Righting Software:

Tools to Improve Software Development

James Larus
Microsoft Research
March 2004

Why is Software Development so Hard?

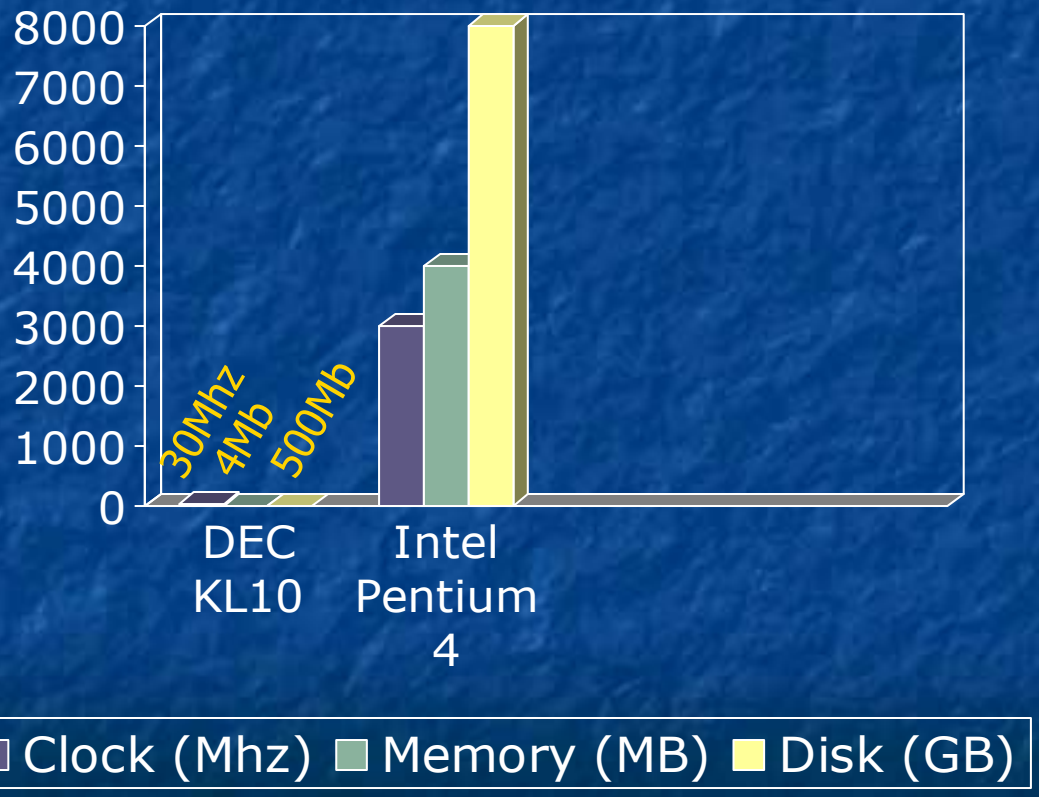
- No one is happy with final product
  - Consumers
  - Developers
- Development is painful and unreliable process
- Computers (tools) should be able to help

Overview of Microsoft Tools Effort

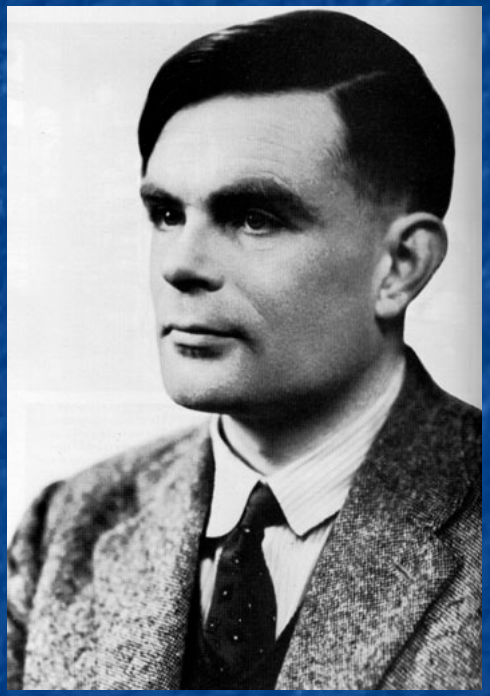
- 1998 – present
- Focused on Microsoft Research efforts
  - Internal tools
  - Innovative research
  - Gradually moving out to products

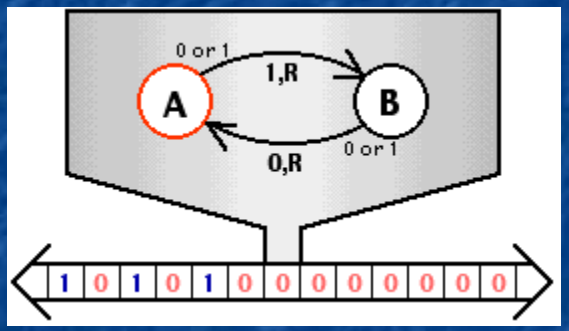
Software Development Tools Are Old

Processor Performance 1978-2003



Software Tools Can't Find All Bugs



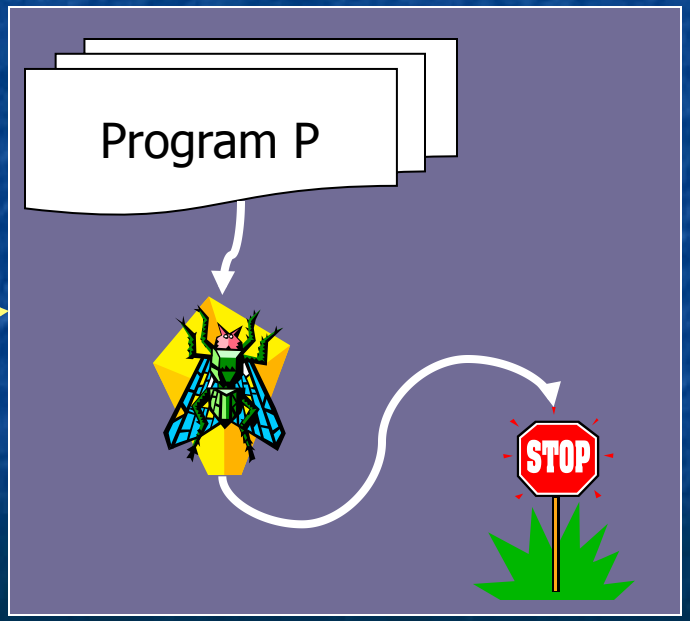


On Computable Numbers, with an application to the Entscheidungsproblem, Proceedings London Mathematical Society (series 2) vol 42, 1936-7, pp.230-265.

Turing Halting Problem

- No program, given an arbitrary program P and its input I, can decide if P halts on I

Does program P' execute the bug?



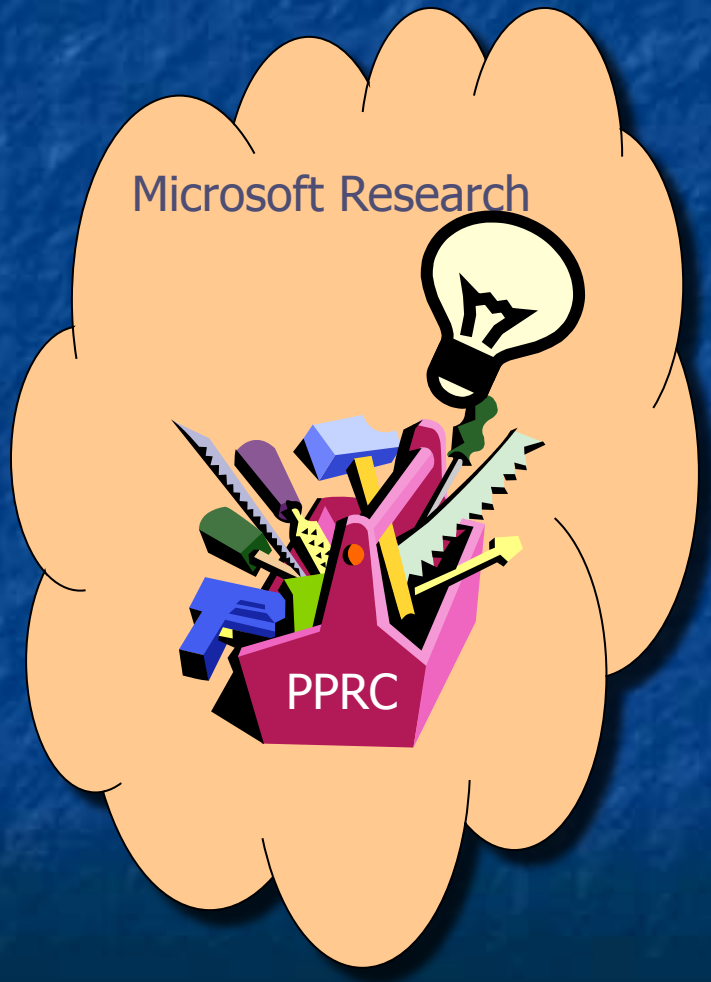
If yes, then P must not halt.

Computation to Aid Software Development

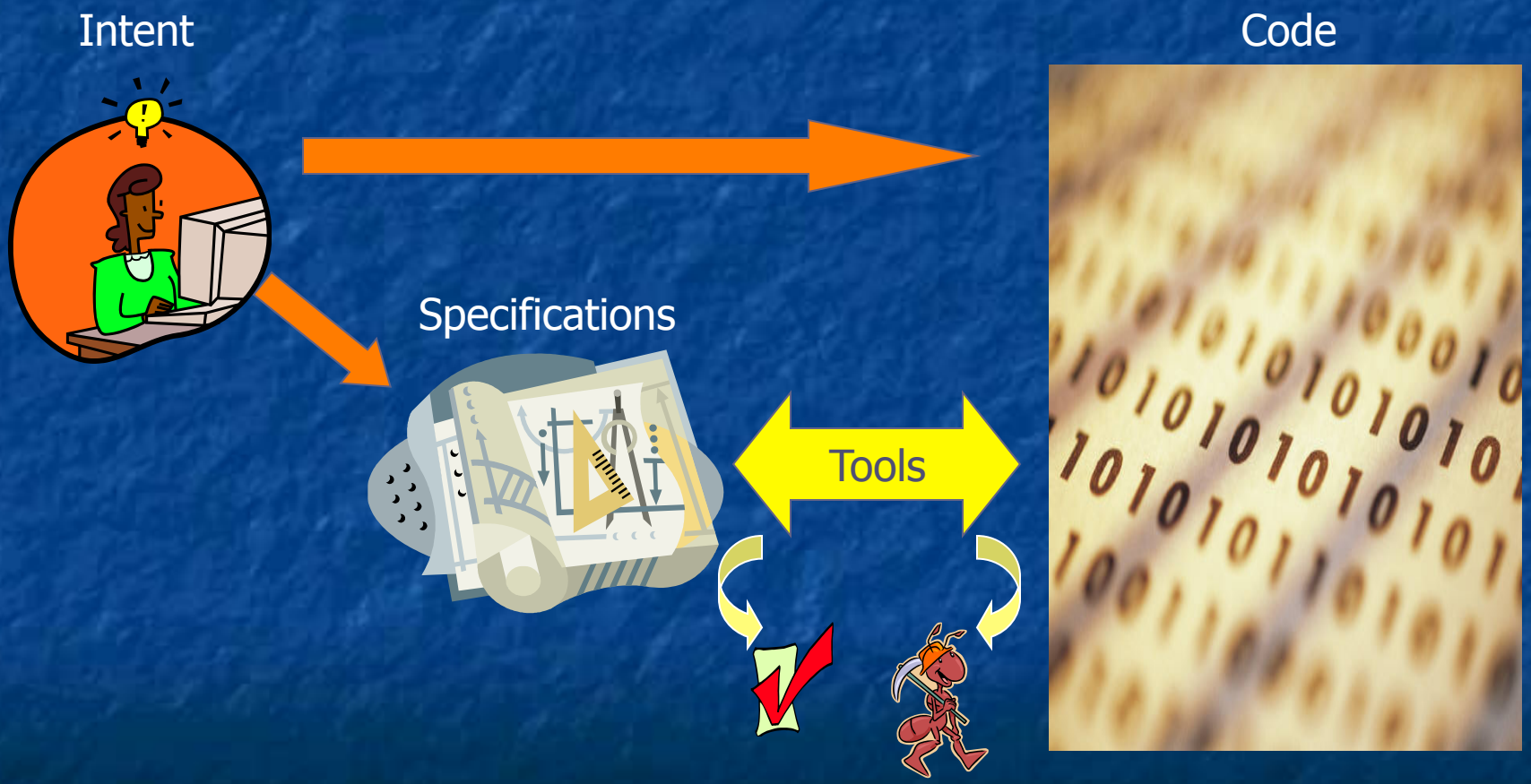
- Aid (not replace) developers & testers
- Amplify human effort
  - manage details
  - find inconsistencies
  - ensure quality
- Goal is not perfection (verification)

3 Generations of PPRC Correctness Tools

1. Scalable, heuristic tools
   - PREfix, PREfast
2. Sound, declarative tools
   - SLAM, Fugue, ESP
3. Targeted tools
   - concurrency, security



Tools Bridge Intent and Code



Types of Errors

- Language usage
  - uninitialized variable, null dereferences, …
- API usage
  - close file twice, ignore error result, hold lock, …
- Semantic errors
  - deadlock
- …
- Incorrect computation (≈program verification)
  - Not our goal!

Static Program Analysis

- Tool explores all possible executions (and then some)
  - complements run-time checking
    - completeness vs. precision
  - analyses exist for only a few properties
- Choose at most one:
  - sound analysis finds all instances of error
  - complete analysis reports no false errors
  - can't have both: Turing halting problem

Static Analysis Limitations

```
FILE* f;
if (complex_calc1())
    f = fopen(…);
…
if (complex_calc2())
    fclose(f);
```

Outline

- 1st generation tools (scalable, heuristic)
  - Prefix
  - Prefast
- 2nd generation tools
- 3rd generation tools
- Conclusion

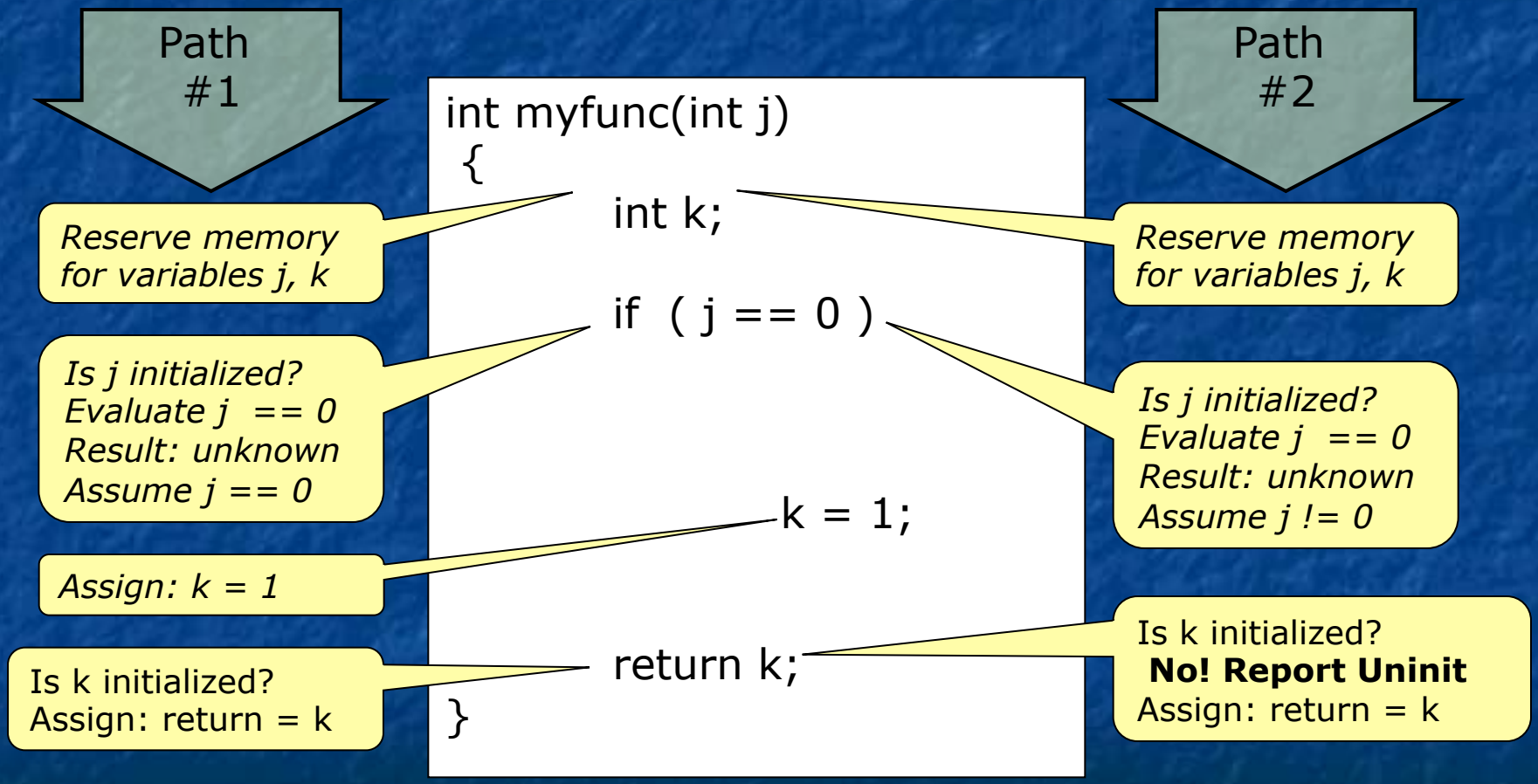
PPRC's First Correctness Tools

- PREfix
  - detailed, path-by-path interprocedural analysis
  - heuristic (unsound, incomplete)
  - expensive (4 days on Windows)
  - effective at finding bugs
- PREfast
  - user-supplied plug-ins find bugs by traversing AST
  - desktop use, easily customized
- Widely deployed in Microsoft
  - 1/6 of bugs fixed in Windows Server 2003 found by these tools

PREfix Tool

- Detects errors in C/C++ code
  - null pointer, memory allocation, uninitialized value, resource state, library usage, …
- Interprocedural
  - bottom up traversal of call graph
  - models routine by examining limited set of paths (100)
  - apply model at call site
  - expensive, batch computation for large systems
- Large effort to minimize effects of false positives
  - filtering and prioritizing error reports
  - heuristics tuned to reduce noise (at cost of precision)

PREfix Example



PREfast Example

What's wrong with this code?

```
char *p1 = new char[10];
char *p2 = new char[10];
…
delete[] p2;
delete[] p1;
```

Can leak memory <*pointer*> due to an exception from second new operation.

```
char *p1 = new char[10];
char *p2;
try { p2 = new char[10]; }
catch (std::bad_alloc *e) {
 delete[] p1;
}
…
delete[] p2;
delete[] p1;
```

Yes, its ugly,
and necessary in a non-GC language.

SpecStrings

- Company-wide effort to annotate C/C++ APIs
  - find buffer overruns
  - built on PREfast
  - standard annotation language (SAL)

```
BOOL WINAPI SetupGetStringFieldW( …
                          __ecount(ReturnBufferSize) OUT PWSTR ReturnBuffer,
                          IN DWORD ReturnBufferSize,
        …);

WCHAR szPersonalFlag[20];
…
SetupGetStringFieldW(&Context, 1, szPersonalFlag, 50, NULL);
```

PREfast: warning 202: Buffer overrun for stack buffer 'szPersonalFlag' in call to 'SetupGetStringFieldW': length 100 exceeds buffer size 40.

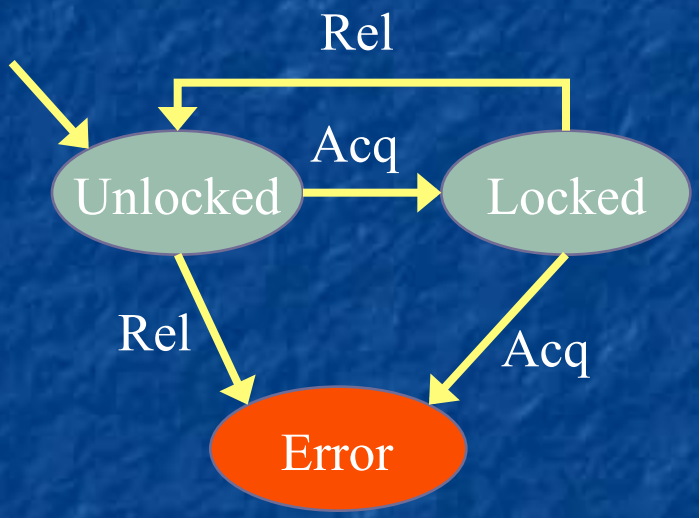
Outline

- 1st generation tools
- 2nd generation tools (sound, declarative)
  - SLAM
  - ESP
  - Fugue
- 3rd generation tools
- Conclusion

SLAM Tool

- Software model checking
- Input
  - C source code “as is”
  - API rules in SLIC language
- Automatically create abstraction of C program
  - abstract model = Boolean program
- Systematic exploration of model’s state space
  - does feasible path lead to error state in SLIC spec?
- Demand-driven refinement of model
  - exclude infeasible paths

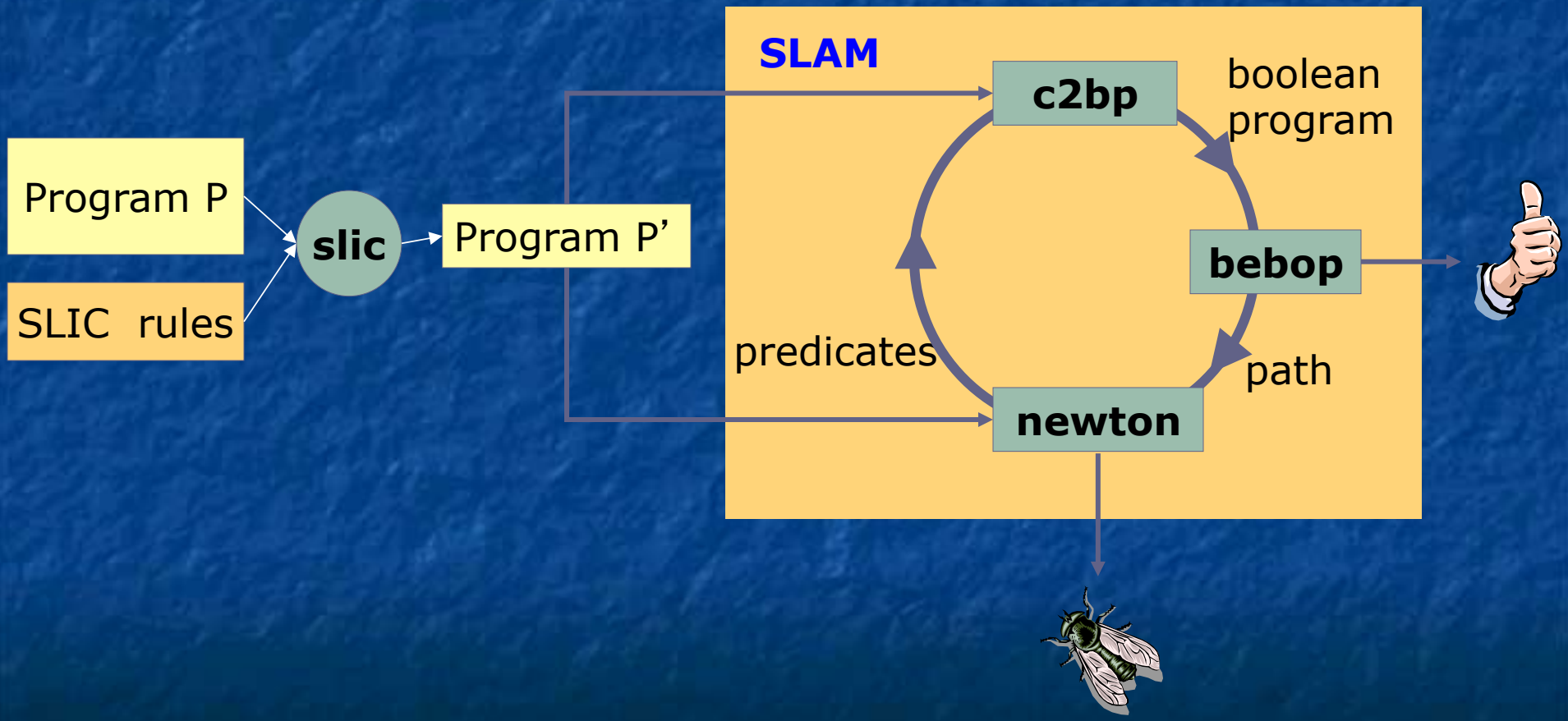
State Machine for Locking



Locking Rule in SLIC

```
state {
  enum {Locked,Unlocked}
       s = Unlocked;
}
KeAcquireSpinLock.entry {
  if (s==Locked) abort;
  else s = Locked;
}
KeReleaseSpinLock.entry {
  if (s==Unlocked) abort;
  else s = Unlocked;
}
```

SLAM Process



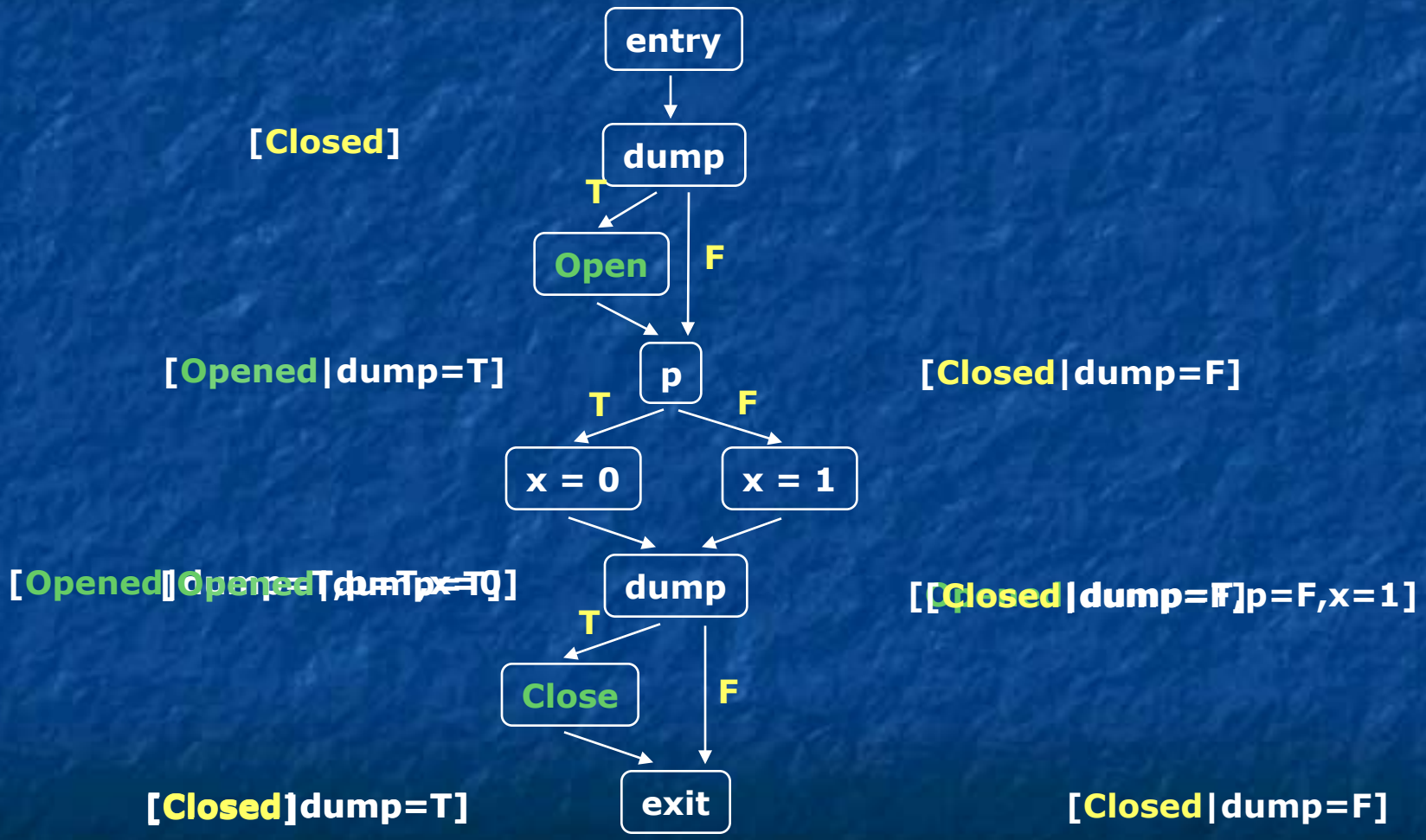
ESP Tool

- Systematically find errors in large programs (10+ MLOC)
  - scalable and accurate program analysis
  - trade little precision for large scalability
  - sacrifice specification complexity
- ESP:
  - scalable whole-system value flow analysis
  - scalable module-level path-sensitive analysis
  - finite-state property specification

Path Sensitive Analysis

- Rules expressed as finite-state automata (FSA)
- Tool symbolically evaluates program
  - along each path, track
    - FSA state
    - program state
  - watch for FSA transitions into error state
- At branches
  - does program state determines branch direction?
    - yes: process appropriate branch
    - no: process both branches, updating state
- Exponential growth in state size at branches

Example


entry
[Closed]
dump
T
Open
F
[Opened|dump=T]
p
[Closed|dump=F]
T
F
x = 0
x = 1
dump
T
Close
F
exit
[Closed]dump=T]
[Closed|dump=F]

Fugue Tool

- Make common API specifications statically checkable
  - can this method return null?
  - who owns this resource? do I have to free it?
  - do I have to call these methods in a particular order?
  - do these fields have data invariants I have to obey?
- Make documented rules part of the API itself

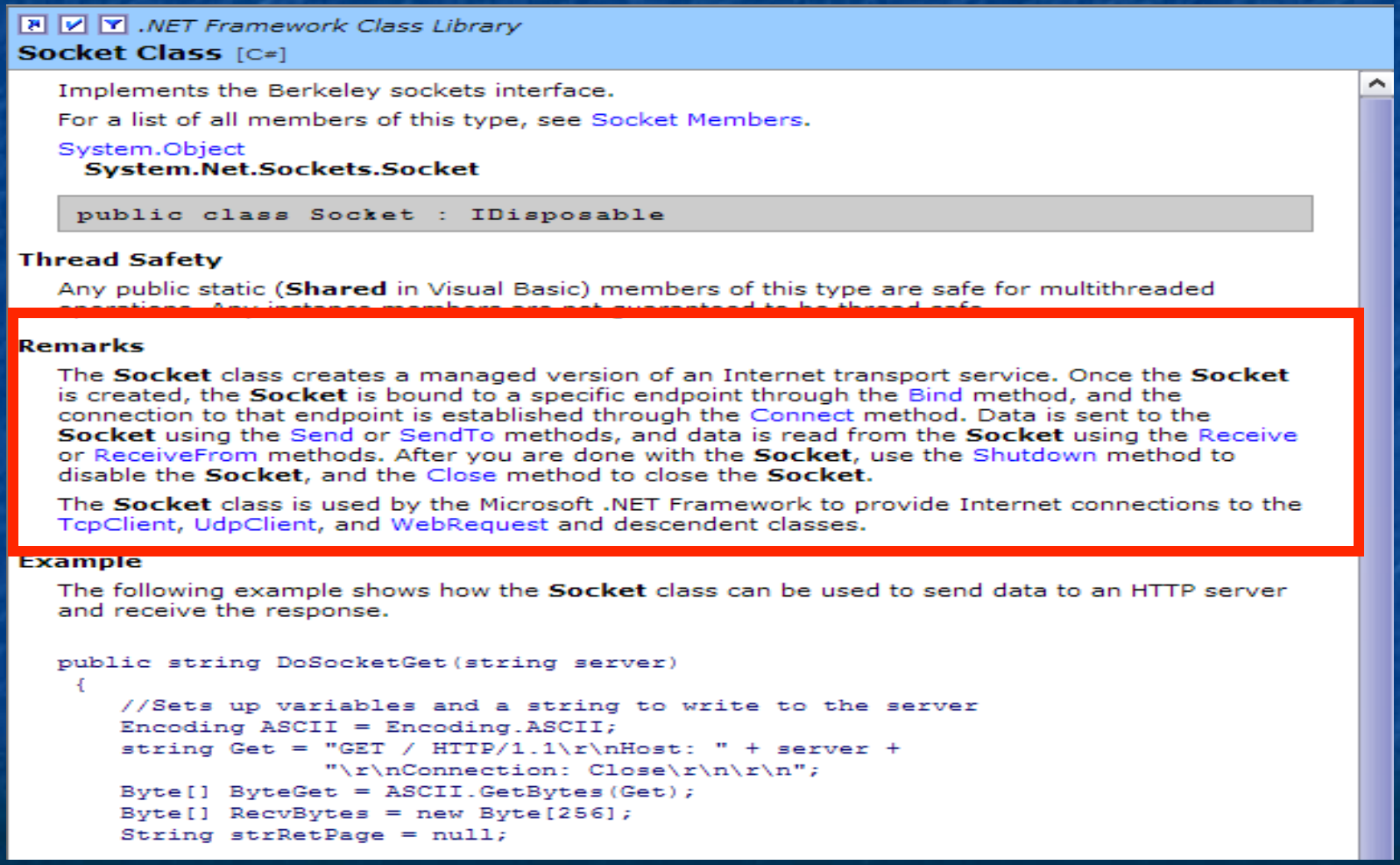
```
/// <param name="url">the address of a web page to fetch</param>
/// <returns><para>A string containing the content of
/// the web page at the <paramref name="url"/></para></returns>
/// <exception cref="System.ArgumentNullException">
///  <paramref name="url"/> is <see langword="null"/>.
/// </exception>
public string GetPage (string url);
```

Fugue Tool

- Make common API specifications statically checkable
  - can this method return null?
  - who owns this resource? Do I have to free it?
  - do I have to call these methods in a particular order?
  - do these fields have data invariants I have to obey?
- Make documented rules part of the API itself

```
[return:NotNull]
public string GetPage ([NotNull] string url);
```

API Usage Rules



.NET Framework Class Library

Socket Class [C#]

Implements the Berkeley sockets interface.

For a list of all members of this type, see Socket Members.

System.Object
  System.Net.Sockets.Socket

```
public class Socket : IDisposable
```

Thread Safety

Any public static (**Shared** in Visual Basic) members of this type are safe for multithreaded

Remarks

The **Socket** class creates a managed version of an Internet transport service. Once the **Socket** is created, the **Socket** is bound to a specific endpoint through the Bind method, and the connection to that endpoint is established through the Connect method. Data is sent to the **Socket** using the Send or SendTo methods, and data is read from the **Socket** using the Receive or ReceiveFrom methods. After you are done with the **Socket**, use the Shutdown method to disable the **Socket**, and the Close method to close the **Socket**.

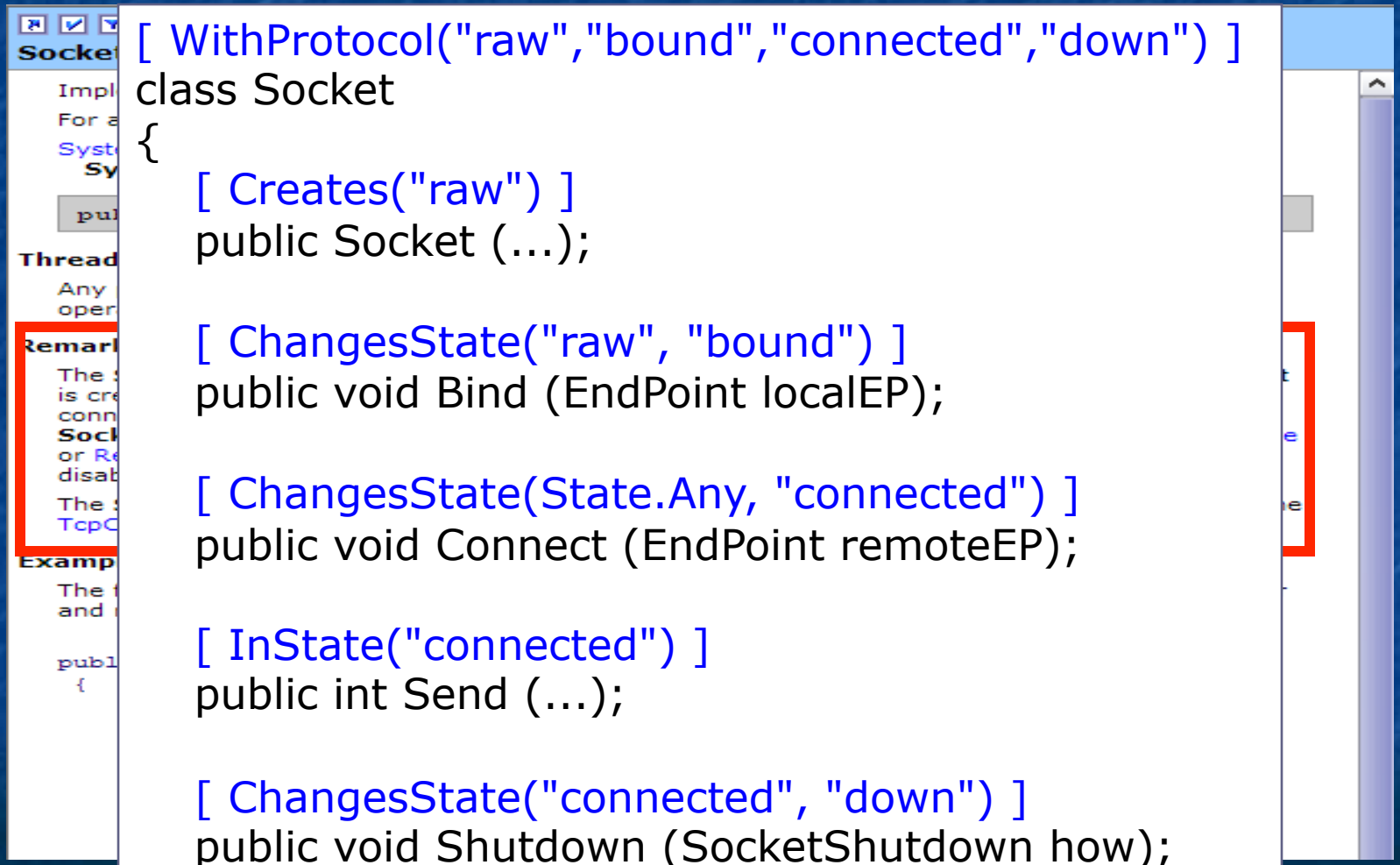
The **Socket** class is used by the Microsoft .NET Framework to provide Internet connections to the TcpClient, UdpClient, and WebRequest and descendent classes.

Example

The following example shows how the **Socket** class can be used to send data to an HTTP server and receive the response.

```
public string DoSocketGet(string server)
 {
    //Sets up variables and a string to write to the server
    Encoding ASCII = Encoding.ASCII;
    string Get = "GET / HTTP/1.1\r\nHost: " + server +
                 "\r\nConnection: Close\r\n\r\n";
    Byte[] ByteGet = ASCII.GetBytes(Get);
    Byte[] RecvBytes = new Byte[256];
    String strRetPage = null;
```

API Usage Rules



```
[ WithProtocol("raw","bound","connected","down") ]
class Socket
{
    [ Creates("raw") ]
    public Socket (...);

    [ ChangesState("raw", "bound") ]
    public void Bind (EndPoint localEP);

    [ ChangesState(State.Any, "connected") ]
    public void Connect (EndPoint remoteEP);

    [ InState("connected") ]
    public int Send (...);

    [ ChangesState("connected", "down") ]
    public void Shutdown (SocketShutdown how);
}
```

Outline

- 1st generation tools
- 2nd generation tools
- 3rd generation tools (targeted)
  - security
  - concurrency
- Conclusion

Directions for New Tools

- Increased expressiveness
  - general pre/post-condition and object invariants
- Specialized problems
  - security
  - concurrency
- Combine static and run-time analysis
- New analytic techniques
  - systematic state exploration
  - theorem proving
  - SAT solvers

Outline

- 1st generation tools
- 2nd generation tools
- 3rd generation tools
- Conclusion

Lessons

- Heuristics suffice
  - users happy to find (some) bugs
  - soundness is additional benefit
- User interface is crucial
  - PREfix spent more effort filtering than finding bugs
- Developers are rational
  - usage based on expected cost-benefit tradeoff
- Simple tools pave way for more sophisticated tools
  - change developer attitude and expectations
  - eliminating simple bugs exposes complicated ones

Conclusion

- Tools amplify human effort
  - find significant bugs in huge code bases
  - eliminate entire classes of bugs
  - set quality bar
  - improve developer & tester productivity
- Long way to go
  - crude and inaccurate tools
  - limited class of bugs
  - program analysis is difficult
- Part of larger effort
  - design and modeling
  - process improvement
  - adoption of engineers' mindset

http://research.microsoft.com/spt
http://research.microsoft.com/pprc