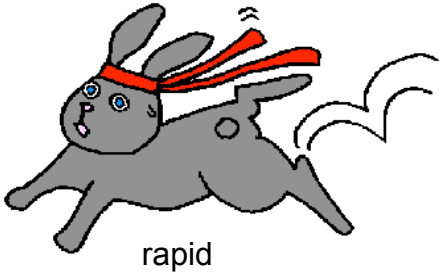
Rewriting Logic and The Maude Execution Environment

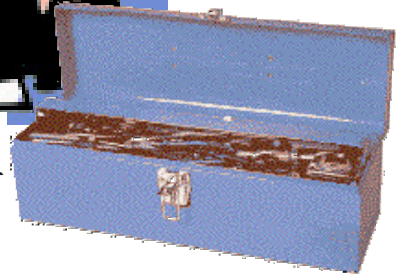
http://maude.cs.uiuc.edu

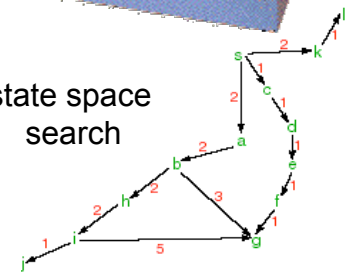
Carolyn Talcott
SRI International

Maude Formal Methodology



rapid prototyping

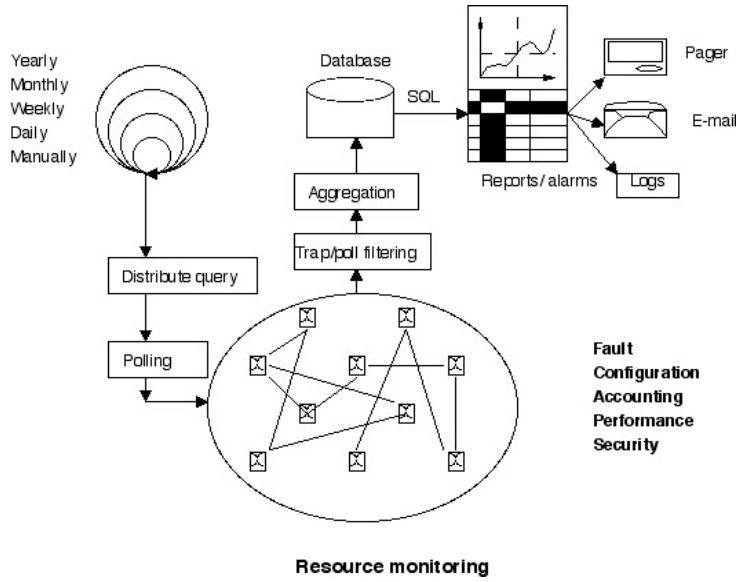




state space search

$$S \models \Phi$$

model checking



Resource monitoring

Plan

Big Picture

Simple examples in some detail

Highlights of some applications and case studies

Rewriting Logic

Q: What is rewriting logic?

A1: A logic to specify, reason about, prototype, and program software systems, that may possibly be concurrent, distributed, or even mobile.

A2: An extension of equational logic with local rewrite rules to express concurrent change over time.

Rewriting Logic as a Semantic Framework

A wide variety of models of computation can be naturally expressed as rewrite theories

- Lambda Calculus, Turing Machines
- Concurrent Objects and Actors
- CCS, CSP, pi-calculus, Petri Nets
- Chemical Abstract Machine, Unity
- Graph rewriting, Dataflow, Neural Networks
- Real-time Systems
- Physical Systems (Biological processes)

Rewriting Logic as a Logical Framework

A wide variety of logics have a natural representation as rewrite theories

- Equational logic
- Horn logic
- Linear logic
- Quantifiers
- Higher-order logics (HOL, NuPrl)
- Open Calculus of Constructions
- Rewriting logic !

Rewriting Logic is Reflective

A reflective logic is a logic in which important aspects of its metatheory (entailment relation, theories, proofs) can be represented at the object level in a consistent way.

This has many applications:

- Metaprogramming
- Module composition
- Reification of maps of logics
- Internal strategies
- Higher-order capabilities in a first-order framework
- Formalization of reflection for concurrent objects
- Domain specific assistants

Simple Examples

Rewrite Theories

- Rewrite theory: (Signature, Labels, Rules)
- Signature: (Sorts, Ops, Eqns) -- an equational theory
  - Describe data types, system state
- Rules have the form label : t => t' if cond
- Rewriting operates modulo equations
  - Describes computations or deductions, also modulo equations

Maude

- Maude is a language and environment based on rewriting logic
- See: http://maude.cs.uiuc.edu
- Features:
  - Executability -- position /rule/object fair rewriting
  - High performance engine --- {ACI} matching
  - Modularity and parameterization
  - Builtins -- booleans, number hierarchy, strings
  - Reflection -- using descent and ascent functions
  - Search and model-checking

Example: NatList

```
fmod NATLIST is
  pr NAT .
  sort NatList .
  subsort Nat < NatList .
  op nil : -> NatList .
  op __ : NatList NatList -> NatList [assoc id: nil] .

  var n: Nat. var nl: NatList .

  op sum : NatList -> Nat .
  eq sum(nil) = 0 .
  eq sum(n nl) = n + sum(nl) .
endfm

Maude> reduce  sum(1 2 3) .
result NzNat: 6
```

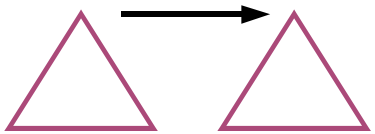
Example: NatList

```
fmod NATLIST1 is
  inc NATLIST .
  var n : Nat .
  vars nl nl' nl'' : NatList .

  op removeDups : NatList -> NatList .
  eq removeDups( nl n nl' n nl'')
        =  removeDups(nl n nl' nl'') .
  eq removeDups(nl) = nl [owise] .
endfm
```

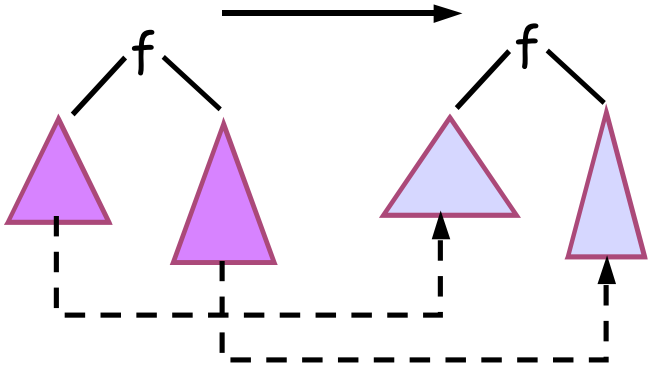
```
Maude> reduce removeDups(0 3 0 1 2 1 0) .
result NatList: 0 3 1 2
```

Deduction/Computation Rules

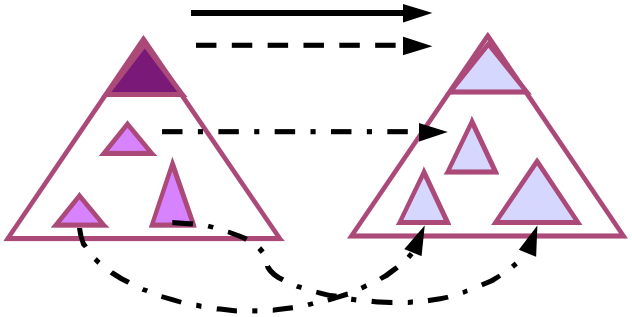
reflexivity:



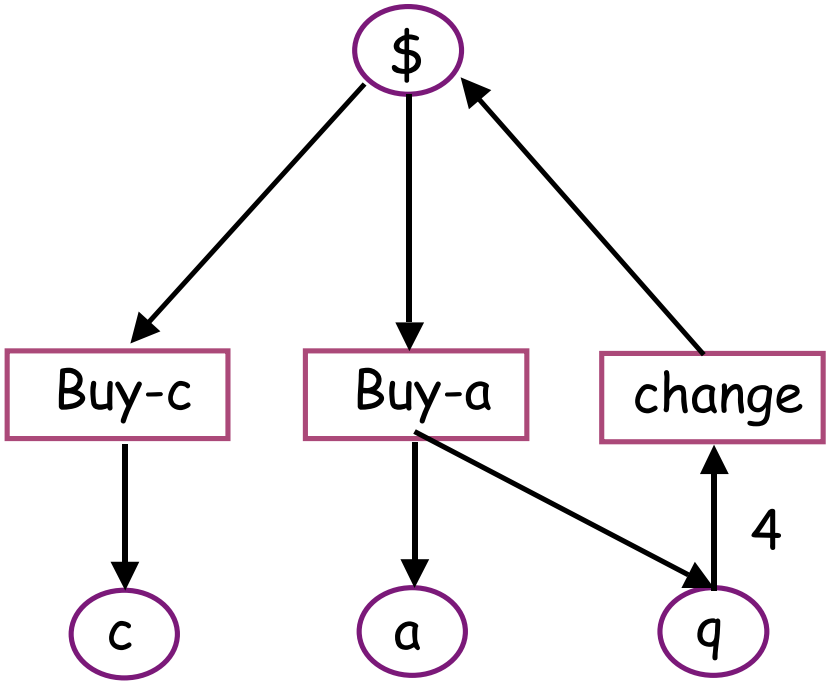
congruence:



replacement:



Petri Net Example
A vending machine



The vending machine as a rewrite theory

A Maude Program

```
mod VENDING-MACHINE is
  sort Place Marking .
  subsort Place < Marking .
  op 1 : -> Marking .          *** empty marking
  ops $,q,a,c : -> Place .
  op _ _ : Marking Marking -> Marking
                  [assoc comm id: 1] .
  rl[buy-c]: $ => c .
  rl[buy-a]: $ => a q .
  rl[change]: q q q q => $ .
endm
```

Using the vending machine

```
Maude> rew $ $ $ .
result Marking: q a c c

Maude> search $ $ $ =>! a a M:Marking .

Solution 1 (state 8)
M:Marking --> q q c

Solution 2 (state 9)
M:Marking --> q q q a

No more solutions.
states: 10  rewrites: 12)
```

Reflection example: Module analysis

```
fmod CONSUMERS is
  inc MY-META .
  var M : Module .    var T : Term .
  var R : Rule .    var RS : RuleSet .

  op consumes : Module Rule Term -> Bool .
  eq consumes(M,R,T)  =
    getTerm(metaReduce(M,'has[getLhs(R),T]))
        == 'true.Bool .

  op consumerRules : Module Term -> QidSet .
  op consumerRules : Module RuleSet Term -> QidSet .

  eq consumerRules(M,T) =
       consumerRules(M,upRls(getName(M),true),T) .
  eq consumerRules(M,none,T) = none .
  eq consumerRules(M,R RS,T) =
   (if consumes(M,R,T) then getRuleId(R) else none fi);
       consumerRules(M,RS,T) .
endfm
```

Reflection Example: Module analysis cntd.

```
mod VEND-X is
  inc VENDING-MACHINE .

  vars M0 M1 : Marking .
  op has : Marking Marking -> Bool .

  eq has(M0 M1, M1) = true .
  eq has(M0, M1) = false [owise] .
endm

select CONSUMERS .

Maude>  red consumerRules(['VEND-X],'$.Coin) .
result: Sort 'buy-a ; buy-c

Maude>  red consumerRules(['VEND-X],'q.Coin) .
result: Sort 'change
```

Reflection example: Strategy

```
fmod METAREWRITE-LIST is
  inc MY-META .
  var M : Module .
  vars T T' : Term .
  var res : Result4Tuple? .
  var rid : Qid .
  var ql : QidList .

  op metaRewList : Module QidList Term -> Term .
  eq metaRewList(M,nil,T) = T .
  ceq metaRewList(M,rid ql,T) = metaRewList(M,ql,T')
    if res := metaXapply(M,T,rid,none,0,unbounded,0)
    /\ T' := if res :: Result4Tuple
             then getTerm(res) else T fi .
endfm
```

Reflection example: Strategy cntd.

```
Maude> red metaRewList(['VENDING-MACHINE],
            'change 'buy-a,
            '__['q.Coin,'q.Coin,'q.Coin,'q.Coin]) .

result GroundTerm: '__['q.Coin,'a.Item]


Maude> red metaRewList(['VENDING-MACHINE],
             'buy-a 'change,
             '__['q.Coin,'q.Coin,'q.Coin,'q.Coin]) .

result Constant: '$.Coin
```

Experience Using Maude

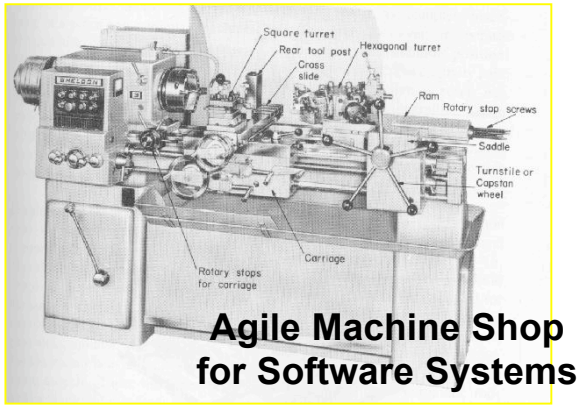
A Tool To Build Tools

Maude interpreters for ...

- PLAN, an active network language
- D'Agents, a mobile agent language
- GAEA, a mobile agent language

Notations and analysis tools

- Object-oriented
- Real-time Maude

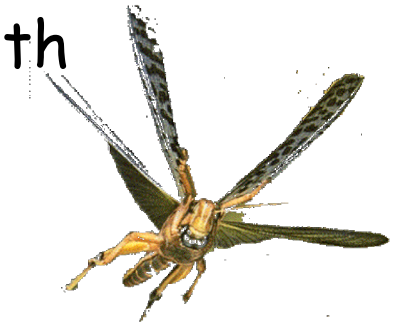


And Maude mappings from ...

- UML → Maude (Universal Modeling Language)
- CAPSL → CIL (cryptographic analysis)
  - Common Authentication Protocol Specification Language (CAPSL) provides interoperability for many tools used in the analysis of computer security protocols
- HOL → NuPrl (Sharing Theory Libraries)

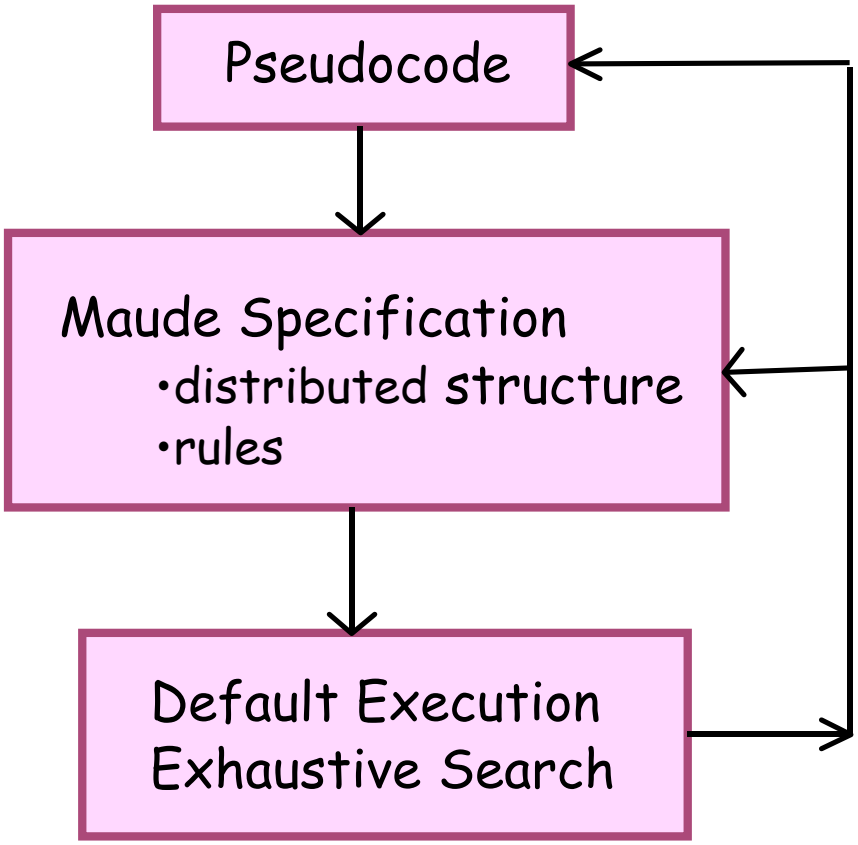
Maude Finds Insidious Bugs in Complex Systems

- A new active network broadcast protocol with dynamic topology (UCSC, 1999)



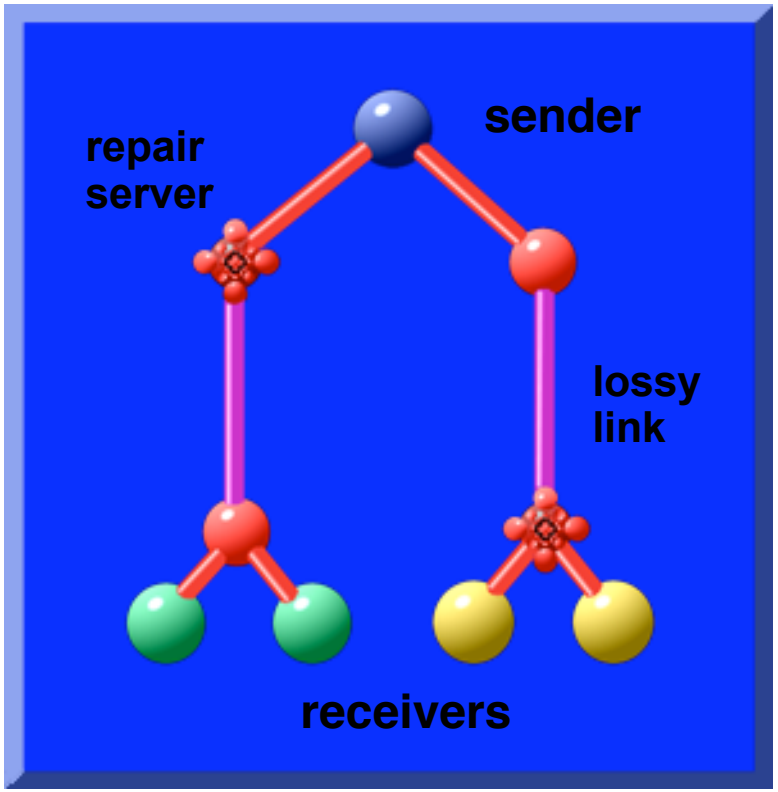
- AER/NCA suite for active network protocols (UMass /TASC, 2000)

RBP
Designing a Reliable Broadcast Protocol



- make assumptions explicit
- discover gaps, missing cases
- repair problems early in design phase
- discover subtle bugs related to concurrency and distribution

Application to Reliable Multicast in Active Networks



- Faithful representation:
  - Network nodes and links
  - Capacity, congestion, etc.
  - Represented in Maude
- Efficient automated analysis
- Uncovered important and subtle bugs not known to network engineers

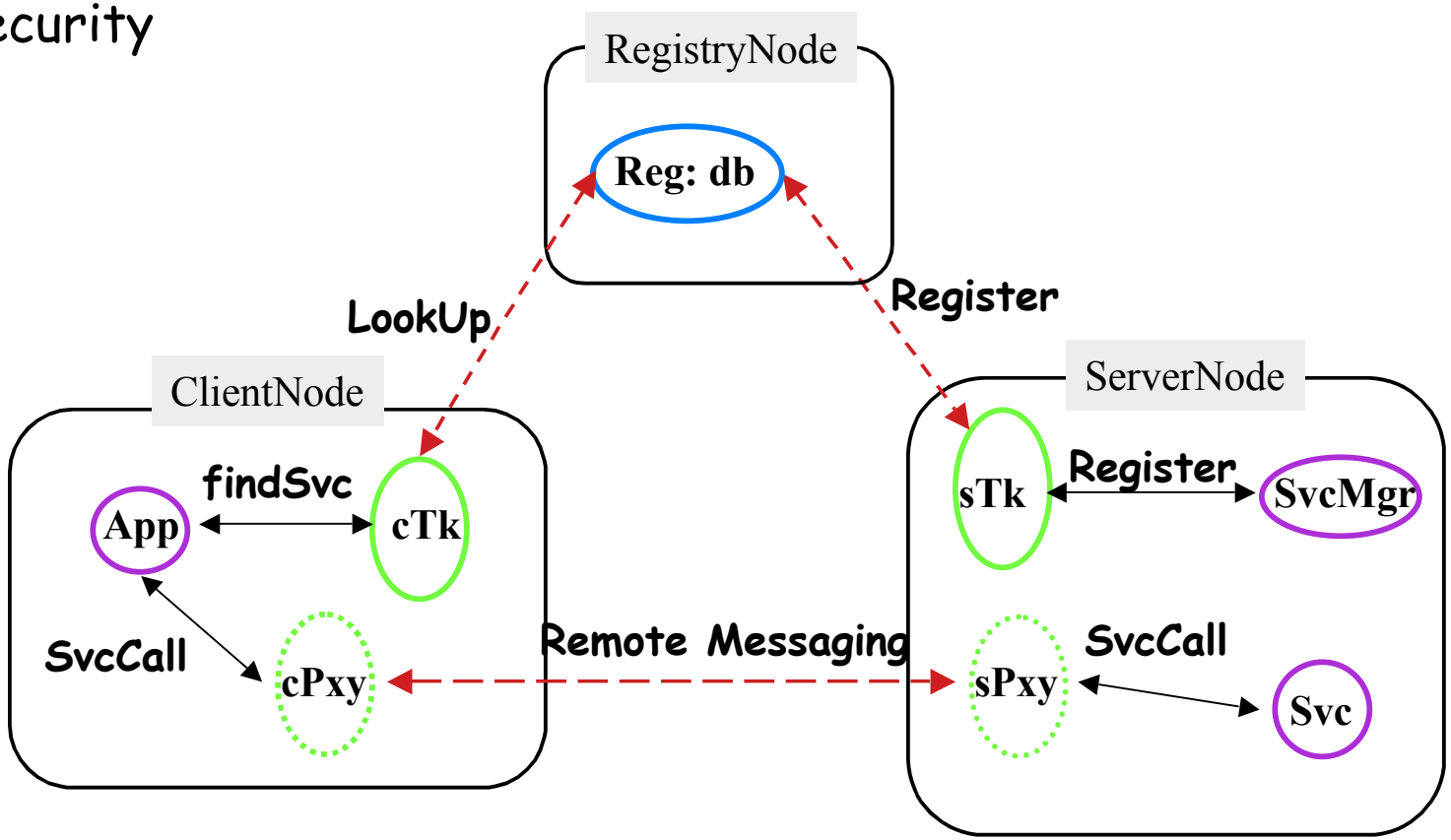
PANAMA

Tasc/UMass

Service Proxy Toolkit (SPTK)

Remote service requirements:

- Publication and discovery
- Remote messaging
- Security



Security Goals

- Goal 0: Client VM protected from evil proxy
- Goal 1: Secure client-server communication
- Goal 2: Client can authenticate service proxy
- Goal 3: Server can also authenticate client

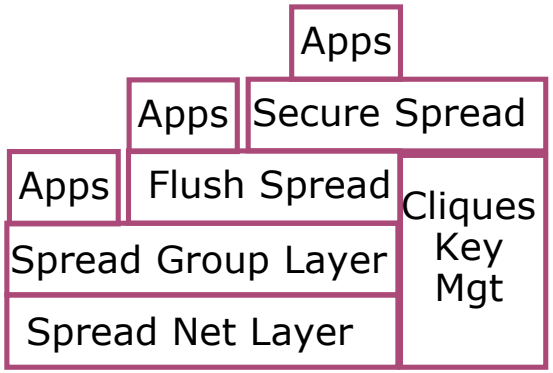
Maude Model of SPTK

- Documentation of SSPTK architecture
  - Modular, tunable security levels
- Formalization of security goals
- Security hole closed
  - signed proxy needs to include service description

http://www-formal.stanford.edu/clt/FTN

Secure Spread

- Spread is group communication system
  - Provides range of message delivery guarantees
    - reliable, fifo, causal, agreed, safe
  - in the presence of network partitions
- Secure spread adds group key management



Secure Spread in Maude

- Objective
  - Abstract executable specification of Secure Spread
  - Model each component and compose
  - Documentation for designers and user
  - Verification of Spread and applications built top of Spread
- Starting point
  - User Guide (informal, many details)
  - Research papers (high-level axiomatic semantics)
  - Spread Source Code (C)
- Modelling Challenges
  - Capture best-effort principle formally

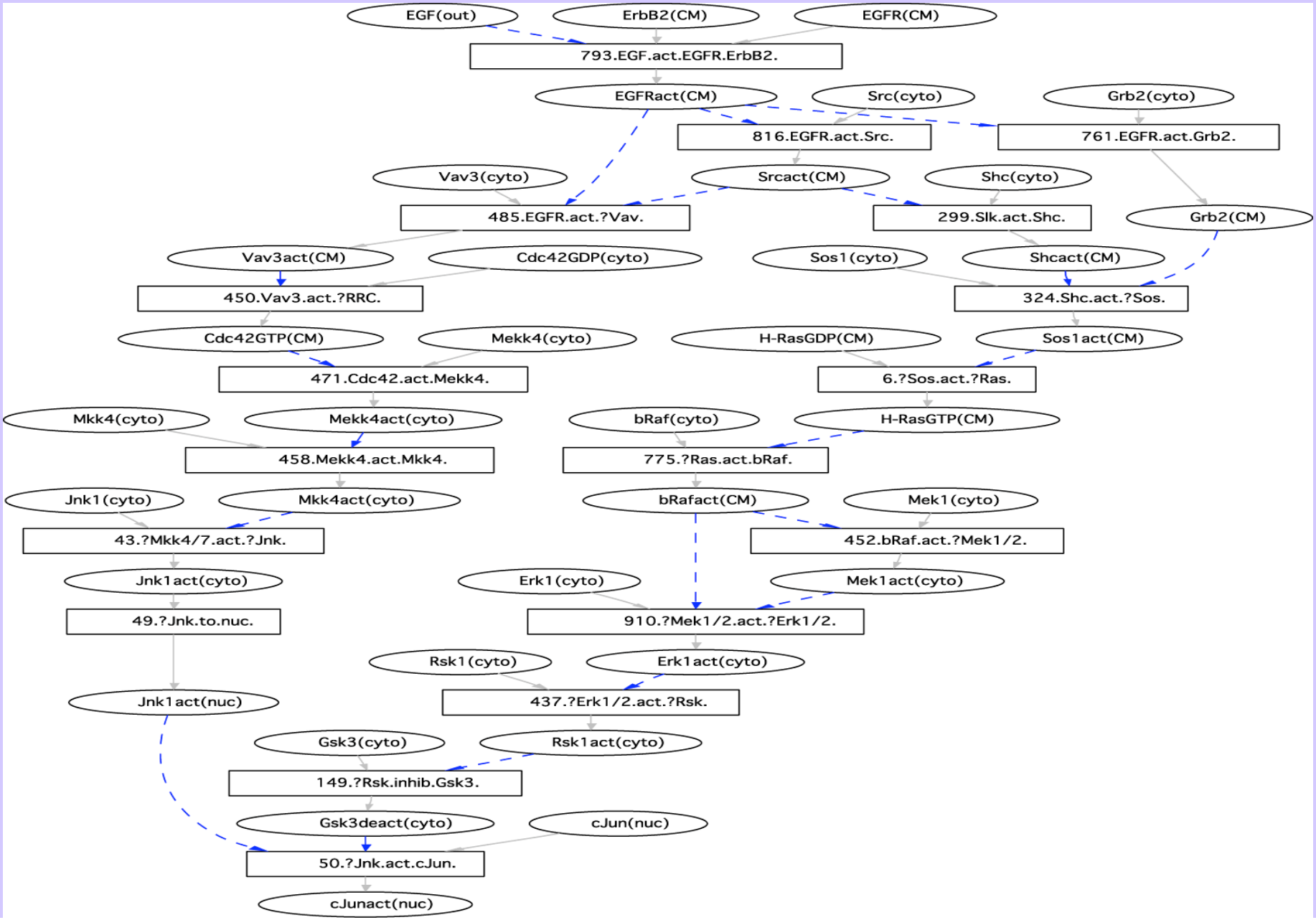
Secure Spread

- Basic tool for exploring alternative designs
- Formal API
- Mapped abstract GCS specification to event partial order semantics of Spread model
- Raised some subtle issues
- Adapting CAPSL specification of Secure Spread to glue Flush Spread and Cliques
- http://www-formal.stanford.edu/clt/FTN

Pathway Logic
Maude Models of Cellular Processes

- Biological entities are represented as terms
- Networks of processes/reactions are represented by collections of rewrite rules.
- The network models can be queried using formal methods tools.
  - Execution--find some pathway through the network
  - Search--find all pathways leading to a specified final condition
  - Model-checking--is there a pathway having particular properties?

Visualizing a EGFR Network as a PetriNet



EGF/EGFR experiments

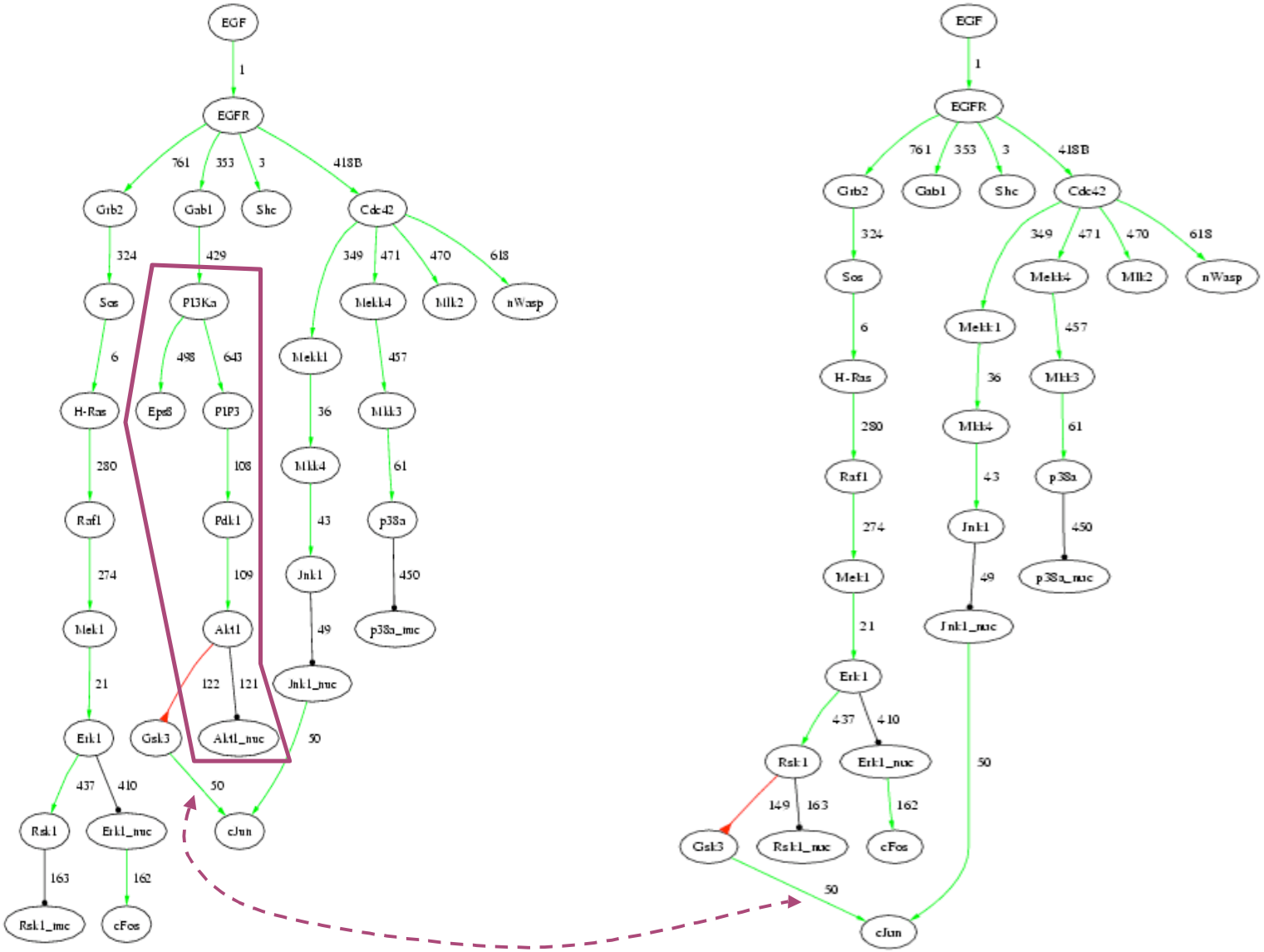
Activation of a transcription factor (cJun cFos) following binding of extracellular epidermal growth factor (EGF) to its receptor (EGFR)

```
ops q1 q1x : -> Dish .
eq q1 = PD(EGF {CM | EGFR Pak1 PIP2 nWasp [H-Ras - GDP]
                     {Akt1  Gab1 Grb2 Gsk3 Eps8 Erk1
                      Mek1 Mekk1 Mekk4 Mkk4 Mkk3 Mlk2
                      Jnk1 p38a p70s6k Pdk1 PI3Ka PKCb1
                      Raf1 Rsk1 Shc Sos [Cdc42 - GDP]
                      {NM | empty {cJun cFos }}}}) .

eq q1x = q1 - < PI3Ka , cyto >    *** knockout
```

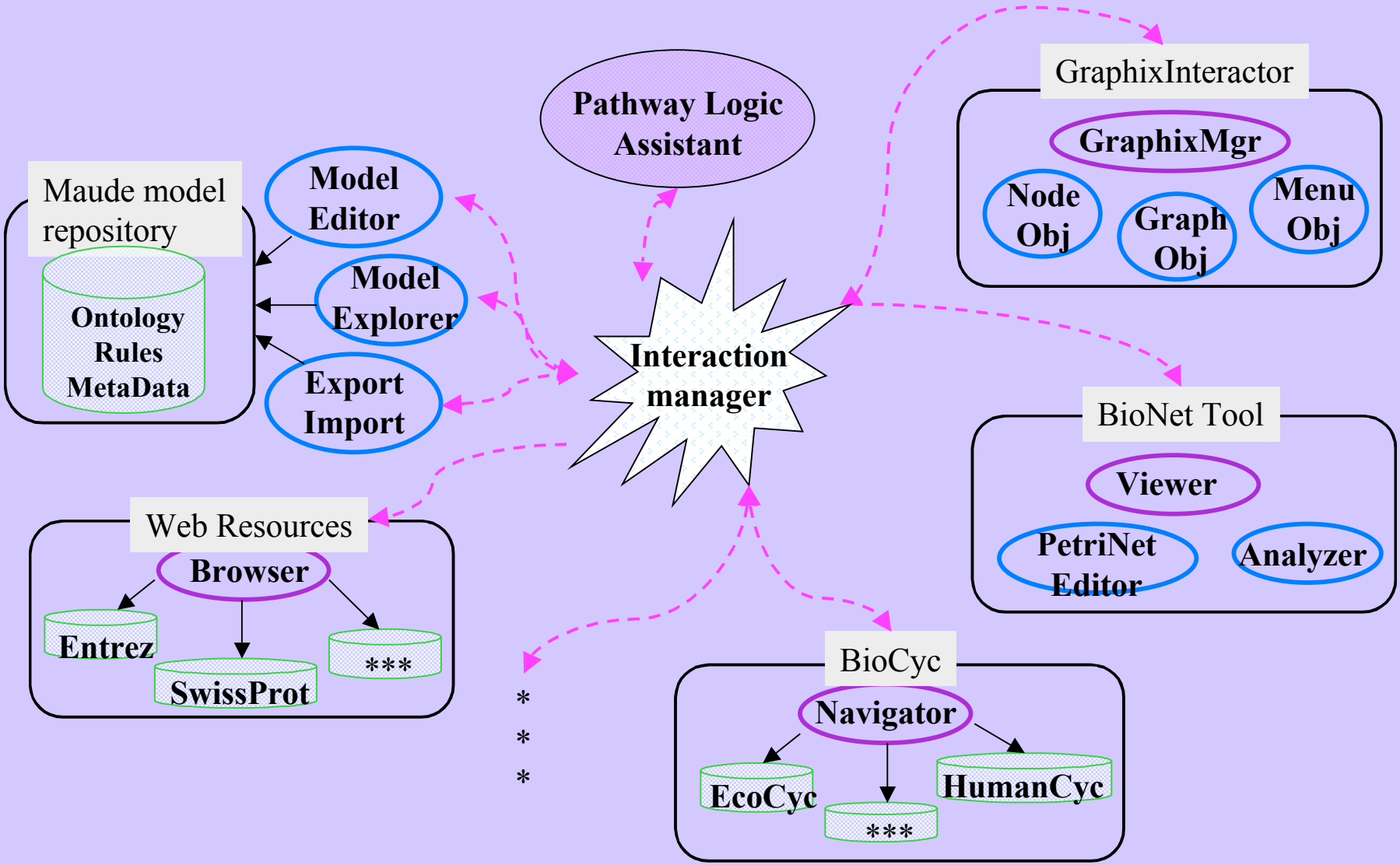
Model Checking

```
subsort  Dish < State .
eq PD(out:Soup
      {CM | cm:Soup
           {cyto:Soup
            {NM | nm:Soup
                  {nuc:Soup
                   [cJun - act] [cFos - act] }}}})
    |= prop1 = true .

eq findPath(S:State,P:Prop)
         = getPath(P:Prop, S:State |= ~ <> P:Prop) .

red  findPath(q1,prop1) .
red  findPath(q1x,prop1) .
```

Roadmaps for q1,q1x runs



Pathway Logic Workbench



Challenges for a Next Generation FM Framework

- Natural modeling of a wide range of features
- Combining and interoperating different models of a system, and/or models of subsystems
- Factoring models/analyses/code -- scale and reuse
- Transforming and abstracting models
- Analysis techniques and tools that require the right level of effort for the required level of assurance
- Promising candidate: rewriting logic and Maude

Coming Attractions

- Mobile Maude
- Probablistic and stochastic reasoning
- Animation and visualization capabilities
- More interoperation with other tools