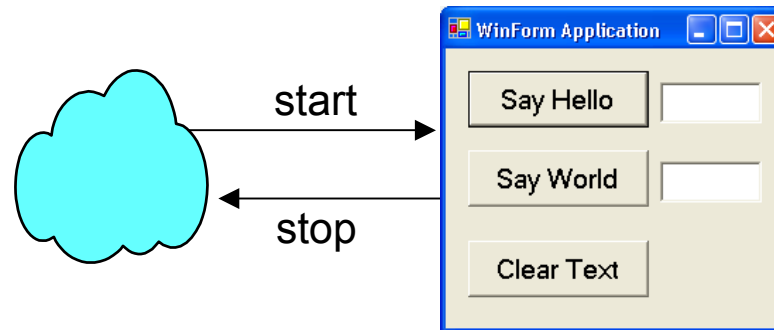
1. start
2. world
3. stop

But, really, what are you left with?

- Hand-coded test cases, maintained manually
- Tests that do only what you told them to
- Tests that no longer find bugs

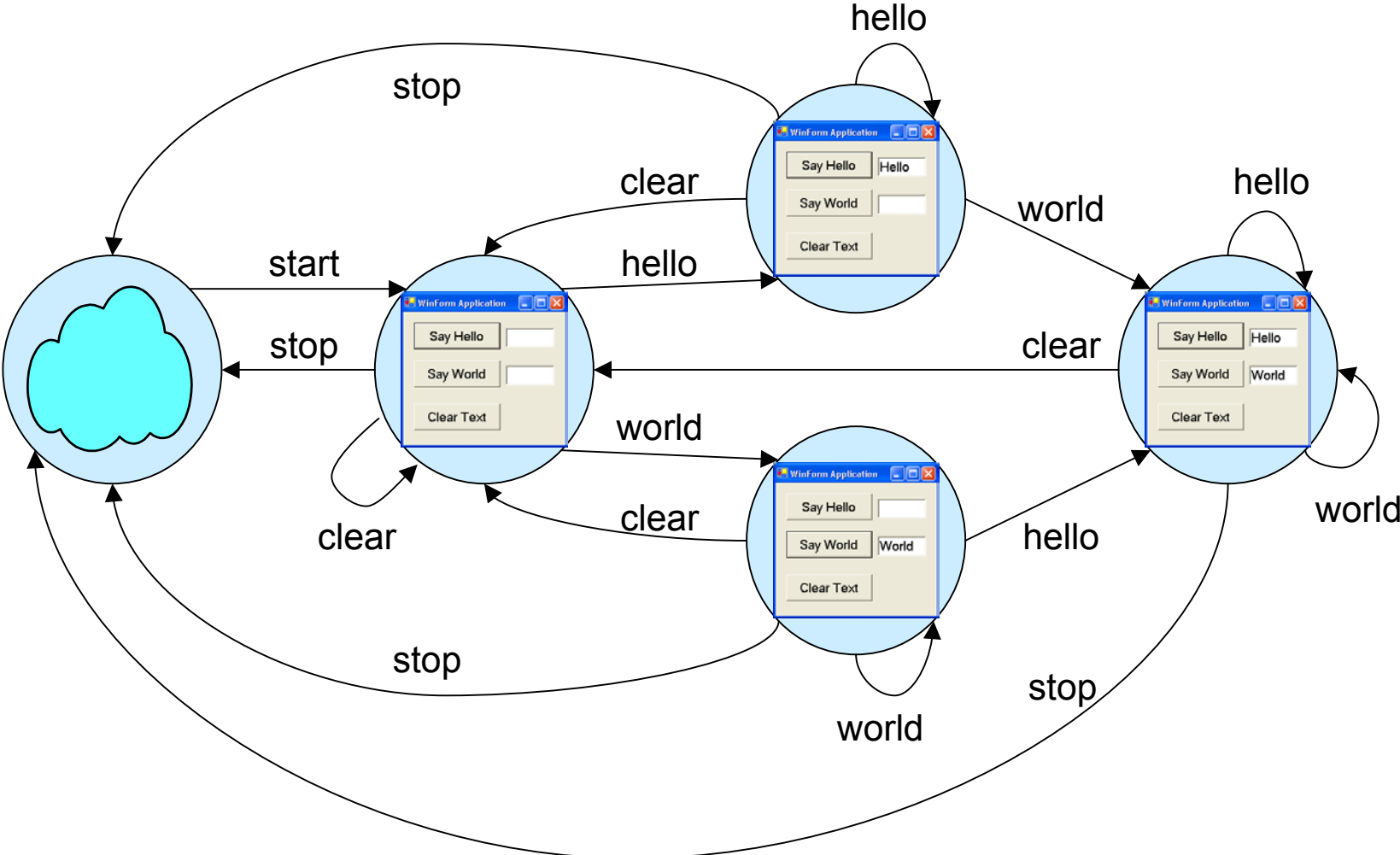


Let the machine do the heavy lifting



Make the model machine-readable ...

From whiteboard to machine-readable



We are Tracking 3 State Variables

The System is either

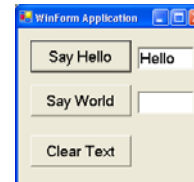
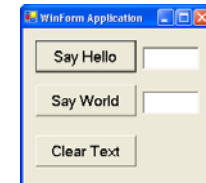
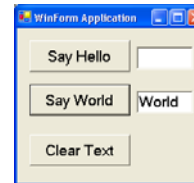
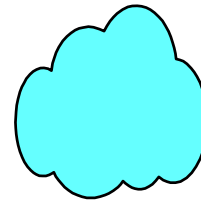
- Not Running or
- Running

The top field is either

- blank or
- Hello

The bottom field is either

- blank or
- World



Let the machine manage the model

Rule for the “Hello” Action:

- When the System variable is **Not Running**, the user cannot execute the **Hello** action.
- When the System variable is **Running**, the user can execute the **Hello** action.
- After the **Hello** action executes, the Top Field variable is set to **Hello**.

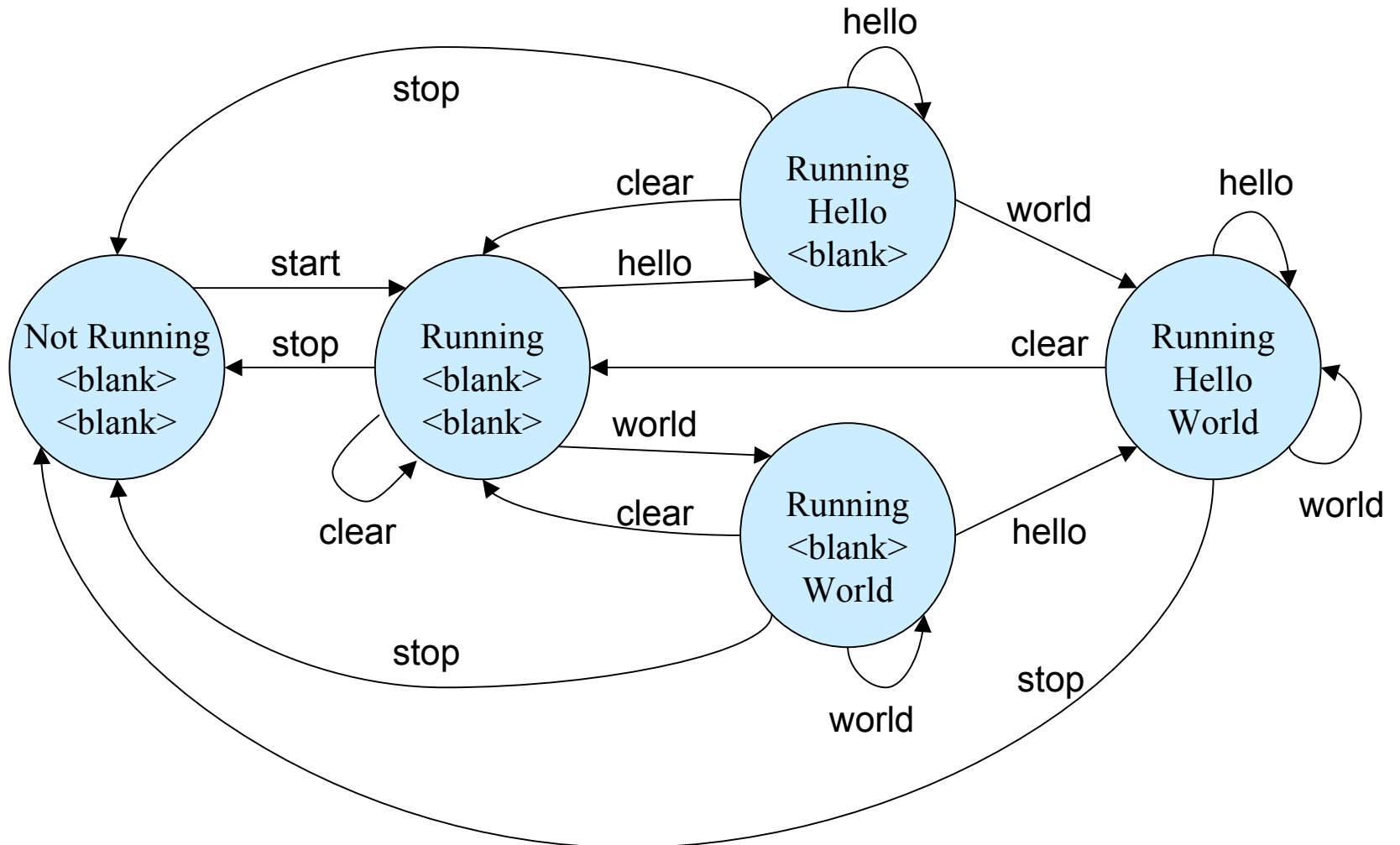
Rule for the “Stop” Action:

- When the System variable is **Not Running**, the user cannot execute the **Stop** action.
- When the System variable is **Running**, the user can execute the **Stop** action.
- After the **Stop** action executes, the System variable is set to **Not Running**.

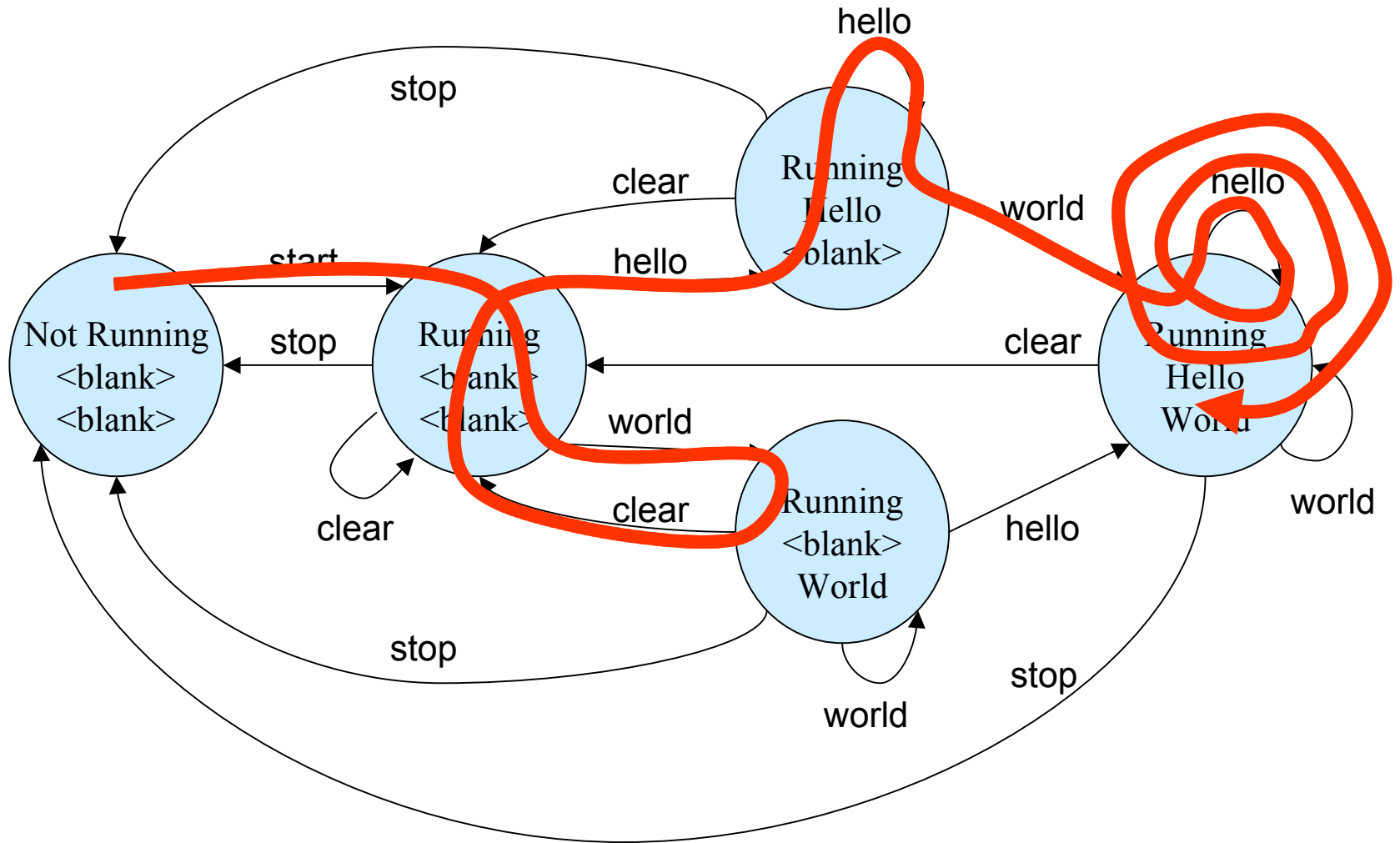
Generated State Table

Beginning State	Action	Ending State
NotRunning Blank Blank	Start	Running Blank Blank
Running Blank Blank	Hello	Running Hello Blank
Running Blank Blank	World	Running Blank World
Running Blank Blank	Clear	Running Blank Blank
Running Blank Blank	Stop	NotRunning Blank Blank
Running Hello Blank	World	Running Hello World
Running Hello Blank	Clear	Running Blank Blank
...

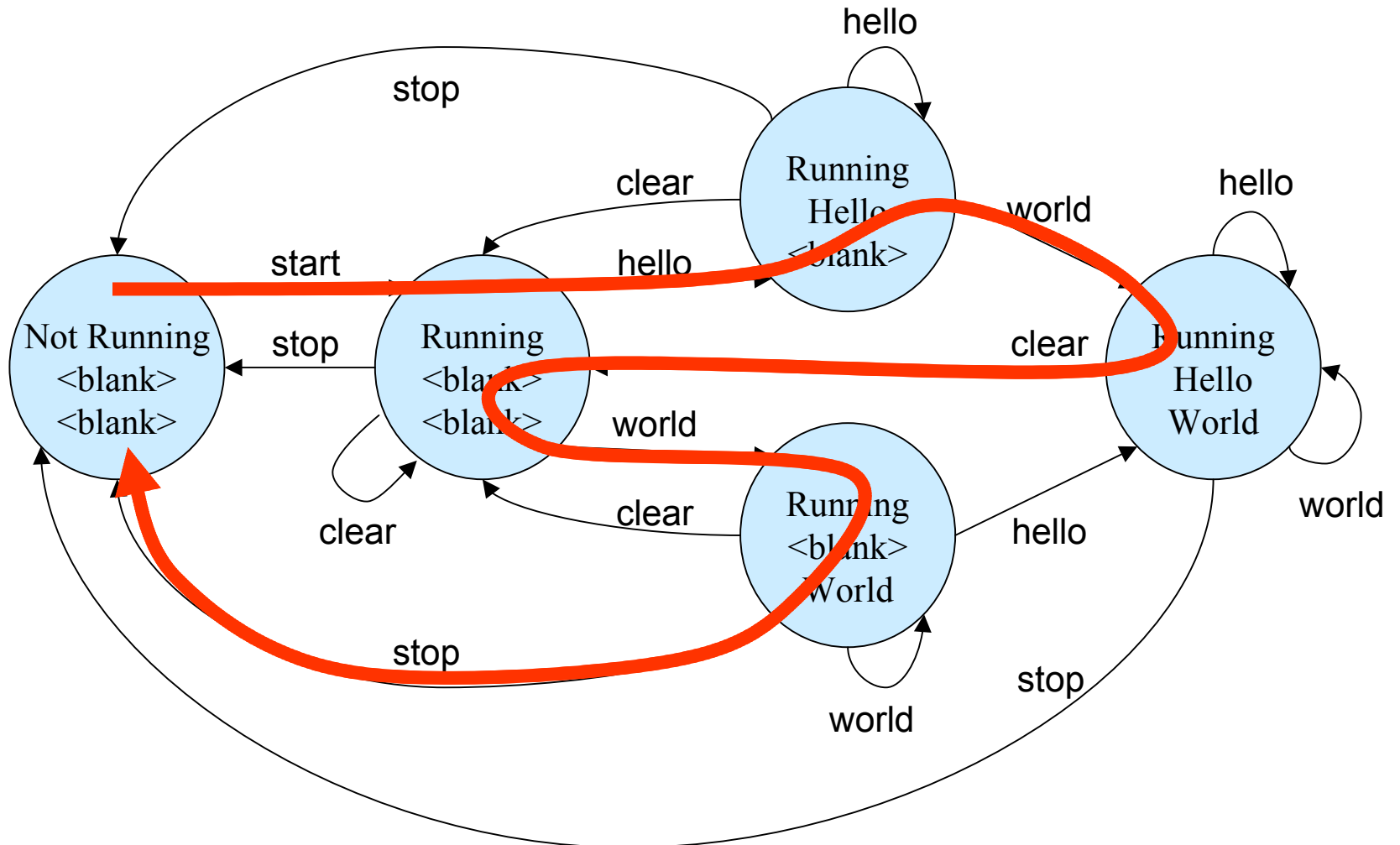
State Model



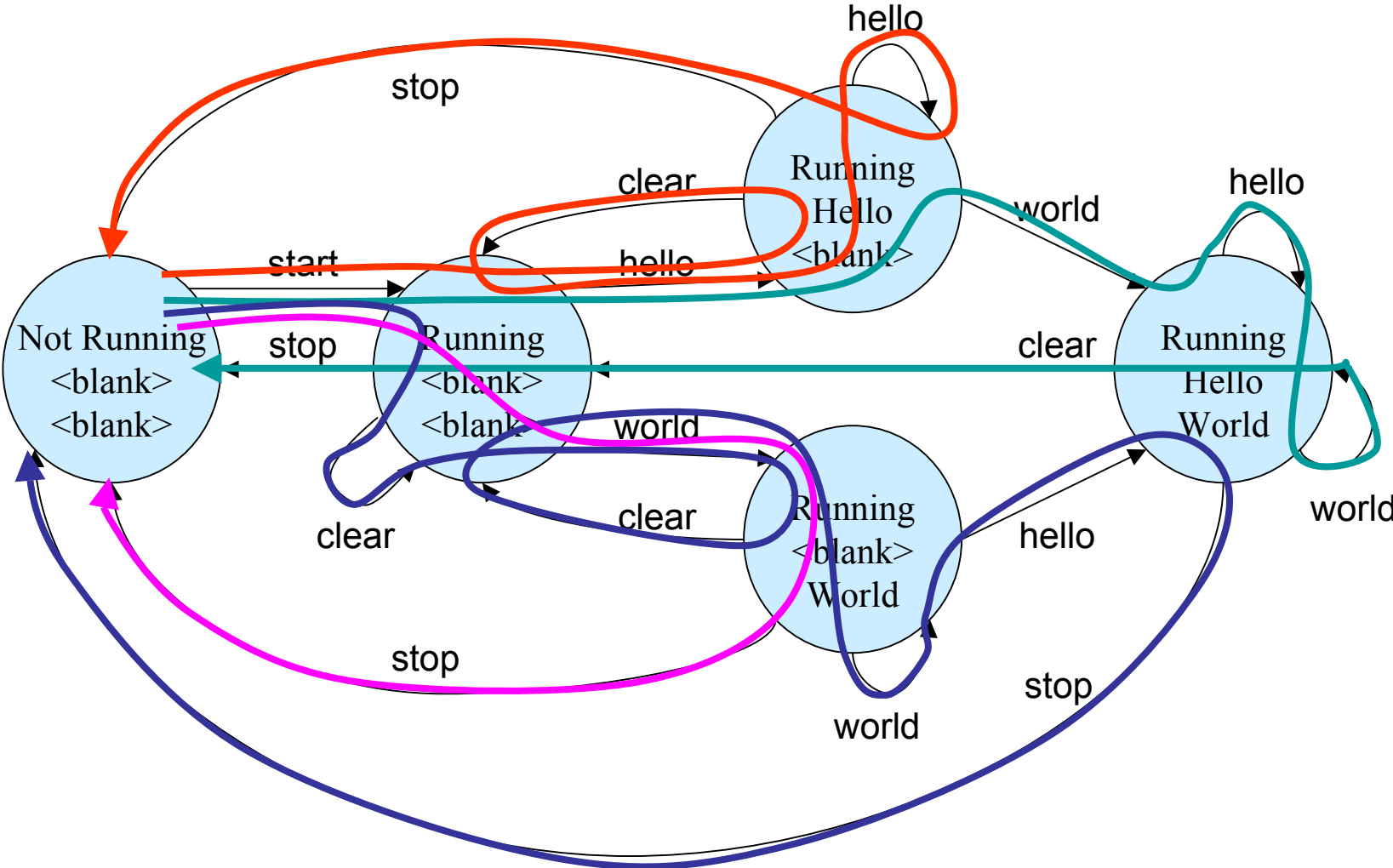
Random Walk



Visit All States



Exercise All Transitions



How can we use C#
for model-based testing?

Specifying rules in C# model

```
static string AnEnabledAction()
{
    ArrayList a = new ArrayList();

    if (AppStatus == AppStatusValues.NotRunning) a.Add("StartApp");
    if (AppStatus == AppStatusValues.Running)    a.Add("StopApp");
    if (AppStatus == AppStatusValues.Running)    a.Add("SayHello");
    if (AppStatus == AppStatusValues.Running)    a.Add("SayWorld");
    if (AppStatus == AppStatusValues.Running)    a.Add("Clear");

    return (string) a[r.Next(a.Count)];
}
```

Generating traversals in C# model

```
using (StreamWriter sw = File.CreateText(path))
{
    for ( int i=0; i<NrOfSteps; i++)
    {
        switch (action = AnEnabledAction())
        {
            case "StartApp": StartApp(); break;
            case "StopApp": StopApp(); break;
            case "SayHello": SayHello(); break;
            case "SayWorld": SayWorld(); break;
            case "Clear": Clear(); break;
        }

        Console.WriteLine(action + " : " + AppStatus + " " + TextBox1Status+ " " + TextBox2Status);
        sw.WriteLine(action);
    }
}
```

Executing the Test Actions

```
while ((s = sr.ReadLine()) != null)
{
    switch(s)
    {
        case "StartApp":
            startApp();
            break;
        case "StopApp":
            stopApp();
            break;
        case "SayHello":
            InvokeMethod(testForm, "sayHello_Click", p);
            break;
        case "SayWorld":
            InvokeMethod(testForm, "sayWorld_Click", p);
            break;
        case "Clear":
            InvokeMethod(testForm, "clearText_Click", p);
            break;
    }
}
```

Verifying outcome in C# test code

```
string oracle = (string)(GetProperty(GetField(testForm, "textBox2"), "Text"));

if(oracle=="Bug!")
{

    Trace.WriteLine("Bug found!");

    MessageBox.Show("Bug found!",

        "Test Oracle",

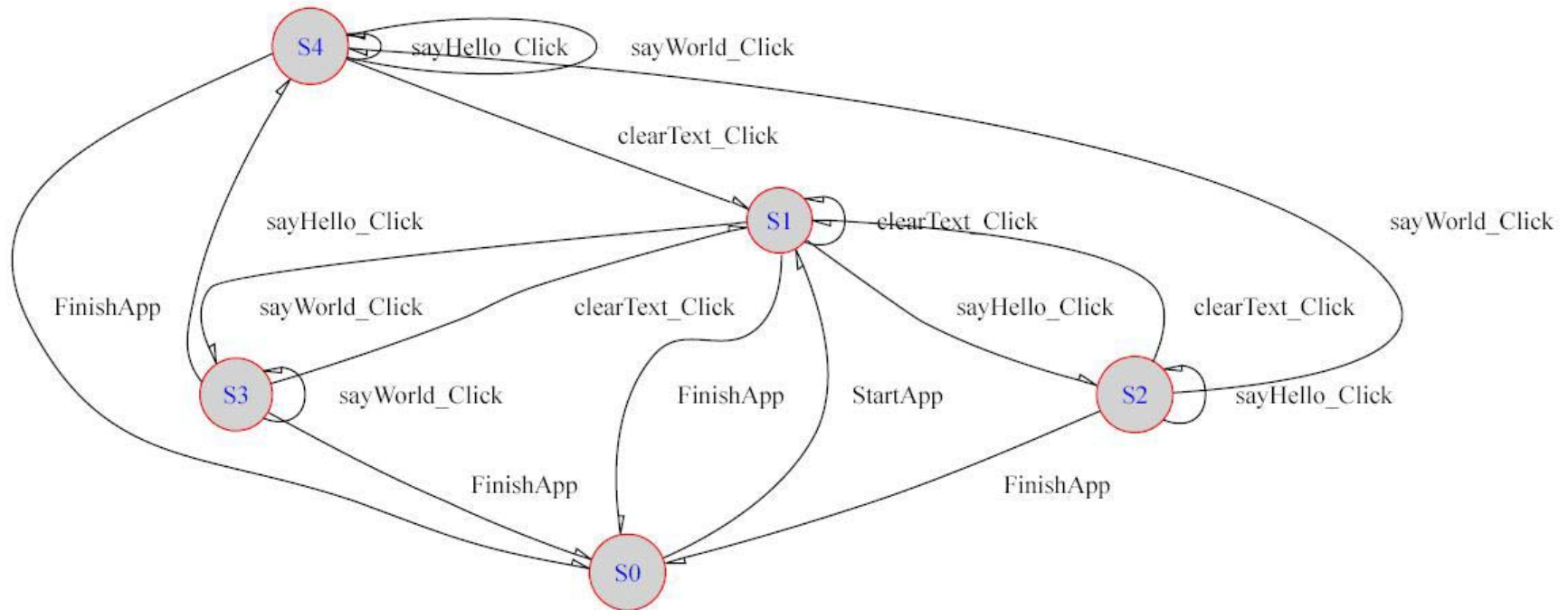
        MessageBoxButtons.OK,

        MessageBoxIcon.Error);

    break;

}
```

Goldilocks Generated State Graph



Let's see how this got made...

Benefits of Model-Based Testing

- Easy test case maintenance
- Reduced costs
- More test cases
- Early bug detection
- Increased bug count
- Time savings
- Time to address bigger test issues
- Improved tester job satisfaction

Obstacles to Model-Based Testing

- Comfort factor
 - This is not the way the industry does test automation
- Skill sets
 - Need testers who can design
- Expectations
 - Models can be a significant upfront investment
 - Will never catch all the bugs
- Metrics
 - Bad metrics: bug counts, number of test cases
 - Better metrics: spec coverage, code coverage

Resources

- Model-based testing website:

www.model-based-testing.org

- Books:

“Black-Box Testing : Techniques for Functional Testing of Software and Systems” by Boris Beizer

“Testing Object-Oriented Systems: Models, Patterns, and Tools” by Robert Binder

“Software Testing: A Craftsman's Approach” by Paul Jorgensen