Specifying the Plan Language in Maude

José Meseguer & Mark-Oliver Stehr, SRI International

Carolyn Talcott, Stanford University

Carl A. Gunter & Pankaj Kakkar, University of Pennsylvania

May 2000

PLAN -:)

- History – the FBAR (Flow-Based Adaptive Routing) specification in Maude
- Present – The PLAN language specification in Maude
  - Why
  - How
  - Example
  - Status/Results
- Future – Refining, evolving and using the Maude specification

History

- The first step was to specify the PLAN FBAR algorithm in Maude

- This went well and we were able to do some analysis: an instance verification for a small network

- But this was a *hand* translation of a PLAN program into Maude

Specifying the PLAN Language in Maude

What would be gained?

- close the gap between PLAN program and Maude analysis
- provide an executable specification and PLAN simulator
- provide a tool for experimenting with new language extensions or modifications

Who Benefits

- application programmers
  - clarifying the language constructs
  - simulation for testing and analysis, viz. resource measurement
- compiler programmers
  - a clear and flexible specification
  - a reference model for testing
- language designers
  - a working document

    where we've got to – rules for constructs
  - analysis, clear statements of properties

Developing the Maude Specification I

Preparation

- Listening to some talks about PLAN
- Reading
  - PLAN Standard Specification
  - Paper about the specification of PLAN
  - PLAN Programmers Guide and Tutorial
- Asking a few questions

Developing the Maude Specification II

- Problems with the current standard specification
  - not executable
  - network model has some problems
  - does not accurately model the concurrency
  - gaps, inconsistencies, and ambiguities typical of semi-formal (on paper) operational semantics

- Nevertheless, the PLAN documents (and some knowledge of semantics of functional languages and networks) were enough to produce a first Maude specification.

- Accomplished in about a week by 2 people working about half-time.

Design of the Maude Specification

- 3 level semantics
  - local steps
  - tasks
  - node services

- rules for local steps are purely functional

  – using standard techniques for operational semantics

- rules at the task level use resource information carried in the packet

  – factored out to simplify the rules

- rules at the node level use node state and implement PLAN services

Using Maude Features

- The network infrastructure is represented in Maude as a multiset of objects (PLAN nodes) and packets (in transit). A key challenge is to get the right level of abstraction here.

- Service packages modelled by subclassing nodes

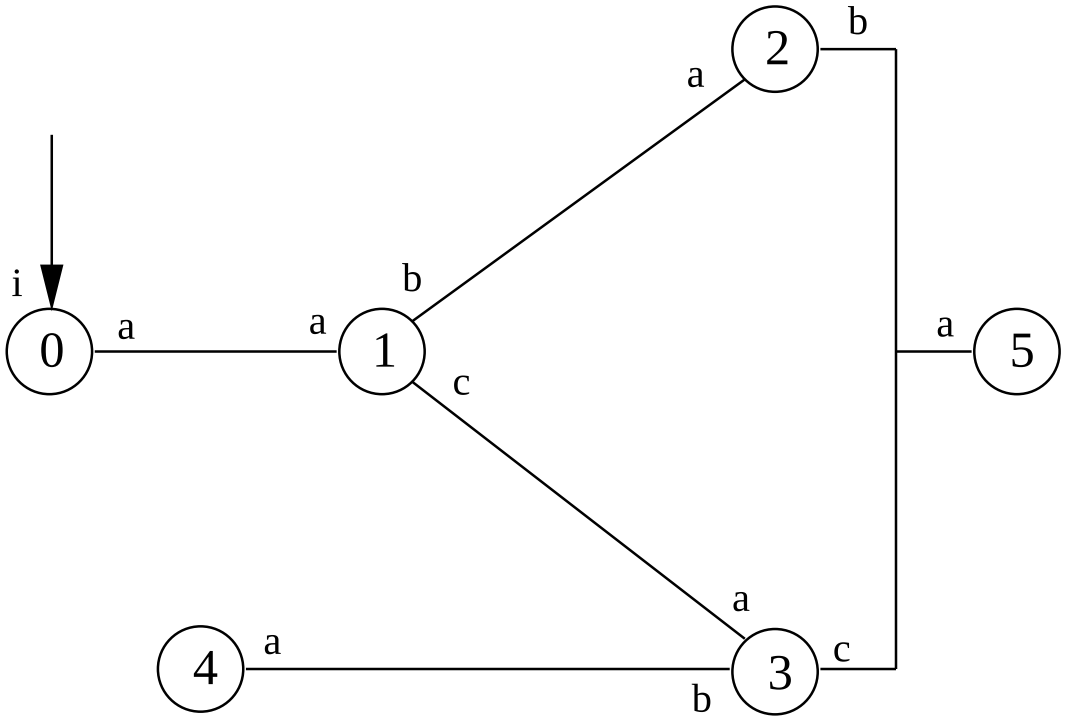
- concurrency modelled by multiset rewriting
  - at the configuration level – concurrent activity among nodes
  - at the node level – packets executing concurrently

Representation of a PLAN Program in Maude

The distributed route-finding example can be directly represented and executed in Maude. Some errors in the paper version of the original program have been detected by dynamic typechecking and have been corrected.

```
eq findfun =

fun find [ (dest : host),
           (route : (host * host) list),
           (visited : host list) ] : unit =
 let
   (val neighbors : (host * dev) list =
     getNeighbors[emptyExpList]),
   ....
   (fun sendChild [(n : host * dev), (u : unit) ] : unit =
        OnNeighbor[
        | find |[var(dest),
                 cons[tup[thisHostOf[var(srcdev)],
                          thisHostOf[snd[var(n)]]],var(route)],
                 append[var(myaddrs),var(visited)]],
        fst[var(n)],
        var(childRB),
        snd[var(n)]])
 in
   (if thisHostIs[var(dest)]
    then goback[var(route),reverse[var(route)]]
    else foldr[var(sendChild), var(newNeighbors), unitC])
 end .
```

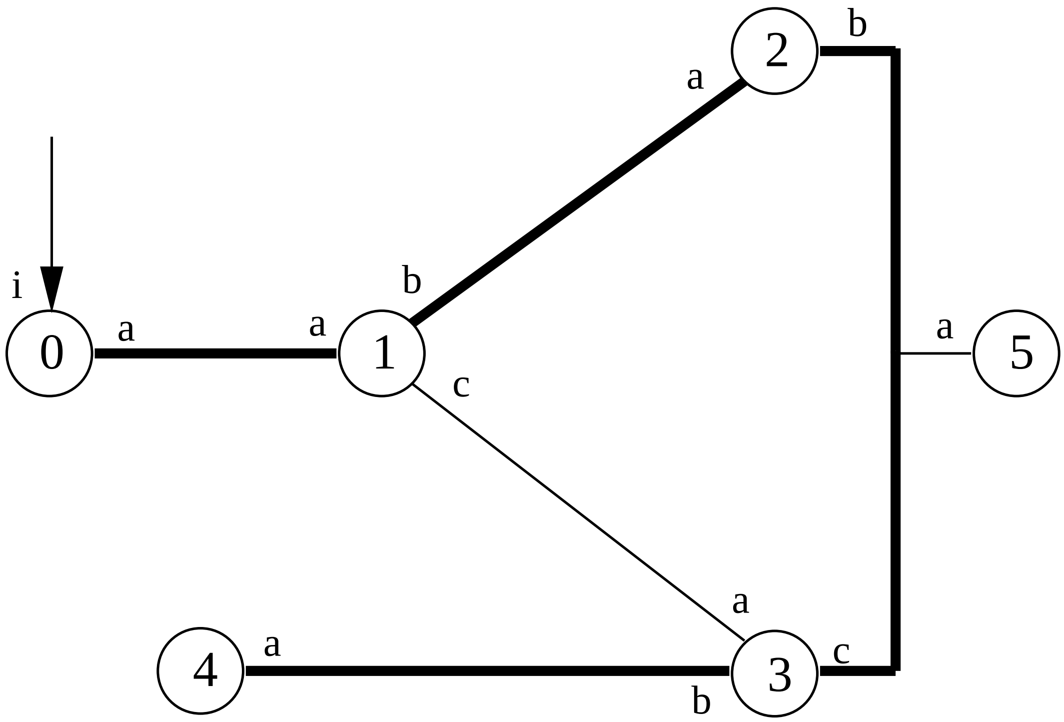
A PLAN Network



Representation of the PLAN Network in Maude

```
 eq conf = inject(node0,pgm(topdef),
                  find, (host(h5a),emptylst,emptylst), 100)

< node0 : PlanEndNode | output : emptyTVList,
  interfaces : (d0i,d0a), addrs : (h0i,h0a),
  neighbors : (nbr(h0i,d0i),nbr(h1a,d0a)),
  me : h0i, port : d0i,
  tasks : noTask >

< node1 : PlanNode | output : emptyTVList,
  interfaces : (d1a,d1b,d1c), addrs : (h1a,h1b,h1c),
  neighbors : (nbr(h0a,d1a),nbr(h2a,d1b),nbr(h3a,d1b)),
  tasks : noTask >

< node2 : PlanNode | output : emptyTVList,
  interfaces : (d2a,d2b), addrs : (h2a,h2b),
  neighbors : (nbr(h1b,d2a),nbr(h3c,d2b),nbr(h5a,d2b)),
  tasks : noTask >

< node3 : PlanNode | output : emptyTVList,
  interfaces : (d3a,d3b,d3c), addrs : (h3a,h3b,h3c),
  neighbors : (nbr(h1c,d3a),nbr(h4a,d3b),
               nbr(h2b,d3c),nbr(h5a,d3c)),
  tasks : noTask >

...
```

A Computed Route from Node 0 to 4



Flexibility

- Informal requirement: anytime typechecking
  - a node may typecheck statically, dynamically, ...
  - typechecking can be node-specific
- Formally captured in Maude
  - by rules that
    * catch ill-typed operations in the execution
    * statically type check programs at injection time
    * type-check execution states
  - combined with use of meta level strategies capturing implementation level design — a node conforms if it raises exceptions according to some strategy for applying the rules.

Future I – Completion

- Completing the specification
  - Fill in missing bits
  - Review with PLAN people
  - make a final pass
- Develop Sample Execution Strategies
- Carry out some case studies

Future II – Evolution

The goal is a living specification that evolves with PLAN and that can be used as a simulator and program analysis tool, and can play with other artifacts.

- PLAN is an evolving language
  - PLAN 3.1 – the paper specifiation
  - PLAN 3.2 – current implementation with polymorphic types
  - PLAN 3.3 – being designed to add cryptographic reflection

Possible Role of Maude in PLAN 3.3

- Experiment with alternative designs
- Execute test cases for multiple nodes
- Ask questions like “who got to see this procedure?”
- Analyze information flow and secrecy requirements