Programatica: The early years

A personal recollection

Mark P Jones, OGI

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- Programatica Demo
  - Standard Prelude
    - PreludeCore
    - PreludeNumeric
    - PreludeList
    - PreludeText
    - PreludeIO
  - Standard Libraries
  - Monad Transformers
    - MonadT
    - EnvT
    - StateT
    - MaybeT
    - ContT
  - Separation Model
    - Memory
    - Algorithms
    - RegFile
    - ChipModel

What's new in Programatica 3.2?

Welcome to the new release of Programatica! Here's a quick overview of some of the new features we've added, including the new mechanisms for liability and contract management. Further details can be found at any time under the Help menu or b We hope you'll enjoy this new release!

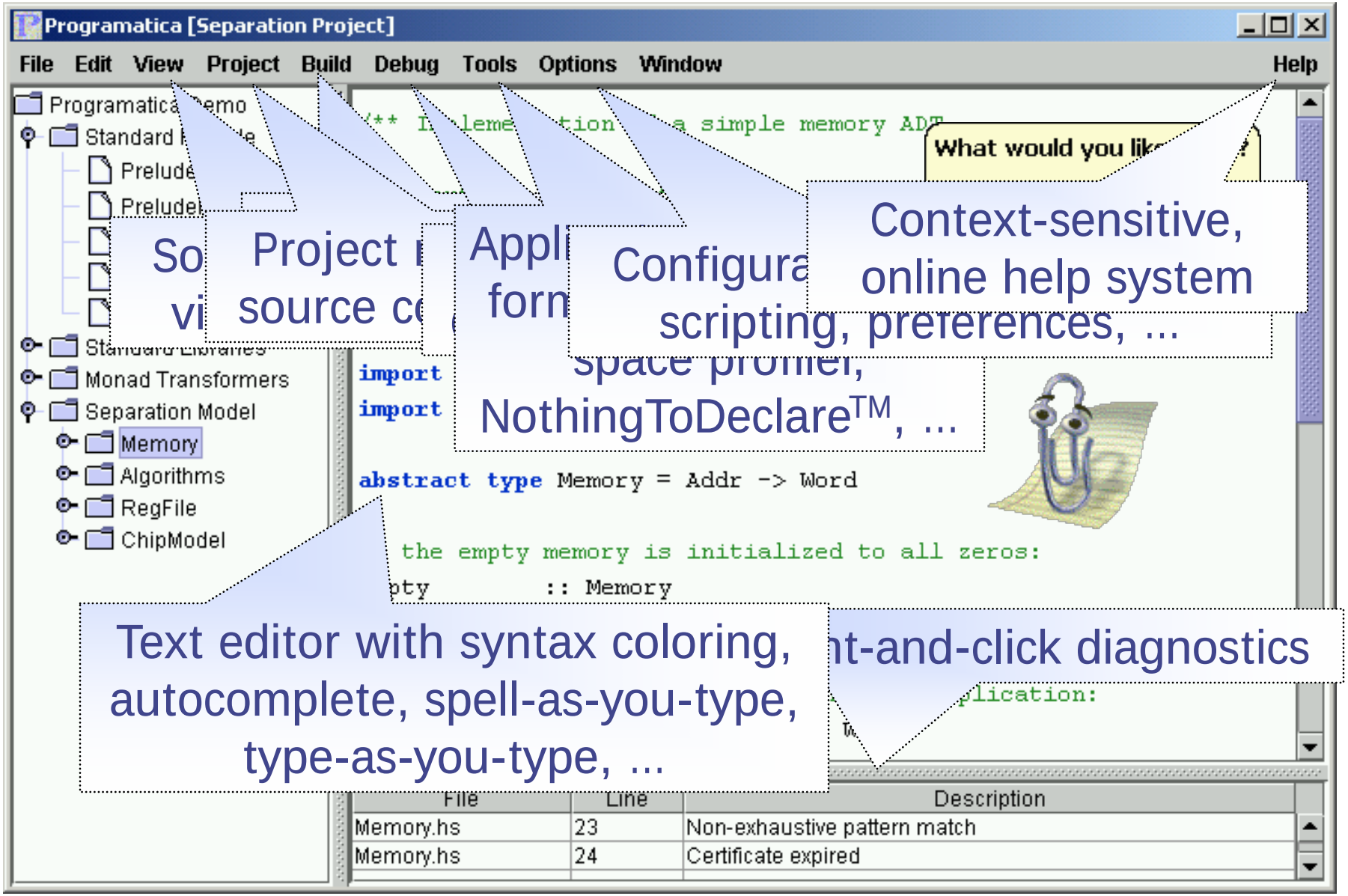
Of course, es that you've come to expect from previ

- Fu ard;
- Se PJ Hugs;
- Co to all of the most popular the ckers, and space invader game consoles;
- Hotsync with your PDA using ProofConduit;
- Automatic update and maintainence via programatica.com's unique "we'll be back" service;
- Polymorphicality analysis with Programatica's uniquely approximate pseudowobble auto variance;
- $100 mail-in rebate on grocery purchases over $4500 at

Programatica

Build 2322 January 16, 2003

OK


Programatica [Separation Project]
File Edit View Project Build Debug Tools Options Window Help
Standard Libraries
Monad Transformers
Separation Model
Memory
Algorithms
RegFile
ChipModel
import
import
abstract type Memory = Addr -> Word
the empty memory is initialized to all zeros:
:: Memory
What would you like
File Line Description
Memory.hs 23 Non-exhaustive pattern match
Memory.hs 24 Certificate expired
Project
source c
Appl
forn
Configura
scripting, preferences, ...
Context-sensitive,
online help system
space profiler,
NothingToDeclare™, ...
Text editor with syntax coloring,
autocomplete, spell-as-you-type,
type-as-you-type, ...
nt-and-click diagnostics

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- Programatica Demo
  - Standard Prelude
    - PreludeCore
    - PreludeNumeric
    - PreludeList
    - PreludeText
    - PreludeIO
  - Standard Libraries
  - Monad Transformers
  - Separation Model
    - Memory
    - Algorith
    - RegFile
    - ChipMo

Edit
Browse Names
Browse Types
Browse Classes
Report
Read-Eval-Print
Run main
Compile
Debug
Properties

```
/** Implementation of a simple memory ADT.
 *
 *  @author  Programatica Team
 *  @version May 23, 2000
 */
module Memory where

import Addr
import Word

act type Memory = Addr -> Word

e empty memory is initialized to all zeros:
        :: Memory
         = \addr -> int2word 0

ading from memory is just function application:
        :: Memory -> Addr -> Word
```

File	Line	Description
.hs	23	Non-exhaustive pattern match
Memory.hs	24	Certificate expired

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

Programatica Demo
- Standard Pre
  - PreludeC
  - PreludeN
  - PreludeL
  - PreludeT
  - PreludeI
- Standard Lib
- Monad Trans
- Separation M
  - Memory
  - Algorithm
  - RegFile
  - ChipMod

/** Implementation of a simple memory ADT.

Programatica Session [Prelude]

Session Edit View Window Help

```
Prelude> sum
65
(267 reducti
Prelude> n *
55.0
(28 reductio
Prelude> :set
Prelude> f <-
Prelude> writ
Prelude> clos
Prelude> _
```

56473 cells allocat

Programatica Session [Memory]

Session Edit View Window Help

```
empty
(1 name listed)
Memory> read (write empty 0 1) 1
0
Memory> read (write empty 0 1) 0
1
Memory> read (write (write 0 23 empty) 0 1) 0
ERROR: Type error in application
*** Expression     : write 0 23 empty
*** Term           : empty
*** Type           : Memory
*** Does not match : Int

Memory> read (write (write empty 0 23) 0 1) 0
1
Memory> _
```

14343 cells allocated, 45654 reductions

- In it’s time, Programatica was the most sophisticated program development environment on the market;

- “It scares me to think that we nearly ended up in a world dominated by Java technology ... Programatica was a godsend; we couldn’t have made the transition to Haskell without it ...”

James Gosling, Microsoft CEO, Wall Street Journal
March 2007

- Back then:
  - Priorities: Time to market, raw functionality, ...
  - Non-issues: reliability, security, robustness, accountability, ...
  - Formal methods: academic toys, expensive, won't scale, irrelevant, ...

- Programatica was part of the new wave:

  "Programming as if correctness mattered"

The Programatica Vision:

- Build a program development environment that supports and encourages its users in thinking about, stating, and validating key properties.
- Enable programming and validation to proceed hand in hand, using properties to link the two.
- Allow users to realize benefits gradually by choosing between varying levels of assurance.

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- Programatica Demo
  - Standard Prelude
  - Standard Libraries
  - Monad Transformers
  - Separation Model
    - Memory
    - Algorithms
    - RegFile

```
-- reading from memory is just function application:
read          :: Memory -> Addr -> Word
read mem addr = mem addr

property ReadEmpty = All (a::Addr).
                         read empty a == 0

                                    ion extension:
                                    dr -> Word -> Memory

                                    addr==addr' then val else mem addr'

property ReadWrite = All (a::Addr).
                         All (w::Word).
                          All (m::Memory).
                           read (write m a w) a == w

property WriteWrite = All (a::Addr).
                          All (w,w' :: Word).
                           All  (m::Memory).
                            write (write m a w') a w = write m a w
```

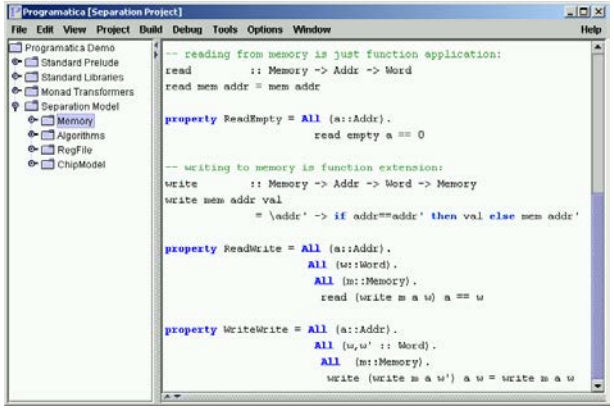
Property statements as an integral part of source code.

The Language of Properties:

- Properties expressed using:
  - Standard logical constructs and primitives;
  - The same syntactic conventions as executable code.

- In short, a property notation that was immediately familiar to programmers:

```
property ReadWrite
  = All a v m.
      read a (write a v m) === v
```



User supplied, domain-specific toolsets...

Type checking

Random test generator

Instrumenting compiler

Interactive proof editor

Model checker

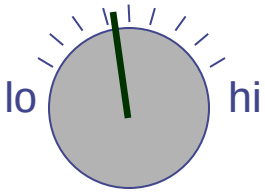
Prosper

HOL 2001

Isabelle

Prover

The “Knob”



Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- Programatica Demo
  - Standard Prelude
  - Standard Libraries
  - Monad Transformers
  - Separation Model
    - Memory
      - Types
      - Values
      - Properties
    - Algorithms
    - RegFile
    - ChipModel

```
-- reading from memory is just function application:
read          :: Memory -> Addr -> Word
read mem addr = mem addr

property ReadEmpty = All a::Addr.
                       read empty a == 0

-- w
writ
writ                                                    addr'

property ReadWrite = All (a::Addr).
                     All (w::Word).
                      All (m::Memory).
                       read (write m a w) a == w

property WriteWrite = All (a::Addr).
                      All (w,w' :: Word).
                       All  (m::Memory).
                        write (write m a w') a w = write m a w
```

These property statements have been annotated with certificates...

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- Programatica Demo
  - Standard Prelude
  - Standard Libraries
  - Monad Transformers
  - Separation Model
    - Memory
      - Types
      - Values
      - Properties
    - Algorithms
    - RegFile
    - ChipModel

```
-- reading from memory is just function application:
read          :: Memory -> Addr -> Word
read mem addr = mem addr

property ReadEmpty = All a::Addr.
                       read empty a

-- writing to memory is function ext
write         :: Memory -> Addr -> Wo
write mem addr val
              = \addr' -> if addr==a          lse mem addr'

property ReadWrite = All (a::Addr).
                      All (w::Word).
                       All (m::Memory).
                        read (write m a w) a == w

property WriteWrite = All (a::Addr).
                       All (w,w' :: Word).
                        All  (m::Memory).
                         write (write m a w') a w = write m a w
```

Edit Certificate
Update
Copy
Cut
Delete
Properties

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- Programatica Demo
  - Standard Prelude
  - Standard Libraries
  - Monad Transformers
  - Separation Model
    - Memory
      - Types
      - Values
      - Properties
    - Algorithms
    - RegFile
    - ChipModel

```
-- reading from memory is just function application:
read          :: M
read mem addr = m

property ReadEmpt


-- writing to mem
write         :: M
write mem addr va
              = \                                  m addr'

property ReadWrit




property WriteWri
```

Certificate Viewer

Summary | Dependencies | Tools

Type:	Jape editor session
Author:	Mark P Jones (mpj)
Signature:	none given
Status:	Incomplete
Created:	Mar 23, 2001 3:23:12 PM
Modified:	Mar 28, 2001 6:32:15 AM
Accessed:	Dec 25, 2006 11:16:02 AM

OK | Cancel | Apply

```
write (write m a w') a w = write m a w
```

Certificates:

- Certificates were “embedded objects” in source documents.
- Certificates were <u>not</u> part of the language:
  - They were not named or typed;
  - They were not propagated between modules.
- Programatica could be configured to support many different certificate types.
- They were queried and invoked through a generic interface/API: the “validation bus”.

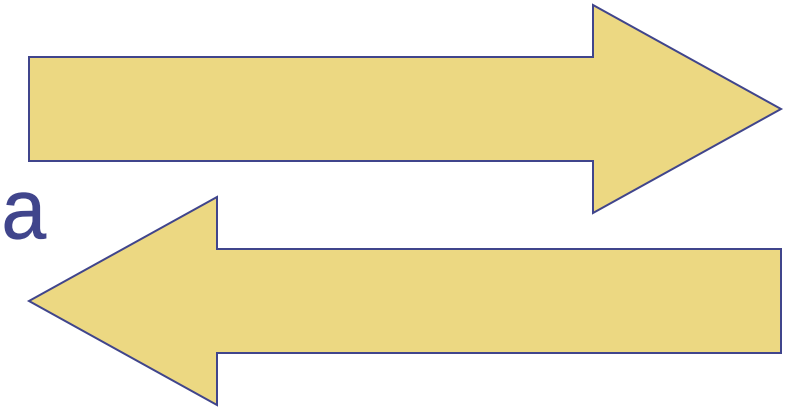
The Validation Bus:

Property translation and subsetting

Property propagation and theory formation

Query and invocation mechanisms

Programatica



External Tools

Status reporting and auditing

Embedded display and editing

Encapsulated sessions and state

Property Management:

- Programatica's <u>property management</u> facilities provided the link between property statements and certificates;

- Programatica supported:
  - *Pay-as-you-go*: Zero or more certificates for each property;
  - *Mix-and-match*: Different types of certificate could be used together in any given program.

- The tools helped users (and their managers) to understand the extent to which properties had been validated: "where is the knob"?

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- Programatica Demo
  - Standard Prelude
  - Standard Libraries
  - Monad Transformers
  - Separation Model
    - Memory
      - Types
      - Values
      - Properties
        - ✓ ReadEmpty
        - O ReadWrite
        - ! WriteWrite
    - Algorithms
    - RegFile
    - ChipModel

```
-- reading from memory is just function application:
read          :: Memory -> Addr -> Word
read mem addr = mem addr
```

Key:

✓ At least one “valid” certificate;

! Certificate invalid, or proof incomplete;

O No certificates provided;

P Parameterized property.

```
All (w,w' :: Word).
All (m::Memory).
 write (write m a w') a w = write m a w
```

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- gramatica Demo
- Standard Prelude
- Standard Libraries
- Monad Transformers
- Separation Model
  - Memory
    - Types
    - Values
    - Properties
      - ReadEmpty
        - Jape: proof comple
      - ReadWrite
      - WriteWrite
  - Algorithms
  - RegFile
  - ChipModel

```
-- the empty memory is initialized to all zeros:
empty          :: Memory
empty           = \addr -> int2word 0

-- reading from memory is just function application:
read           :: Memory -> Addr -> Word
read mem addr = mem addr

property ReadEmpty = All a::Addr.
                        read empty a == 0

-- writing to memory is function extension:
write          :: Memory -> Addr -> Word -> Memory
write mem addr val
                = \addr' -> if addr==addr' then val else mem addr'

property ReadWrite = All (a::Addr).
                       All (w::Word).
                        All (m::Memory).
                         read (write m a w) a == w
```

Programatica [Separation Project]

File Edit View Project Build Debug Tools Options Window Help

- gramatica Demo
- Standard Prelude
- Standard Libraries
- Monad Transformers
- Separation Model
  - Memory
    - Types
    - Values
    - Properties
      - ReadEmpty
        - Jape: proof comple
      - ReadWrite
      - WriteWrite
        - SMV: found counter
        - Prosper.HOL: old c
  - Algorithms
  - RegFile
  - ChipModel

```
-- reading from memory is just function application:
read          :: Memory -> Addr -> Word
read mem addr = mem addr

property ReadEmpty = All a::Addr.
                        read empty a == 0

-- writing to memory is function extension:
write         :: Memory -> Addr -> Word -> Memory
write mem addr val
              = \addr' -> if addr==addr' then val else mem addr'

property ReadWrite = All (a::Addr).
                       All (w::Word).
                        All (m::Memory).
                         read (write m a w) a == w

property WriteWrite = All (a::Addr).
                        All (w,w' :: Word).
                         All  (m::Memory).
                          write (write m a w') a w = write m a w
```

```
*** STOP: 0x000000A (0x00000000,0x00000002,0x00000000,8038c240)
IRQL_NOT_LESS_OR_EQUAL*** Address 8038c240 has base at 8038c000 - Ntfs.SYS

CPUID:Genuine Intel 6.3.3 irql:1f SYSVER 0xf0000565

Dll Base DateStmp - Name                 Dll Base DateStmp - Name
80100000 336546bf - ntoskrnl.exe         80010000 33247f88 - hal.dll
80000100 334d3a53 - atapi.sys            80007000 33248043 - SCSIPORT.SYS
802aa000 33013e6b - epst.mpd             802b5000 336016a2 - Disk.sys
802b9000 336015af - CLASS2.SYS           8038c000 3356d637 - Ntfs.sys
802bd000 33d844be - Siwvid.sys           803e4000 33d84553 - NTice.sys
f9318000 31ec6c8d - Floppy.SYS           f95c9000 31ec6c99 - Null.SYS
f9468000 31ed868b - KSecDD.SYS           f95ca000 335e60cf - Beep.SYS
f9358000 335                                                        .sys
f947c000 31e                                                        .SYS
f9370000 332
f9490000 31e
f90f0000 332
a0000000 335
fe0c9000 335                                                        SYS
fe108000 31e                                                        YS
f9050000 332

Address  dword dump Build [1314]                                - Name
801afc24 80149905 80149905 ff8e6b8c 80129c2c ff8e6b94 8025c000 - Ntfs.SYS
801afc2c 80129c2c 80129c2c ff8e6b94 00000000 ff8e6b94 80100000 - ntoskrnl.exe
801afc34 801240f2 80124f02 ff8e6df4 ff8e6f60 ff8e6c58 80100000 - ntoskrnl.exe
801afc54 80124f16 80124f16 ff8e6f60 ff8e6c3c 8015ac7e 80100000 - ntoskrnl.exe
801afc64 8015ac7e 8015ac7e ff8e6df4 ff8e6f60 ff8e6c58 80100000 - ntoskrnl.exe
801afc70 80129bda 80129bda 00000000 80088000 80106fc0 80100000 - ntoskrnl.exe

Restart and set the recovery options in the system control panel
or the /CRASHDEBUG system start option. If this message reappears,
contact your system administrator or technical support group.
```

Time Machine

Rebooting into the Year 2001

Please wait a moment ...

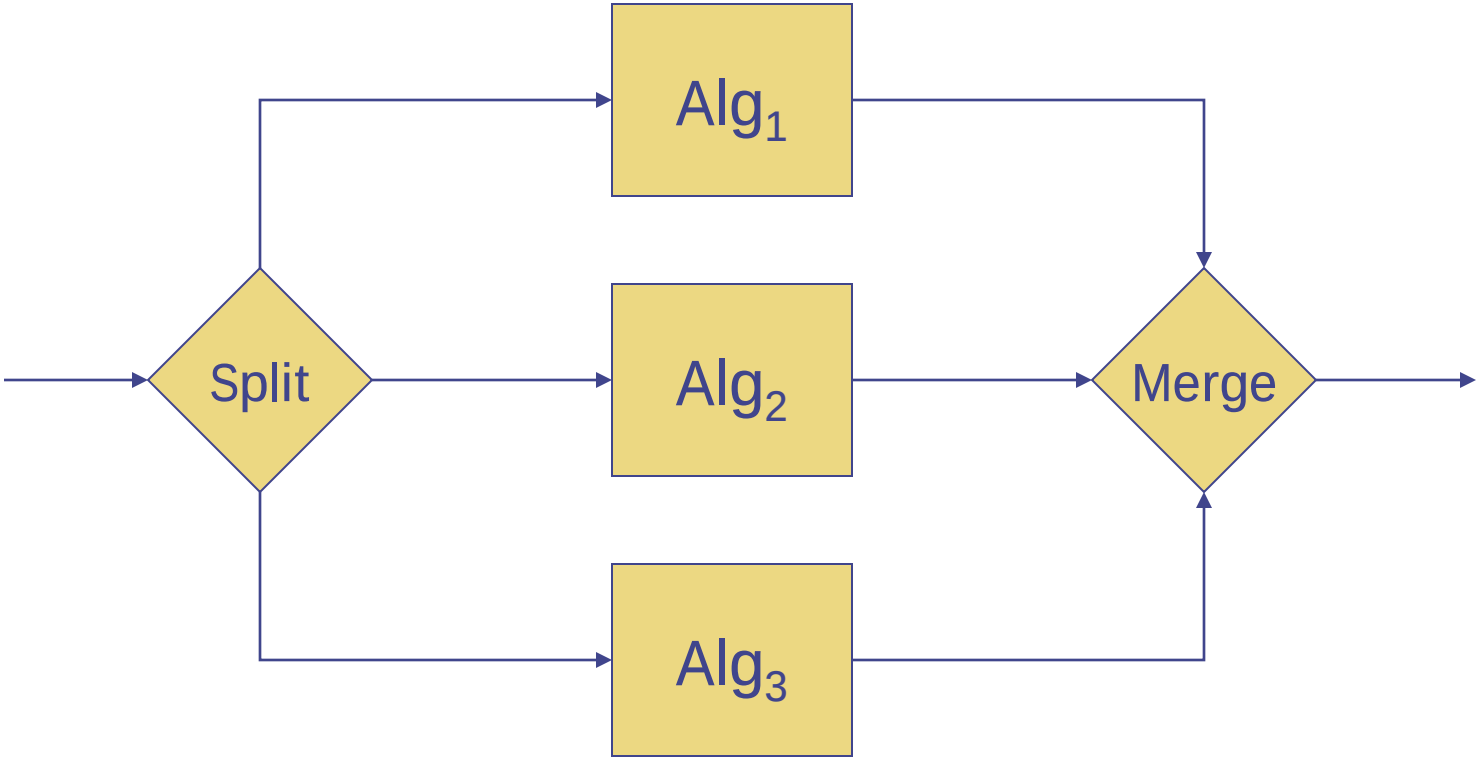
The Real Programatica:

- There really is a Programatica project at OGI ...
- The team includes: Jim Hook (PI), Mark Jones, Dick Kieburtz, John Launchbury, Tim Sheard, Peter White, Bill Harrison, and Andy Moran.
- Peter White is also our first “Customer”.

The Road to Programatica:

- We are currently building Programatica, version 1
- It will look quite different to the mockups I showed earlier ...
- But the basic vision and concepts are the same!
- Our design & development efforts are informed by ongoing experiments to help us understand how we will use the Programatica tools in practice ...

Example: Modeling a Crypto-Chip



- One chip, multiple channels;
- Channels may use different algorithms;
- Separation of channels GUARANTEED.

High-level view:

Map channels to algorithms

```
chip :: (Channel→ Alg)→
          [Packet]→ [Packet]
```

Packet Filter

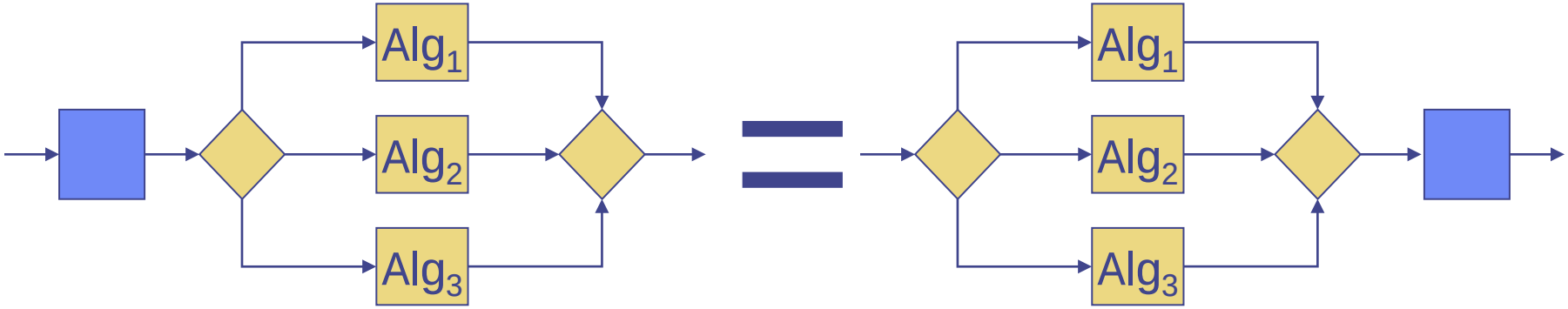
```
type Packet = (Channel, Data)
```

Channel Id

Payload

Separation of Channels:

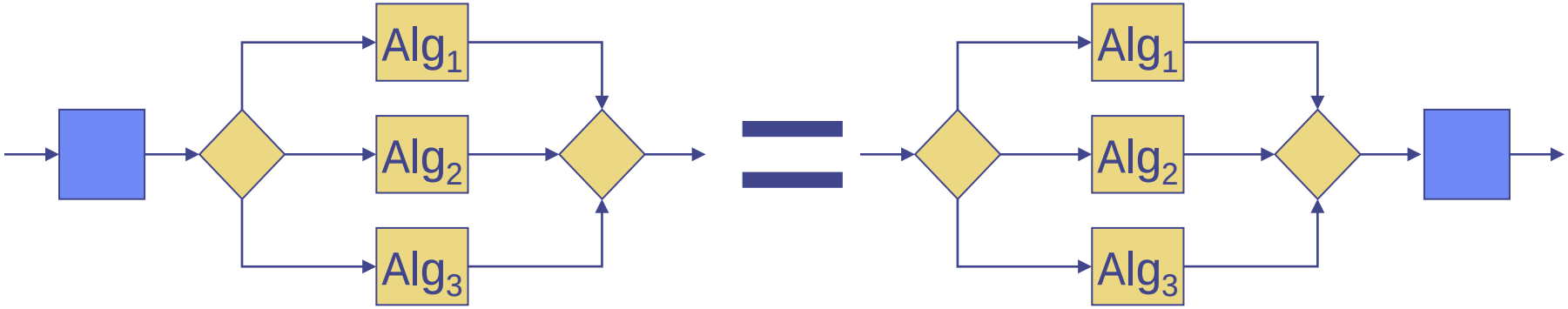
```
All (algs :: (Channel→Alg)).
 All (select :: (Channel→Bool)).
  All (ps :: [Packet]).
   filter (select . fst) (chip algs ps)
       ==
   chip algs (filter (select . fst) ps)
```



Separation of Channels:

This law guarantees that:

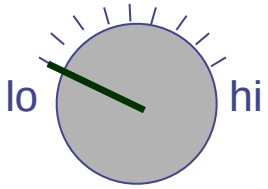
- Outputs do not depend on inputs to other channels.
- Channels do not generate spurious outputs.



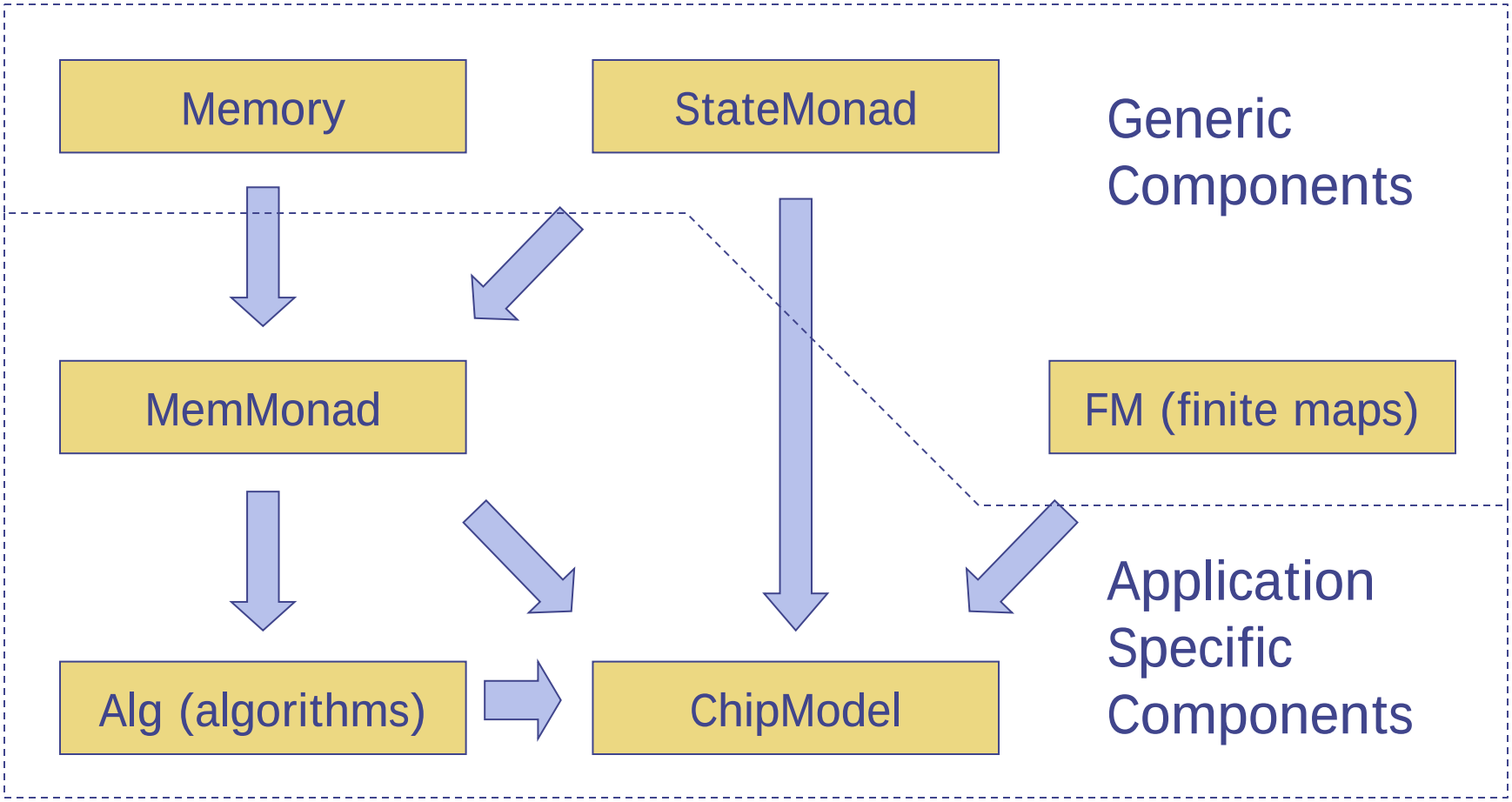
Putting Programatica to Work:

- Our goal is to build tools that will help to establish and automate validation of properties like this.
- We have described the non-interference property at a high-level;
- But we want to model the `chip` at a level that is closer to its implementation on silicon.

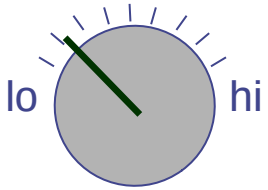
Building the Model (1):



We developed an executable model of the `chip` as a short Haskell program:



Building the Model (2):



We annotated the model with properties ...

... and quickly spotted bugs in our code!

Just writing properties had heightened our thinking about correctness ...

Properties of off-the-shelf components

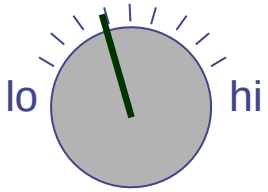
↓

Properties of internal components

↓

Properties that guarantee more secure and reliable software

Building the Model (3):



We built a quick prototype for parsing and type checking properties ...

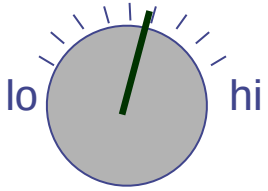
```
property AllocMem
   = forall ws m.
        let (m', r) = allocMem ws m
        in forall a.
              if (r `includes` a)
                 then readMem a m' == ws!!(a-l) &&
                      readMem a m  == readMem a initMemory
                 else readMem a m' == readMem a m
```

Arguments in the wrong order!

Undefined symbols!

... and immediately found bugs in our specifications!

Building the Model (4):

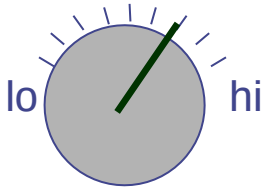


We recast the channel separation property in an imperative style using monads.

A serious bug was uncovered, the result of failing to zero temporary storage after each packet (or of using absolute addresses...)

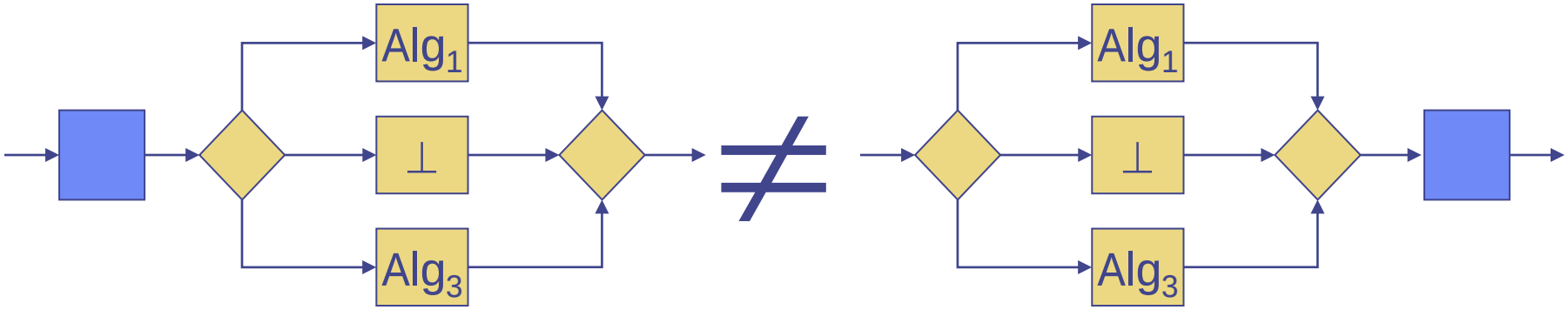
Bug detection <u>and</u> feedback to the designers!

Building the Model (5):

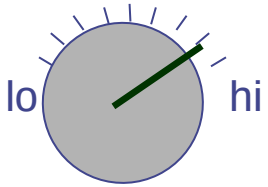


We proved channel separation by hand (and type checked the proof by machine):

- New insights into the pragmatics of designing and using a suitable logic;
- Details of interpretation must be pinned down (e.g., sets or pointed domains?) ...



Building the Model (6):



We are formalizing the model in HOL and redoing the proof in this setting:

- To obtain more rigorous proofs (and debugged code!) for channel separation;
- To develop techniques for automating the translation into HOL;
- To determine conditions under which specs can be faithfully embedded in HOL.

Key Points:

- A new kind of program development environment that encourages thinking about program correctness.
- A flexible and expressive notation for modeling, and for rapid prototyping. (Executable models!)
- Properties can be used to state key properties of software systems. Certificates can be used to attach supporting evidence of validity.
- Writing properties is easy, and proceeds hand in hand with programming.
- The quality of validation can be increased as higher levels of assurance are required:
  - From type checking ...
  - ... through automated test case generation ...
  - ... to full-blown theorem proving.