Formal Methods Efforts at JPL

IFIP WG 1.9/2.15 – Leuven, May 11, 2017

Klaus Havelund

With:

Rajeev Joshi (JPL)

Doron Peled (Bar Ilan University, Israel)

Sean Kauffman (University of Waterloo, Canada)

Current Work at JPL

- Scala for modeling
- Runtime verification with BDDs
- Event stream abstraction

Scala for Modeling
Using Scala for development of Hierarchical State Machines

With Rajeev Joshi

VDM++

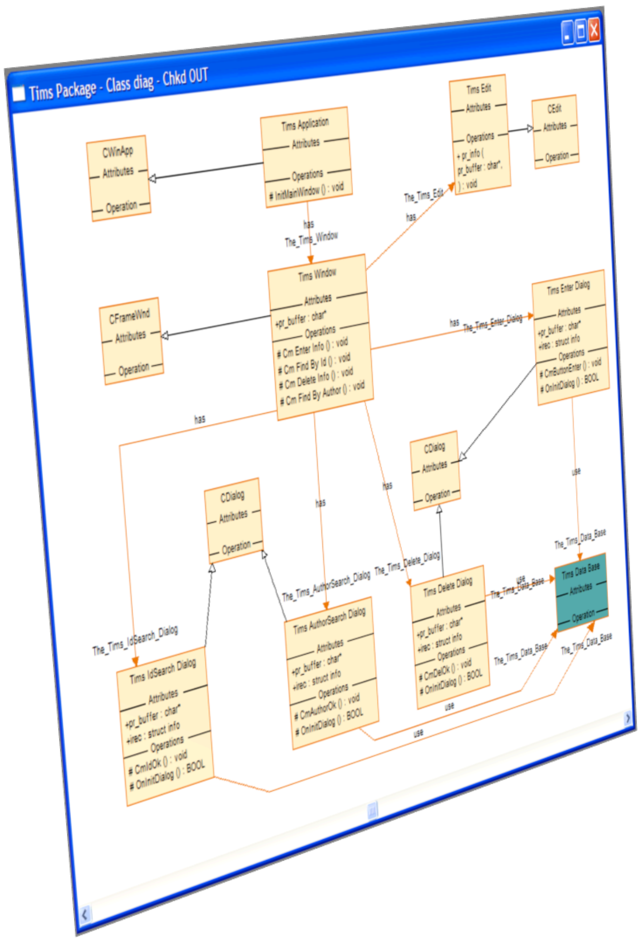
Chemical Plant Alarm System

```
class Plant

instance variables

alarms   : set of Alarm;
schedule : map Period to set of Expert;
inv PlantInv(alarms,schedule);

operations

PlantInv: set of Alarm * map Period to set of Expert ==>
          bool
PlantInv(as,sch) ==
  return
  (forall p in set dom sch & sch(p) <> {}) and
  (forall a in set as &
     forall p in set dom sch &
       exists expert in set sch(p) &
         a.GetReqQuali() in set expert.GetQuali());

types

public Period = token;

operations

public ExpertToPage: Alarm * Period ==> Expert
ExpertToPage(a, p) ==
  let expert in set schedule(p) be st
      a.GetReqQuali() in set expert.GetQuali()
  in
    return expert
pre a in set alarms and
    p in set dom schedule
post let expert = RESULT
     in
       expert in set schedule(p) and
       a.GetReqQuali() in set expert.GetQuali();
```

Scala

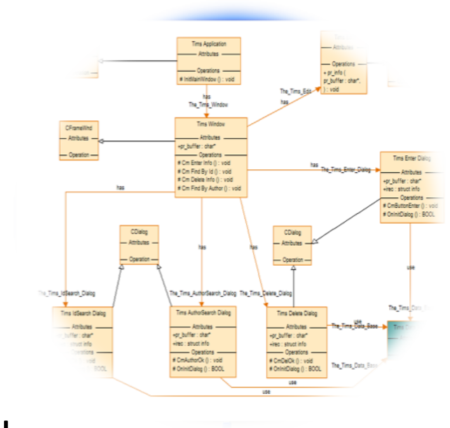
```
class Plant(alarms: Set[Alarm],
    schedule: Map[Period, Set[Expert]]) {
  assert(PlantInv(alarms, schedule))

  def PlantInv(alarms: Set[Alarm], schedule: Map[Period,
    Set[Expert]]): Boolean =
    (schedule.keySet forall { schedule(_) != Set() }) &&
      (alarms forall { a =>
        schedule.keySet forall { p =>
          schedule(p) exists { expert =>
            a.reqQuali ? expert.quali
          }
        }
      })

  def ExpertToPage(a: Alarm, p: Period): Expert = {
    require(a ? alarms && p ? schedule.keySet)
    schedule(p) suchthat {expert =>
      a.reqQuali ? expert.quali}
  } ensuring { expert =>
    a.reqQuali ? expert.quali &&
      expert ? schedule(p)
  }
```

Modeling @ JPL

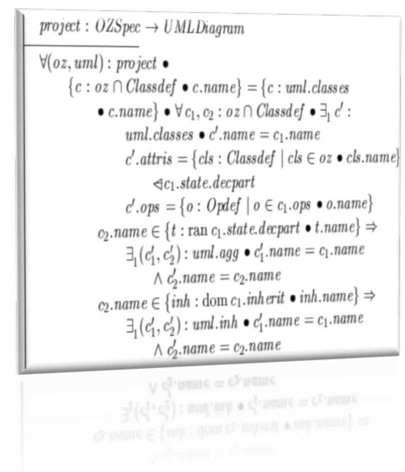


SysML

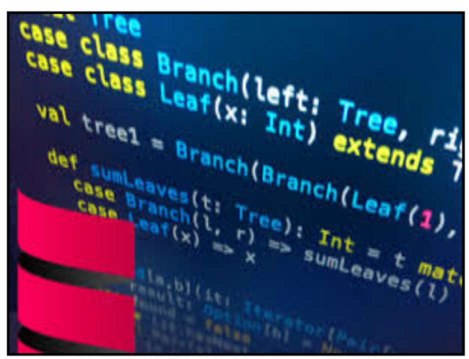
Languages:
graphical, formal, programming



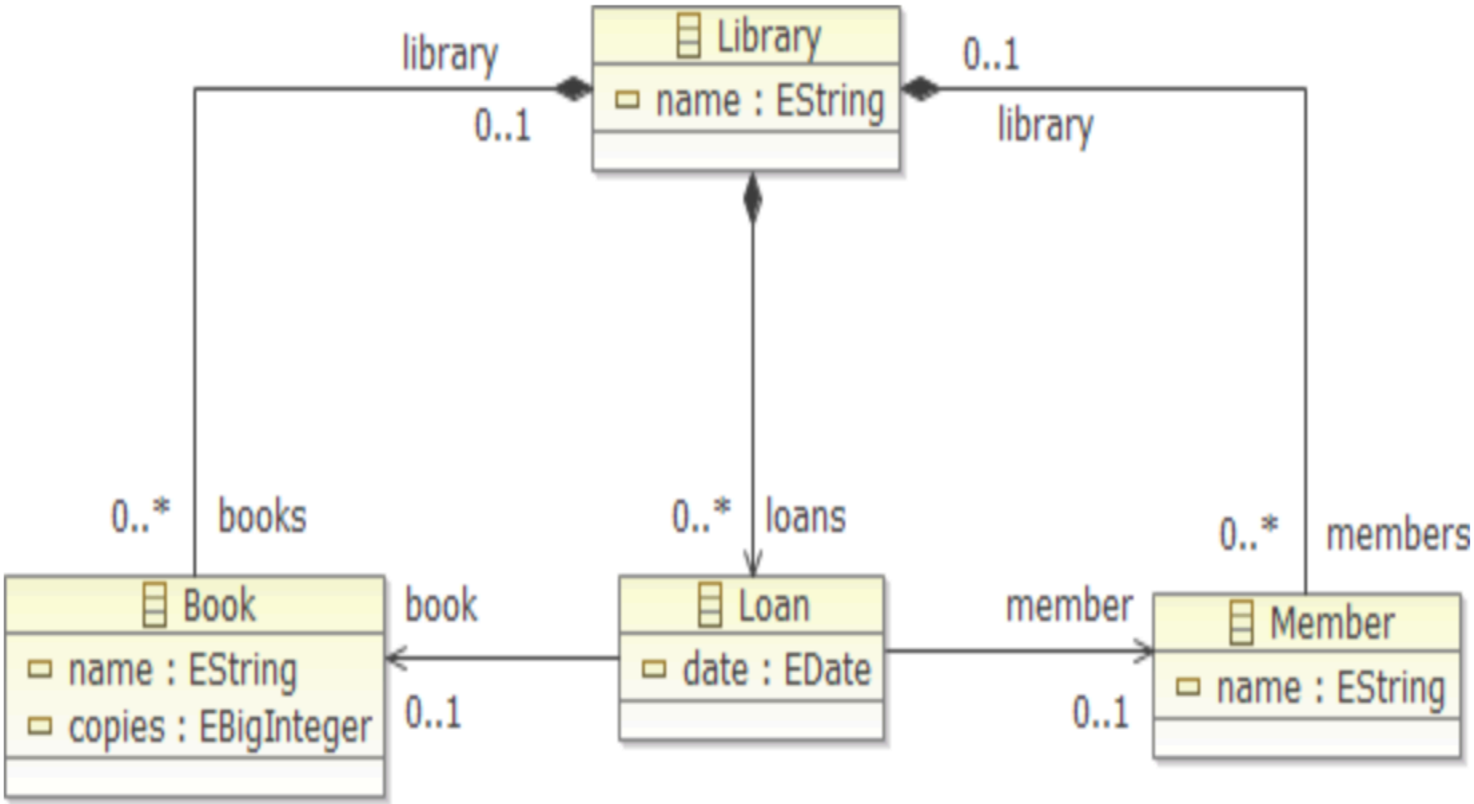
Graphical:
- UML
- SysML
- …



Formal
- VDM
- Z
- PVS
- Promela
- …



Programming
- Python
- Java,
- Scala,
- Ocaml
- …


Library
name : EString
library
0..1
0..1
library
0..* books
0..* loans
0..* members
Book
name : EString
copies : EBigInteger
book
0..1
Loan
date : EDate
member
0..1
Member
name : EString

Requirement

The number of loans that a book is part of should be less than or equal to the number of copies of the book

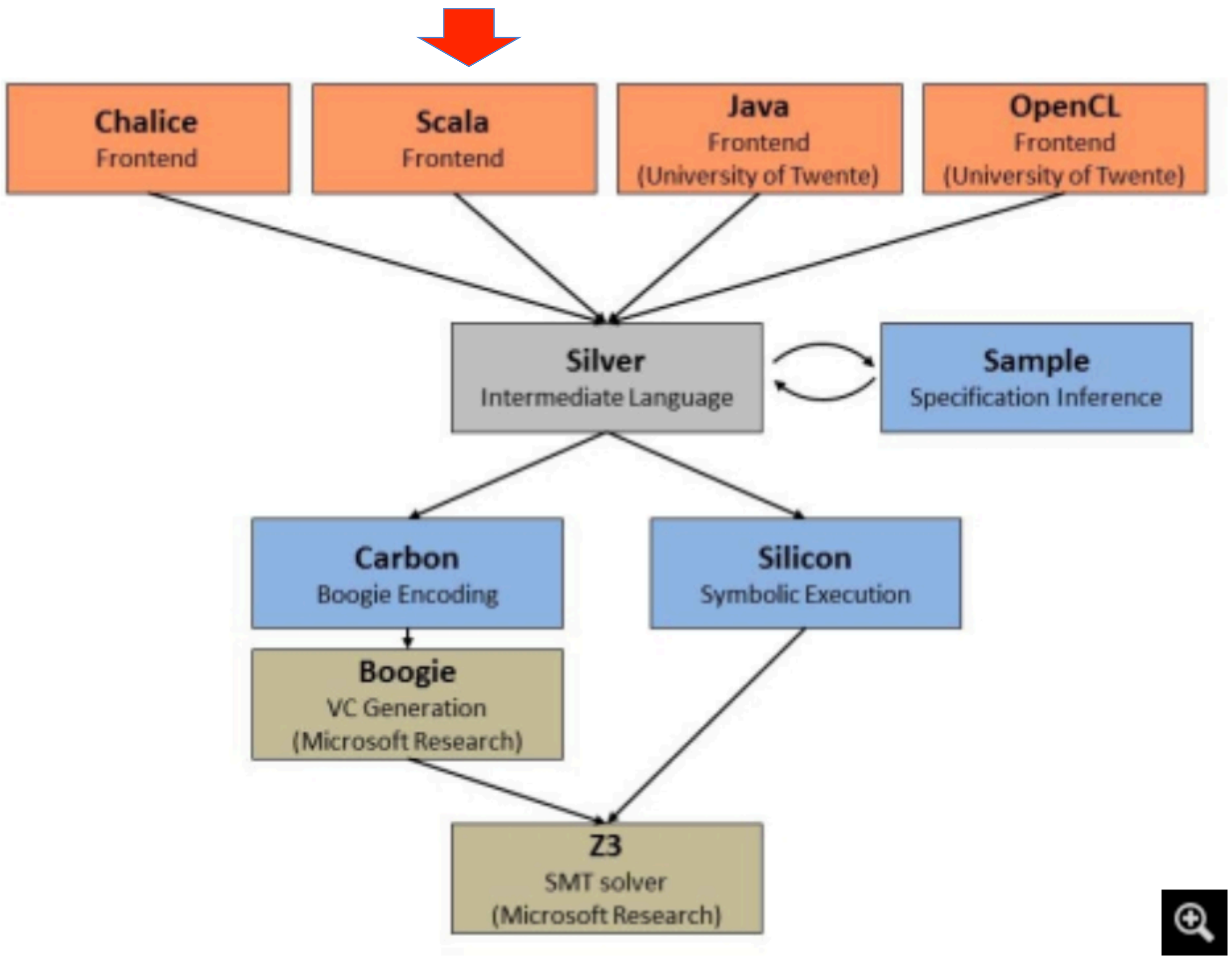
K

```
class Book {
  name : String
  copies : Int
  library : Library

  loans : Set[Loan] =
    Set{ loan | loan : Loan •
      loan ∈ library . loans ∧ loan . book = this
    }

  fun isAvailable : Bool {
    size ( loans ) < copies
  }

  req CopiesPositive : copies > 0
  req SufficientCopies : size ( loans ) <= copies
}
```

Scala

```
trait Book extends Model {
  var name: String
  var copies: Int
  var library: Library

  def loans: Set[Loan] =
    library.loans.filter(_.book eq this)

  def isAvailable(): Boolean = loans.size < copies

  invariant("CopiesPositive") { copies > 0 }

  invariant("SufficientCopies") {
    loans.size <= copies
  }
}
```

Formal Verification of Scala Programs



Peter Muller and Alex Summers

Motivation for focusing on Hierarchical State Machines

- HSMs used extensively in FSW (on MSL, SMAP, M2020, FswCore)
- FSW typically has many HSMs interacting with each other and with devices
- During design, hard to analyze and understand complex interactions among HSMs, and how behavior is affected
  - by faults
  - by devices in off-nominal fashion (e.g., taking longer to warm up)
- We need a way to perform easy design and analysis of HSMs:
  - easy to model existing MSL/SMAP designs in expressive DSL
  - easy to write scenario test cases
  - easy to write properties to be monitored during tests
  - eventually verify

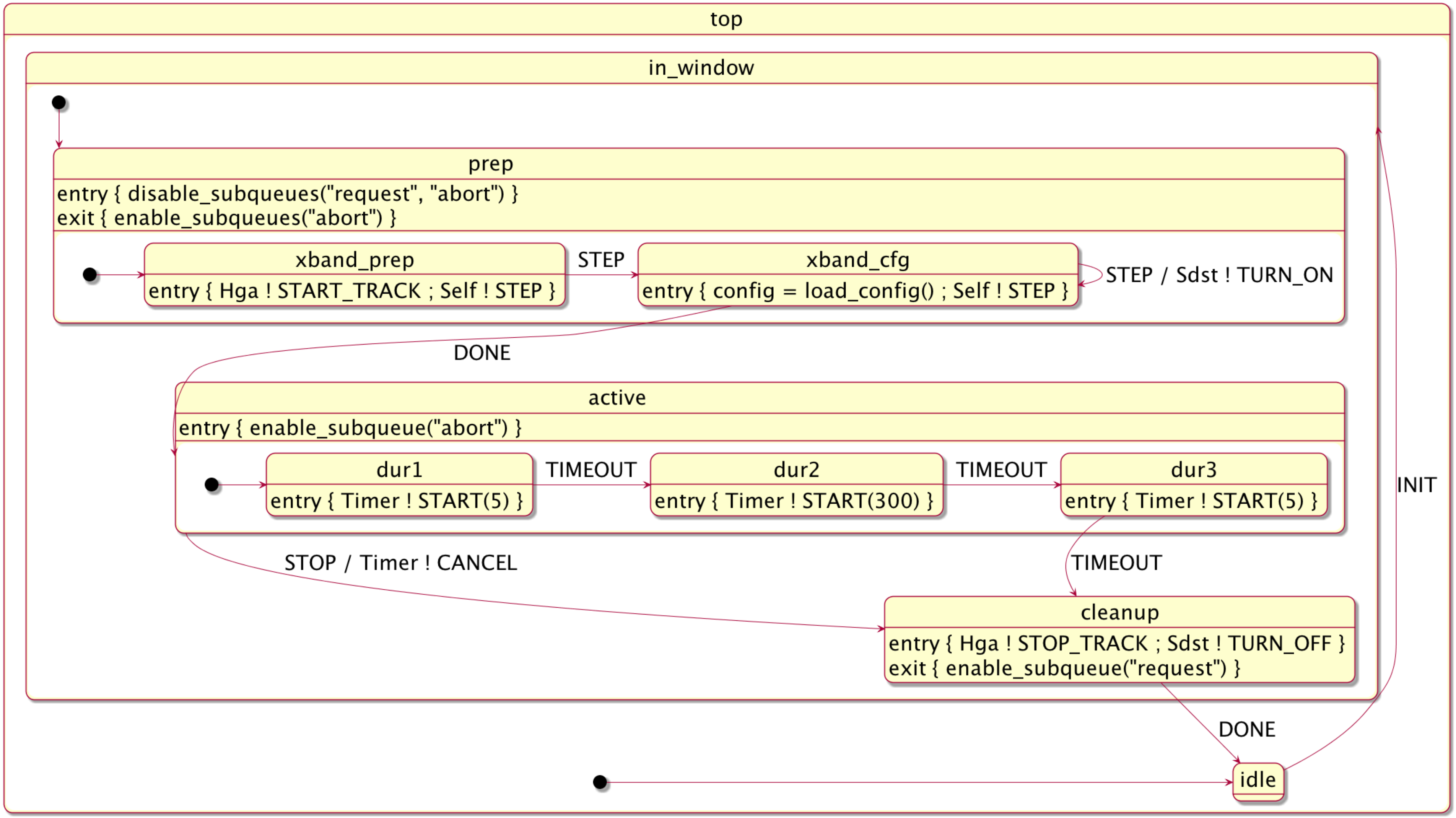
Approach

- Develop domain-specific language as an API (internal DSL) in the high-level strongly typed object-oriented and functional programming language Scala.
- This allows fast and safe prototyping of new functionality.
- Design and implement DSL for Hierarchical State Machines (HSMs) in Scala.
- Design and implement DSL for monitoring and testing in Scala.
- Develop automated testing framework.
- Develop formal verification technology for Scala.

Case Study: CBM (Coordinated Communications Behaviors), used on MSL/M2020/Europa Clipper

- CBM HSM inherited from MSL has ~50 states.
- Interacts with many devices and low-level subsystems in order to establish spacecraft state for telecom configuration and for communication windows.
- Complex interactions have led to surprises during MSL
  - Sol 696-698 anomaly resulted in 3 days of lost science due to frequency sweep not happening on time due to prolonged mounting of file system.
- Ideally, these should have been discovered during V&V
  - but hard to enumerate all interactions using traditional testing.
- Our approach should allow automatic test-case generation from high-level HSM and scenario descriptions; during each test case run, a set of monitored properties are checked automatically.

A Hierarchical State Machine



```
case object STEP extends CbmMessage("transition")
case object DONE extends CbmMessage("transition")
case object TIMEOUT extends CbmMessage("transition")
case object STOP extends CbmMessage("abort")
case object INIT extends CbmMessage("request")

class CbmHsm extends MslHsm {
  object top extends state() {}
  object idle extends state(top, true) {
    when {
      case INIT ⇒ in_window
    }
  }
  object in_window extends state(top) { }
  object prep extends state(in_window, true) {
    entry { disable_subqueues("request", "abort") }
    exit { enable_subqueues("abort") }
  }
  object xband_prep extends state(prep, true) {
    entry { Hga ! START_TRACK ; Self ! STEP }
    when { case STEP ⇒ xband_cfg }
  }
  object xband_cfg extends state(prep) {
    entry { config = load_config() ; Self ! STEP }
    when {
      case STEP ⇒ stay exec { Sdst ! TURN_ON }
      case DONE ⇒ active
    }
  }
  object active extends state(in_window) {
    entry { enable_subqueues("abort") }
    when { case STOP ⇒ cleanup exec { Timer ! CANCEL } }
  }
   ...
}
```

```
// In state in_window, the request subqueue is disabled
class QueueCheck extends MSLMonitor {
  invariant ("requestDisabled") {
    Cbm.inState("in_window") ==> !Cbm.isEnabled("request")
  }
}

// A Timer is never started while a previous timer is active
class TimerCheck extends MSLMonitor {
  always {
    case TIM_EVR_STARTED(_) => watch {
      case TIM_EVR_FIRED(_) | TIM_EVR_CANCELED(_) => ok
      case TIM_EVR_STARTED(_) => error("Timer restarted")
    }
  }
}
```

Scala

Plans for Test Case Generation

```
class scenario1 extends TestScenario {
  at(30) exec { Cbm ! INIT }
}
```

```
class scenario2 extends scenario1 {
  Cbm in "dur2" exec { Cbm ! STOP }
}
```

```
taskSet := set of tasks in scenario
while (taskSet ≠ ∅) {
    choose T in taskSet with least value of T.time such that T is enabled
    advance current time to T.time
    perform T.action
    while (there is an HSM that has a pending message) {
        choose HSM H such that H has a pending message
        execute H on its highest priority message
    }
}
```

HSMs in Scala

```
trait HSM[Event] {...}
```

The HSM trait defines the following types and values used throughout:

```
type Code = Unit ⇒ Unit
type Target = (state, Code)
type Transitions = PartialFunction[Event, Target]
val noTransitions: Transitions = {case _ if false ⇒ null}
val skip: Code = (x: Unit) ⇒ {}
```

HSMs in Scala

```
case class state(parent: state = null, init : Boolean = false) {
  var entryCode: Unit ⇒ Unit = skip
  var exitCode: Unit ⇒ Unit = skip
  var transitions : Transitions = noTransitions
  ...
  def entry(code: ⇒ Unit): Unit = {entryCode = (x: Unit) ⇒ code}
  def exit(code: ⇒ Unit): Unit = {exitCode = (x: Unit) ⇒ code}
  def when(ts: Transitions ): Unit = { transitions = ts}

  implicit def state2Target(s: state ): Target = (s, skip)
  implicit def state2Exec(s: state) = new {
    def exec(code: ⇒ Unit) = (s, (x: Unit) ⇒ code) }
}
```

HSMs in Scala

```
def initial(s: state): Unit = {current = s.getInnerMostState}
```

```
var initialState : state = null
if (parent != null && init) {parent.initialState = this}
def getInnerMostState: state =
  if (initialState == null) this else initialState.getInnerMostState
def getSuperStates: List[state] =
  (if (parent == null) Nil else parent.getSuperStates) ++List(this)
```

HSMs in Scala

```
var current: state = null

def submit(event: Event): Unit = {
  findTriggerHappyState(current, event) match {
    case None ⇒
    case Some(triggerState) ⇒
      val (transitionState, transitionCode) = triggerState.transitions(event)
      val targetState = transitionState.getInnerMostState
      val (exitStates, enterStates) = getExitEnter((current, targetState))
      for (s <- exitStates) s.exitCode()
      transitionCode()
      for (s <- enterStates) s.entryCode()
      current = targetState
  }}
```

HSMs in Scala

```
def findTriggerHappyState(s: state, event: Event): Option[state] =
  if (s.transitions.isDefinedAt(event)) Some(s) else
  if (s.parent == null) None else findTriggerHappyState(s.parent, event)
```

Monitors in Scala

```
class Monitor[Event] {
  type Transitions = PartialFunction[Event, Set[state]]
  var states: Set[state] = Set()
  var invariants: List[(String, Unit ⇒ Boolean)] = Nil
  ...
  trait state { thisState ⇒
    var transitions: Transitions = noTransitions
    ...
    def apply(event: Event): Option[Set[state]] =
      if (transitions.isDefinedAt(event)) Some(transitions(event)) else None
    def watch(ts: Transitions) { transitions = ts }
    def always(ts: Transitions) { transitions = ts andThen (_ + this) }
  }
  def watch(ts: Transitions) = new state { watch(ts) }
  def always(ts: Transitions) = new state { always(ts) }
}
```

Publications

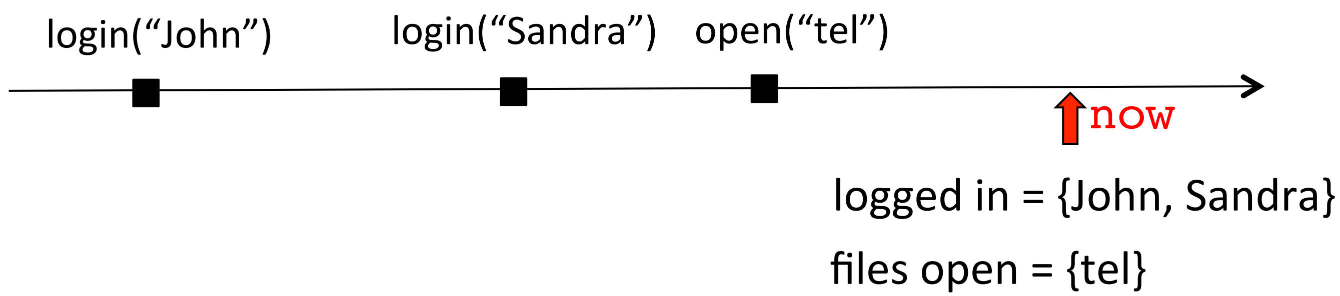
[1] Modeling and Monitoring of Hierarchical State Machines in Scala
K. Havelund and R. Joshi. Submitted March 2017.

[2] **Modeling Rover Communication using Hierarchical State Machines in Scala**
K. Havelund and R. Joshi. Submitting May 5, 2017.

QTL
Quantified temporal logic for monitoring using BDDs

With Doron Peled (Bar Ilan University, Israel)

The Problem



Property:

Whenever a user accesses a file:

- *the user must have logged in*
- *the file must have been opened*

The Logic

$$\varphi ::= true \mid p(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x \bullet \varphi \mid \ominus\varphi \mid \varphi_1 \,\mathbf{S}\, \varphi_2$$

$$t ::= c \mid x$$

Derived Constructs

$$false = \neg true$$

$$\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$$

$$\varphi_1 \Rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$$

$$\forall x \bullet \varphi = \neg\exists x \bullet \neg\varphi$$

$$\mathbf{P}\,\varphi = true\,\mathbf{S}\,\varphi$$

$$\mathbf{H}\,\varphi = \neg\mathbf{P}\,\neg\varphi$$

$$[\varphi_1, \varphi_2) = (\neg\varphi_2)\,\mathbf{S}\,\varphi_1$$

Example

$$\begin{array}{l} \forall\, user \bullet \forall\, file \bullet \\ \quad access(user, file) \Rightarrow \\ \qquad [login(user), logout(user)) \\ \qquad\qquad \wedge \\ \qquad [open(file), close(file)) \end{array}$$

First Semantics

- $(\varepsilon, \sigma, i) \models \mathit{true}$.
- $(\varepsilon, \sigma, i) \models p(a)$ iff $p(a) \in \sigma[i]$.
- $([v \mapsto a], \sigma, i) \models p(v)$ if $p(a) \in \sigma[i]$.
- $(\gamma, \sigma, i) \models (\varphi \wedge \psi)$ if $(\gamma|_{vars(\varphi)}, \sigma, i) \models \varphi$ and $(\gamma|_{vars(\psi)}, \sigma, i) \models \psi$.
- $(\gamma, \sigma, i) \models \neg\varphi$ if not $(\gamma, \sigma, i) \models \varphi$.
- $(\gamma, \sigma, i) \models (\varphi\ \mathcal{S}\ \psi)$ if for some $j \leq i$, $(\gamma|_{vars(\psi)}, \sigma, j) \models \psi$ and for all $j < k \leq i$ $(\gamma|_{vars(\varphi)}, \sigma, k) \models \varphi$.
- $(\gamma, \sigma, i) \models \ominus\varphi$ if $i > 1$ and $(\gamma, \sigma, i-1) \models \varphi$.
- $(\gamma, \sigma, i) \models \exists x\ \varphi$ if there exists $a \in \mathit{domain}(x)$ such that[5] $(\gamma[x \mapsto a], \sigma, i) \models \varphi$.

$$(\varphi\,\mathcal{S}\,\psi) = (\psi \vee (\varphi \wedge \ominus\varphi\,\mathcal{S}\,\psi))$$

- $(\gamma, \sigma, i) \models (\varphi\,\mathcal{S}\,\psi)$ if $(\gamma|_{vars(\psi)}, \sigma, i) \models \psi$ or both $(\gamma|_{vars(\varphi)}, \sigma, i) \models \varphi$ and $(\gamma, \sigma, i-1) \models (\varphi\,\mathcal{S}\,\psi)$.

Second Semantics

- $I[\varphi, \sigma, 0] = \emptyset$.
- $I[true, \sigma, i] = \{\varepsilon\}$.
- $I[p(a), \sigma, i] =$ if $p(a) \in \sigma[i]$ then ε, else $\emptyset$.
- $I[p(v), \sigma, i] = \{[v \mapsto a] | p(a) \in \sigma[i]\}$.
- $I[(\varphi \wedge \psi), \sigma, i] = I[\varphi, \sigma, i] \bigcap I[\psi, \sigma, i]$.
- $I[\neg\varphi, \sigma, i] = A_{vars(\varphi)} \setminus I[\varphi, \sigma, i]$.
- $I[(\varphi \mathcal{S} \psi), \sigma, i] = I[\psi, \sigma, i] \bigcup (I[\varphi, \sigma, i] \bigcap I[(\varphi \mathcal{S} \psi), \sigma, i-1])$.
- $I[\ominus\varphi, \sigma, i] = I[\varphi, \sigma, i-1]$.
- $I[\exists x\, \varphi, \sigma, i] = proj(I[\varphi, \sigma, i], \{x\})$.

Mapping data to BDDs

prop p: **forall** f . close(f) $\rightarrow$ **exists** m . **P** open(f,m)

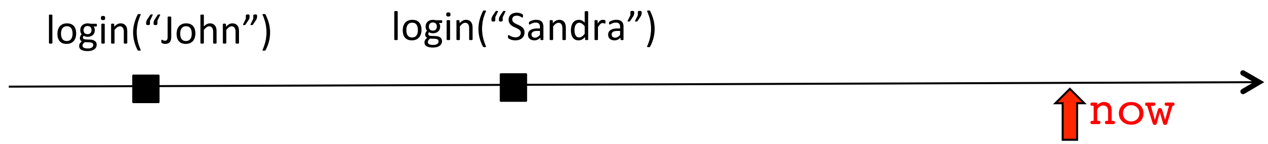
$$\langle open(input, read), open(output, write), close(out)\rangle$$

$$f \mapsto [input \mapsto 000, output \mapsto 001, out \mapsto 010],$$
$$m \mapsto [read \mapsto 000, write \mapsto 001]$$

Algorithm

1. Initially, for each subformula φ, $\mathsf{now}(\varphi) = 0$.
2. Observe a new state (as set of ground predicates) s as input.
3. Let $\mathsf{pre} := \mathsf{now}$.
4. Make the following updates for each subformula. If φ is a subformula of ψ then $\mathsf{now}(\varphi)$ is updated before $\mathsf{now}(\psi)$.
   - $\mathsf{now}(\mathit{false}) = 0$
   - $\mathsf{now}(p(a)) = \text{if } p(a) \in s \text{ then } 1 \text{ else } 0$
   - $\mathsf{now}(p(x)) = \text{if } p(a) \in s \text{ then } \mathbf{build}(x,a) \text{ else } 0$
   - $\mathsf{now}((\varphi \wedge \psi)) = \mathbf{and}(\mathsf{now}(\varphi), \mathsf{now}(\psi))$
   - $\mathsf{now}(\neg\varphi) = \mathbf{not}(\mathsf{now}(\varphi))$
   - $\mathsf{now}((\varphi\, \mathcal{S}\, \psi)) = \mathbf{or}(\mathsf{now}(\psi), \mathbf{and}(\mathsf{now}(\varphi), \mathsf{pre}((\varphi\, \mathcal{S}\, \psi))))$.
   - $\mathsf{now}(\ominus\, \varphi) = \mathsf{pre}(\varphi)$
   - $\mathsf{now}(\exists x\, \varphi) = \mathbf{exists}(\langle x_0, \ldots, x_n \rangle, \mathsf{now}(\varphi))$
5. Goto step 2.

The Implementation
Binary Decision Diagrams



logged in = {John, Sandra}

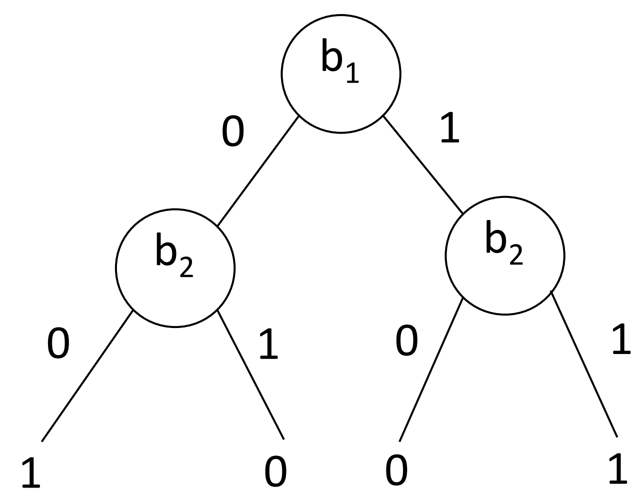
Users as bit patterns

	$b_1 b_2$
John →	00
Ann →	01
Mike →	10
Sandra →	11

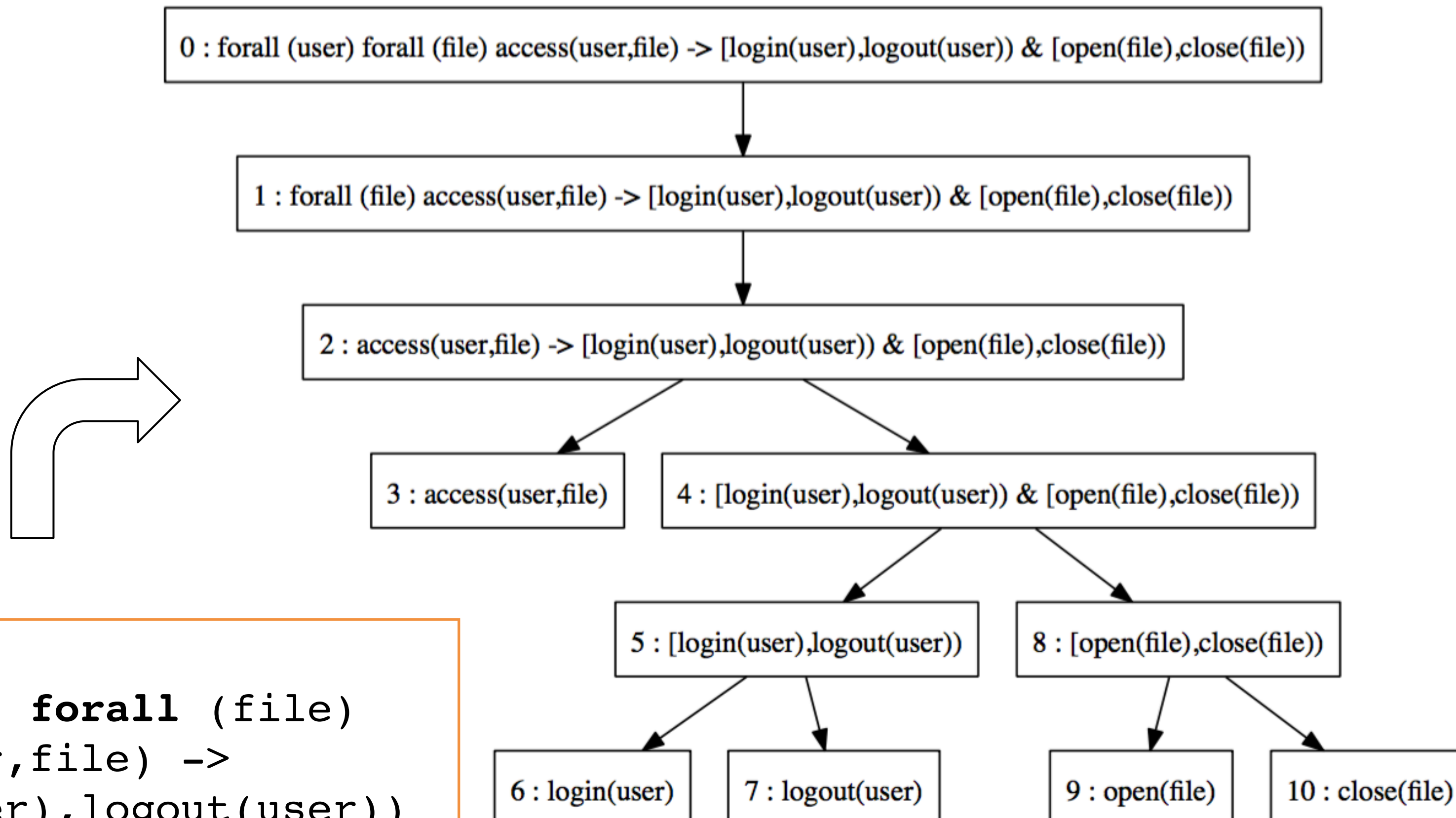
The set {John, Sandra} as a Boolean expression:

$$(b_1 = 0 \wedge b_2 = 0) \vee (b_1 = 1 \wedge b_2 = 1)$$

Boolean expression as BDD:



The Structure of a Property




```
prop secure :
  forall (user) forall (file)
    access(user,file) ->
     [login(user),logout(user))
       &
     [open(file),close(file))
```

Example Application

```
prop secure :
  forall (user) forall (file)
    access(user,file) ->
     [login(user),logout(user))
       &
     [open(file),close(file))
```

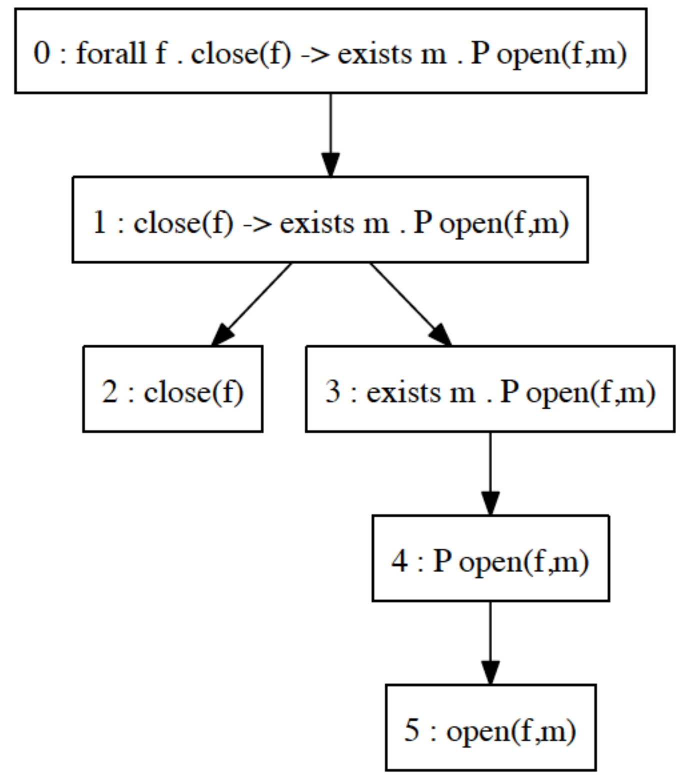
```
login,John
open,tel
access,John,tel
close,tel
access,John,tel
logout,John
```

```
class Formula_secure extends Formula {
  val var_user :: var_file :: Nil = declareVariables("user", "file")

  override def evaluate(): Boolean = {
    now(10) = build("close")(V("file"))
    now(9) = build("open")(V("file"))
    now(8) = now(9).or(now(10).not().and(pre(8)))
    now(7) = build("logout")(V("user"))
    now(6) = build("login")(V("user"))
    now(5) = now(6).or(now(7).not().and(pre(5)))
    now(4) = now(5).and(now(8))
    now(3) = build("access")(V("user"),V("file"))
    now(2) = now(3).not().or(now(4))
    now(1) = now(2).forAll(var_file)
    now(0) = now(1).forAll(var_user)

    tmp = now
    now = pre
    pre = tmp
    !tmp(0).isZero
  }

  pre = Array.fill(11)(False)
  now = Array.fill(11)(False)
}
```

```
*** Property secure violated on event number 5:
    access(John,tel)
```

prop p: **forall** f . close(f) → **exists** m . **P** open(f,m)

```
class Formula_p extends Formula {
  var pre: Array[BDD] = Array.fill(6)(False)
  var now: Array[BDD] = Array.fill(6)(False)
  var tmp: Array[BDD] = null
  val var_f :: var_m :: Nil =
    declareVariables("f", "m")

  override def evaluate(): Boolean = {
    now(5) = build("open")(V("f"),V("m"))
    now(4) = now(5).or(pre(4))
    now(3) = now(4).exist(var_m)
    now(2) = build("close")(V("f"))
    now(1) = now(2).not().or(now(3))
    now(0) = now(1).forAll(var_f)
    tmp = now; now = pre; pre = tmp
    !tmp(0).isZero
  }
}
```



Evaluation of Trace

$$\langle open(input, read), open(output, write), close(out)\rangle$$

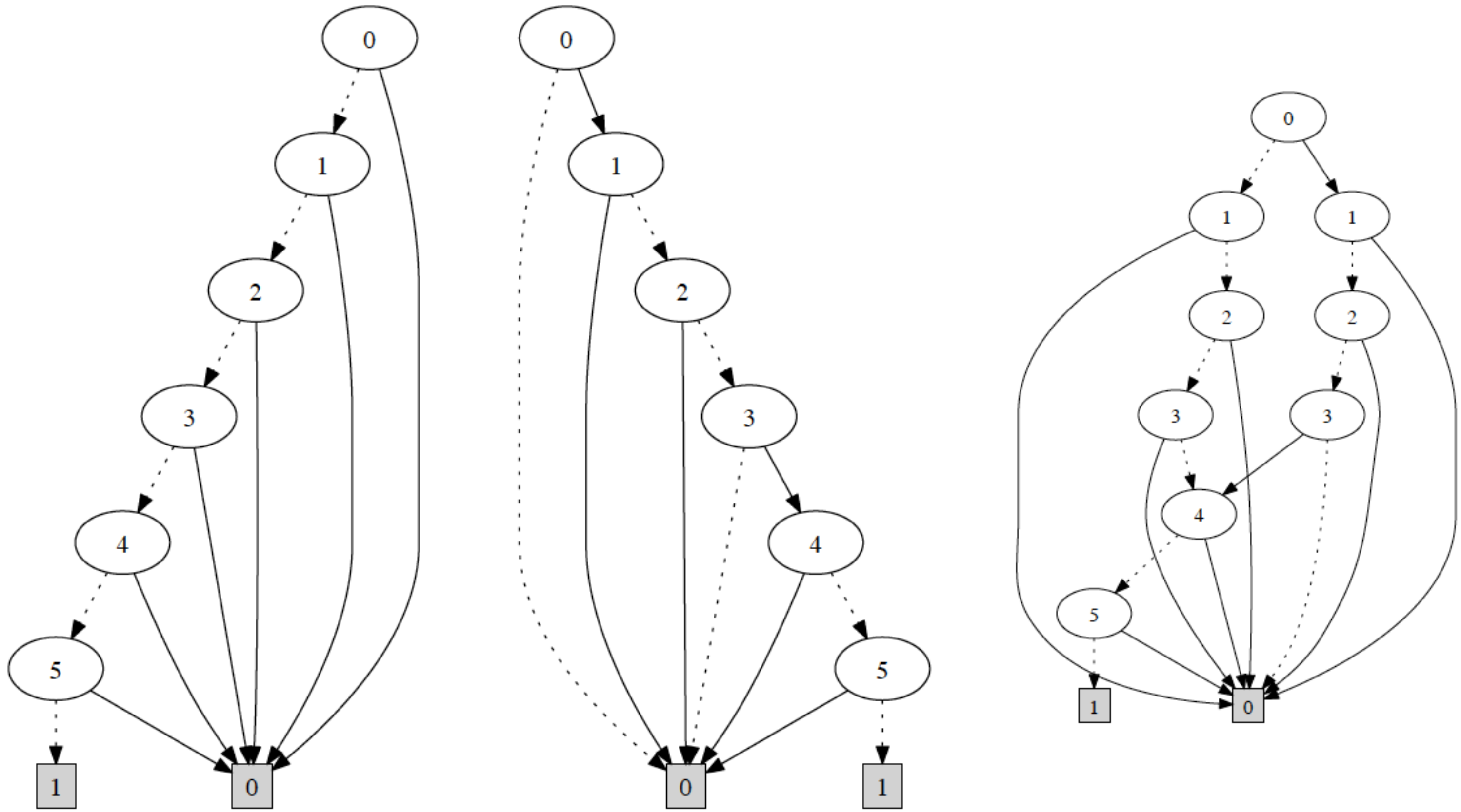


Fig. 2: BDD for nodes for (left) subformula 5 on first event, (mid) subformula 5 on second event, (right) subformula 4 on second event

Evaluation Properties in QTL

```
prop access : forall u . forall f .
  access(u,f) → [login(u),logout(u)) & [open(f),close(f))

prop file : forall f .
  close(f) → exists m . @ [open(f,m),close(f))

prop fifo : forall x .
  (enter(x) → ! @ P enter(x)) &
  (exit(x) → ! @ P exit(x)) &
  (exit(x) → @ P enter(x)) &
  (forall y . (exit(y) & P (enter(y) & @ P enter(x))) → @ P exit(x))
```

Evaluation Properties in MonPoly

```
/* access */ FORALL u. (FORALL f.
  ( access(u,f) IMPLIES
    ((( NOT logout(u)) SINCE login(u)) AND (NOT close(f) SINCE[0,*] open(f)))))

/* file */ FORALL f .
  (close(f) IMPLIES (EXISTS m . PREVIOUS ( NOT close(f) SINCE[0,*] open(f,m) )))

/* fifo */ FORALL x. (
 (enter(x) IMPLIES NOT PREVIOUS ONCE[0,*] enter(x)) AND
 ( exit (x) IMPLIES NOT PREVIOUS ONCE[0,*] exit(x)) AND
 ( exit (x) IMPLIES PREVIOUS ONCE[0,*] enter(x)) AND
 FORALL y.
   (( exit (y) AND ONCE[0,*] (enter(y) AND PREVIOUS ONCE[0,*] enter(x)))
     IMPLIES PREVIOUS ONCE exit(x)))
```

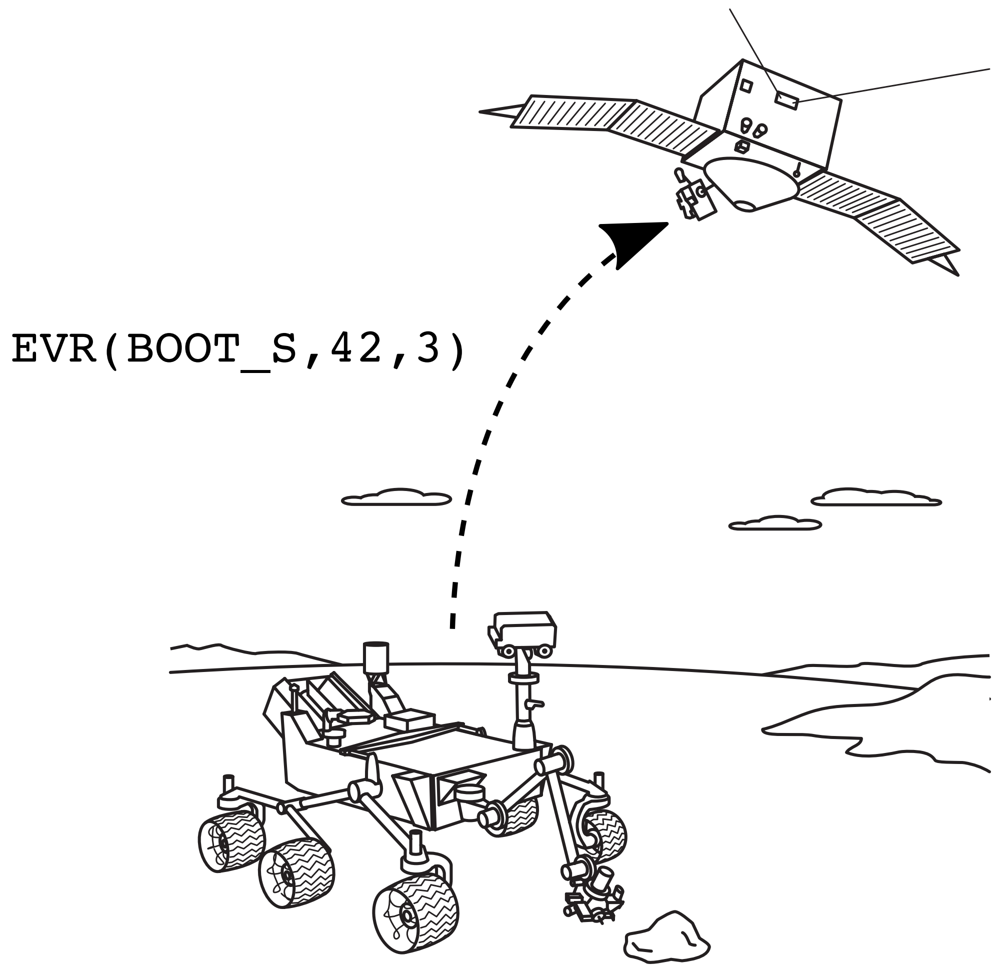
Evaluation Results

Table 1: Evaluation of QTL and MONPOLY

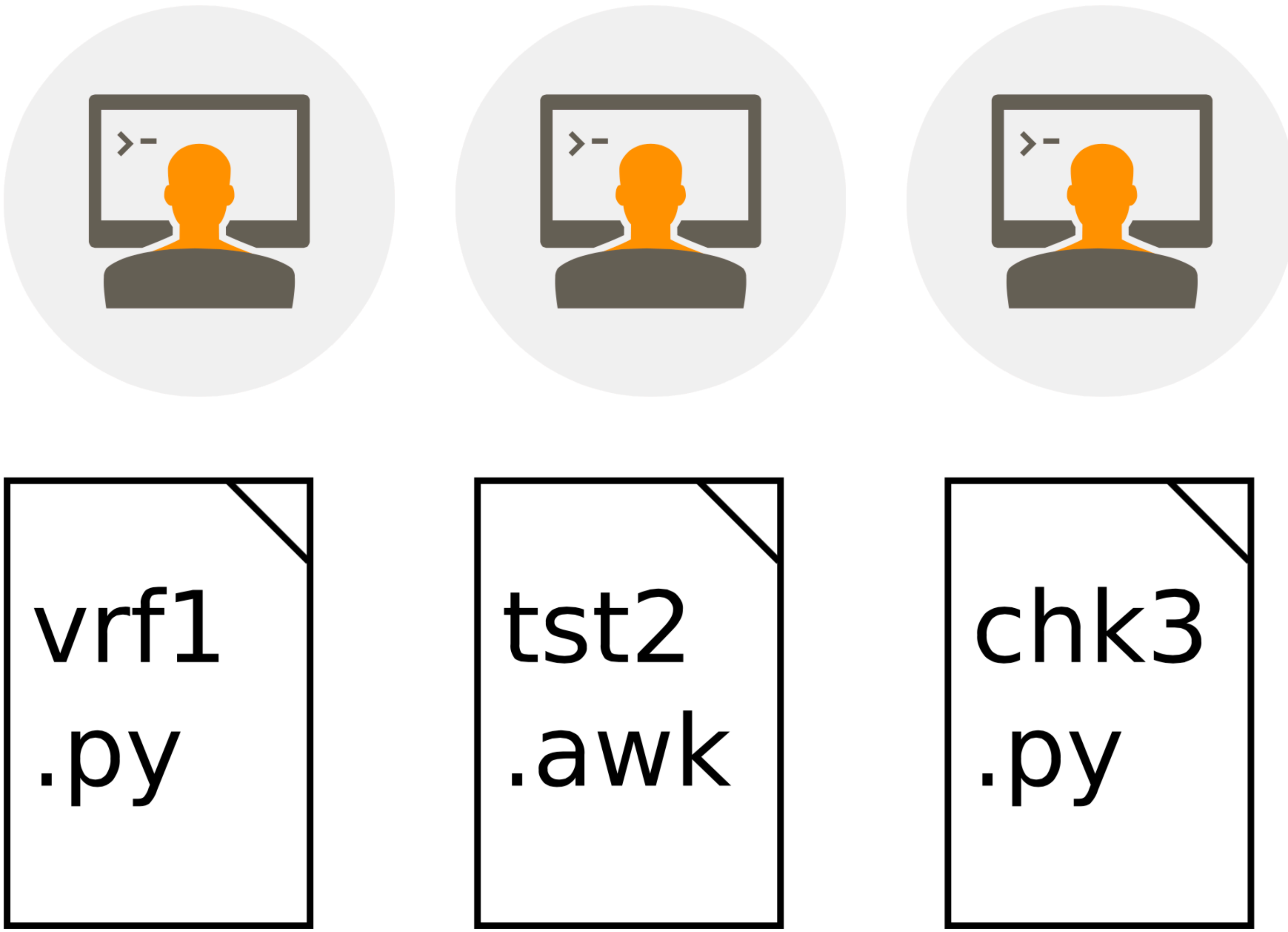
Property	Trace length	MONPOLY (sec)	QTL (sec) bits per var.: 20 (40, 60)
ACCESS	11,006	**1.9**	**3.1** (3.3, 3.2)
	110,006	**241.9**	**6.1** (9.1, 10.9)
	1,100,006	**58,455.8**	**36.8** (61.9, 88.8)
FILE	11,004	**61.1**	**2.8** (2.8, 3.0)
	110,004	**7,348.7**	**6.3** (6.5, 8.6)
	1,100,004	**DNF**	**30.3** (43.9, 59.5)
FIFO	5,051	**158.3**	**195.4** (OOM, ?)
	10,101	**1140.0**	**ERR** (?, ?)

Nfer
A tool for event stream abstraction

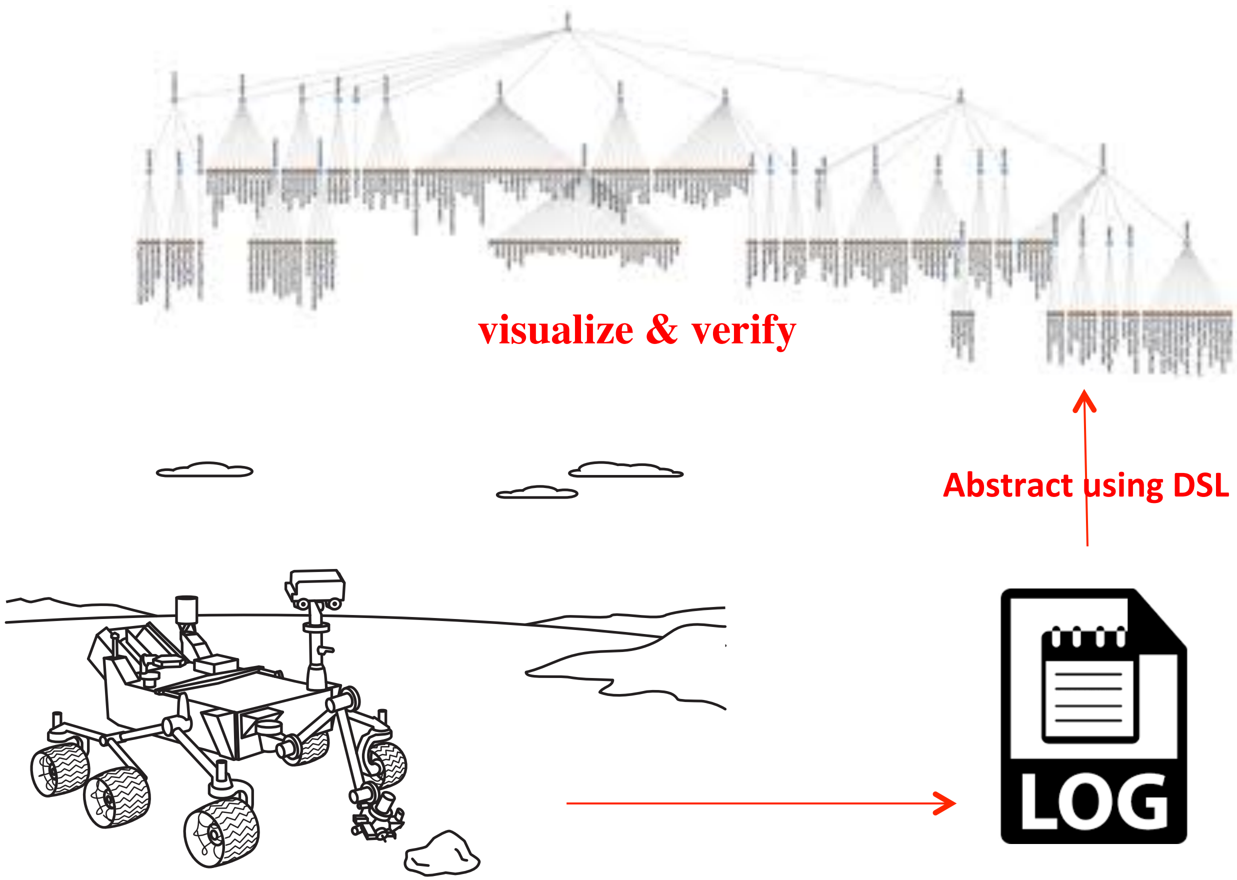
With Rajeev Joshi and Sean Kauffman (U. of Waterloo, Canada)


EVR(BOOT_S,42,3)

Current Situation



Nfer
A Tool for Event Abstraction



Grammar for DSL

```
specification ::= rule+ | module+
module ::= 'module' identifier '{' [imports] rule* '}'
imports ::= 'import' identifier (',' identifier)* ';'
rule ::= identifier ':-' intervalExpression [whereExpression] [mapExpression] [endPoints]
intervalExpression ::= primaryIntervalExpression (intervalOp primaryIntervalExpression)*
primaryIntervalExpression ::= atomicIntervalExpression | parenIntervalExpression
atomicIntervalExpression ::= [label] identifier
parenIntervalExpression ::= '(' intervalExpression ')'
label ::= identifier ':'
whereExpression ::= 'where' expression
mapExpression ::= 'map' '{' identifier '->' expression (',' identifier '->' expression)* '}'
endPoints ::= 'begin' expression 'end' expression
intervalOp ::= 'also'|'before'|'meet'|'during'|'start'|'finish'|'overlap'|'slice'|'coincide'
expression ::= comparisonExpression (andorOp comparisonExpression)*
comparisonExpression ::= plusminusExpression (comparisonOp plusminusExpression)*
plusminusExpression ::= muldivExpression (plusminusOp muldivExpression)*
muldivExpression ::= unaryExpression (muldivOp unaryExpression)*
unaryExpression ::=
  stringLiteral | intLiteral | doubleLiteral | booleanLiteral |
  beginTime | endTime | mapField | callExpression |
  '(' expression ')' | unaryOp unaryExpression
mapField ::= identifier '.' identifier
beginTime ::= identifier '.' BEGIN
endTime ::= identifier '.' END
unaryOp ::= '-' | '!'
muldivOp ::= '*'  | '/'  | '%' )
plusminusOp ::= '+'  | '-'
comparisonOp ::= '<' | '<=' | '>' | '>=' | '=' | '!='
andorOp ::= '&' | '|'
callExpression ::= 'call' '(' identifier ')'
```

Event Abstraction with nfer

NAME	TIME	PARAMS
DOWNLINK	10	size -> 430
BOOT_S	42	count -> 3
TURN_ANTENNA	80	
START_RADIO	90	
DOWNLINK	100	size -> 420
BOOT_E	160	
STOP_RADIO	205	
BOOT_S	255	count -> 4
START_RADIO	286	
BOOT_E	312	
TURN_ANTENNA	412	

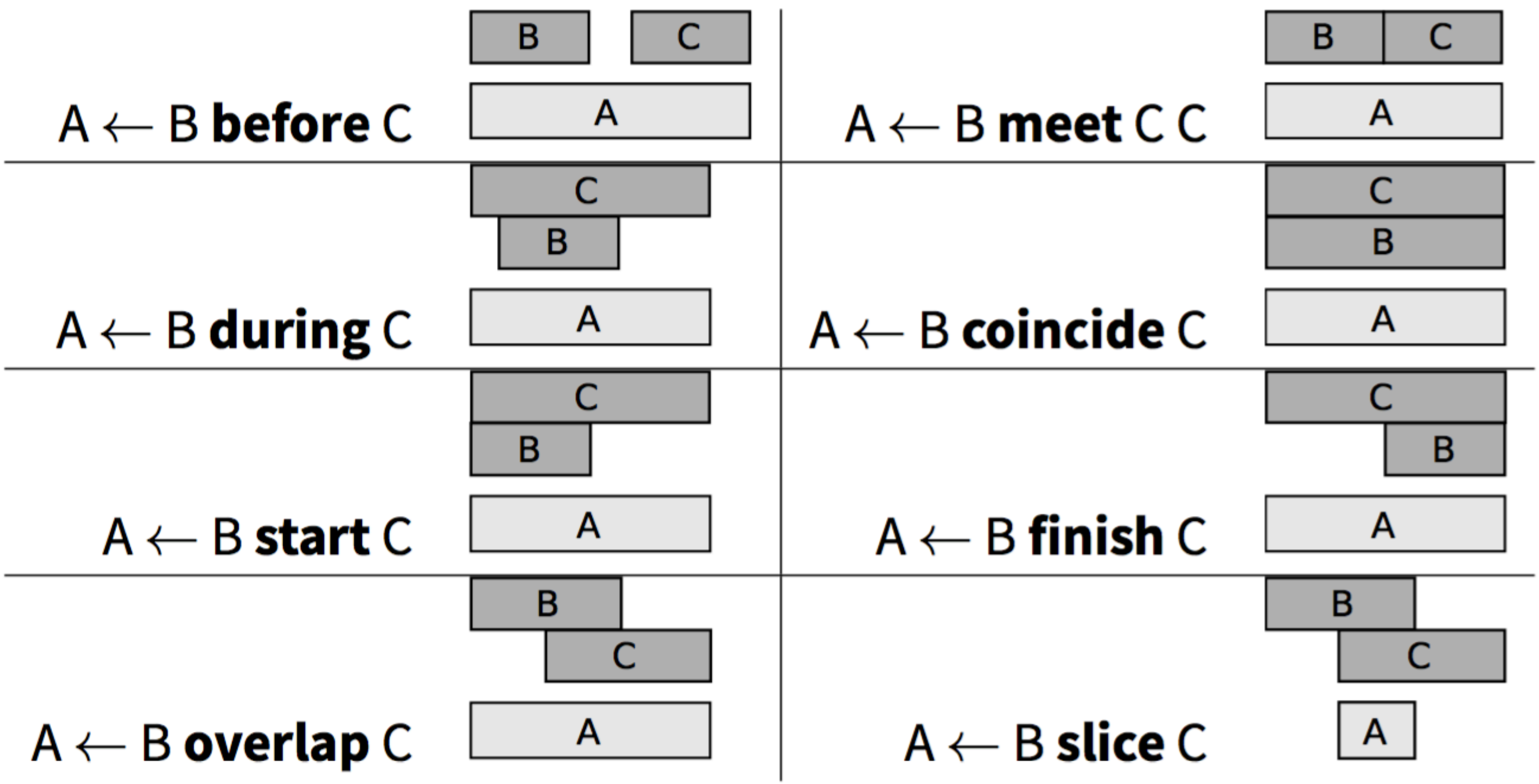
BOOT

BOOT

DBOOT

RISK

Based on Allen Logic



Specification in new DSL

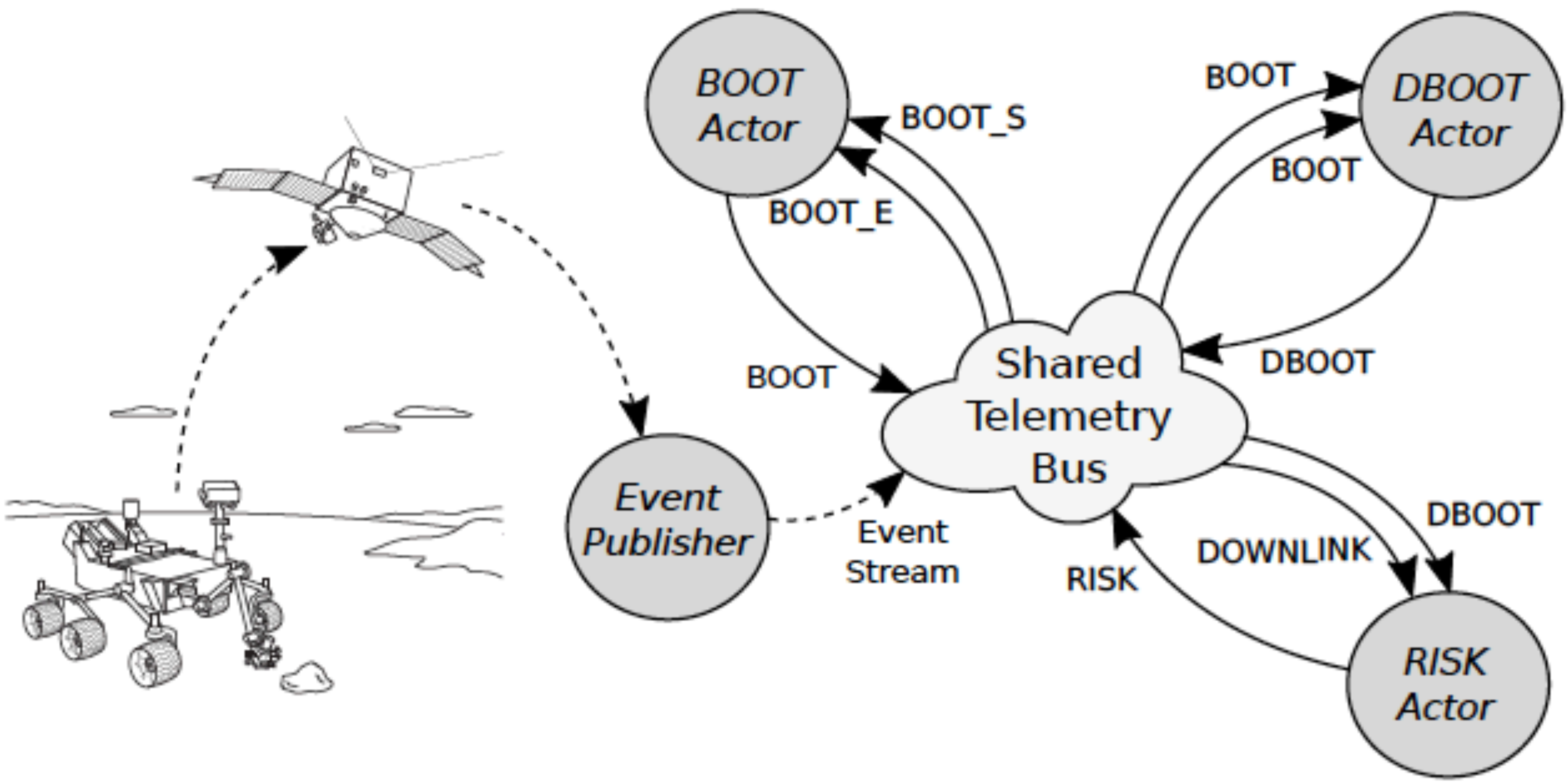
```
BOOT :- BOOT_S before BOOT_E
    map {count -> BOOT_S.count}

DBOOT :- b1:BOOT before b2:BOOT
    where b2.begin - b1.end <= 300
    map {count -> b2.count}

RISK :- DOWNLINK during DBOOT
    map {count -> DBOOT.count}
```

Using modules

```
module Booting {
  BOOT :- BOOT_S before BOOT_E
    map {count -> BOOT_S.count}
}

module DoubleBooting {
  import Booting;

  DBOOT :- b1:BOOT before b2:BOOT
    where b2.begin - b1.end <= 300
    map {count -> b2.count}
}

module Risking {
  import DoubleBooting;

  RISK :- DOWNLINK during DBOOT
    map {count -> DBOOT.count}
}
```

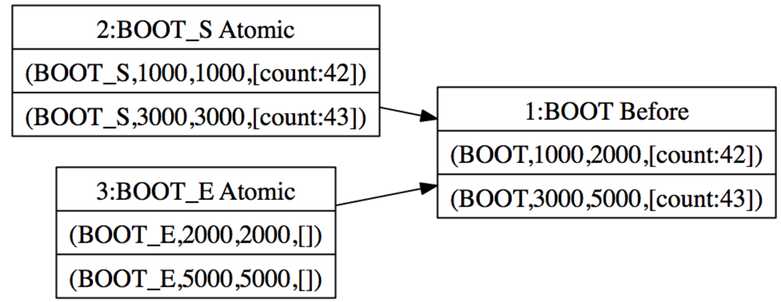
nfer Scala DSL

```
class DoubleBoot extends Nfer {
  "BOOT" :- ("BOOT_S" before "BOOT_E" map {
    case (m1,m2) ⇒ Map("count" → m1("count"))
  })

  "DBOOT" :- ("BOOT" before "BOOT" within 300 map (_._2))

  "RISK" :- ("DOWNLINK" during "DBOOT" map (_._2))
}
```

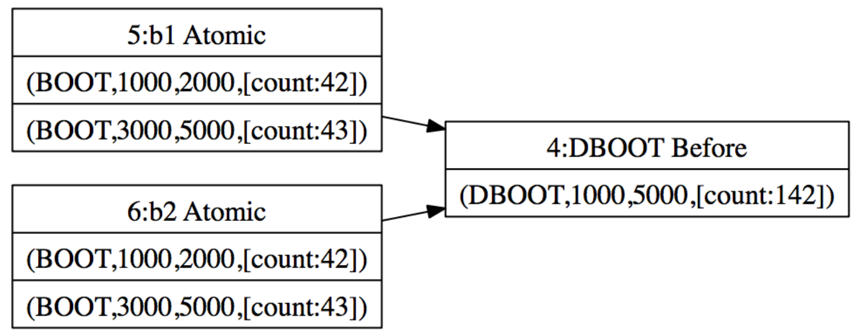

BOOT
Actor
BOOT_S
BOOT_E
BOOT
BOOT
DBOOT
Actor
BOOT
DBOOT
Shared
Telemetry
Bus
Event
Publisher
Event
Stream
DBOOT
DOWNLINK
RISK
RISK
Actor

Implementation of Rules as Trees

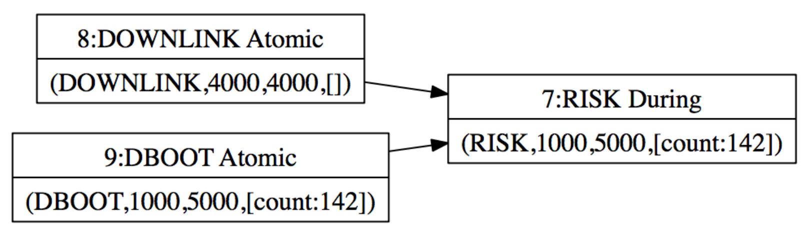
```
BOOT :- BOOT_S before BOOT_E
    where BOOT_E.begin - BOOT_S.end <= 5000
    map {count -> BOOT_S.count}
```




```
DBOOT :- b1:BOOT before b2:BOOT
    where b2.begin - b1.end <= 10000
    map {count -> b1.count}
```




```
RISK :- DOWNLINK during DBOOT
    map {count -> DBOOT.count}
```



#ABCDEFGHIJKLMNOPQRSTUV

display packages only

nfer.repr hide focus

Interval

nfer.system hide focus

Nfer

nfer

package **nfer**

nfer is a Scala package for generating abstractions from *event traces*.

Introduction

nfer allows to define intervals over an event trace. The intervals indicate abstractions that for example can be visualized. An interval consists of a name, two time stamps: the beginning and the end of the interval, and a map from variable names (strings) to values, indicating data that the interval carries.

Writing Specifications

As an example, consider a scenario where we want to be informed if a downlink operation occurs during a 5-minute time interval where the flight computer reboots twice. This scenario could cause a potential loss of downlink information. We want to identify the following intervals.

A BOOT represents an interval during which the rover software is rebooting.

A DBOOT (double boot) represents an interval during which the rover reboots twice within a 5-minute timeframe.

A RISK represents an interval during which the rover reboots twice, and during which also attempts to downlink information.

Our objective now is to formalize the definition of such intervals in the nfer specification language, and extract these intervals from a telemetry stream, based on such a specification. Specifically, in this case, we need a formalism for formally defining the following three kinds of intervals.

A BOOT interval starts with a BOOT_S (boot start) event and ends with a BOOT_E (boot end) event.

A DBOOT (double boot) interval consists of two consecutive BOOT intervals, with no more than 5-minutes from the end of the first BOOT interval to the start of the second BOOT interval.

A RISK interval is a DBOOT interval during which a DOWNLINK occurs.

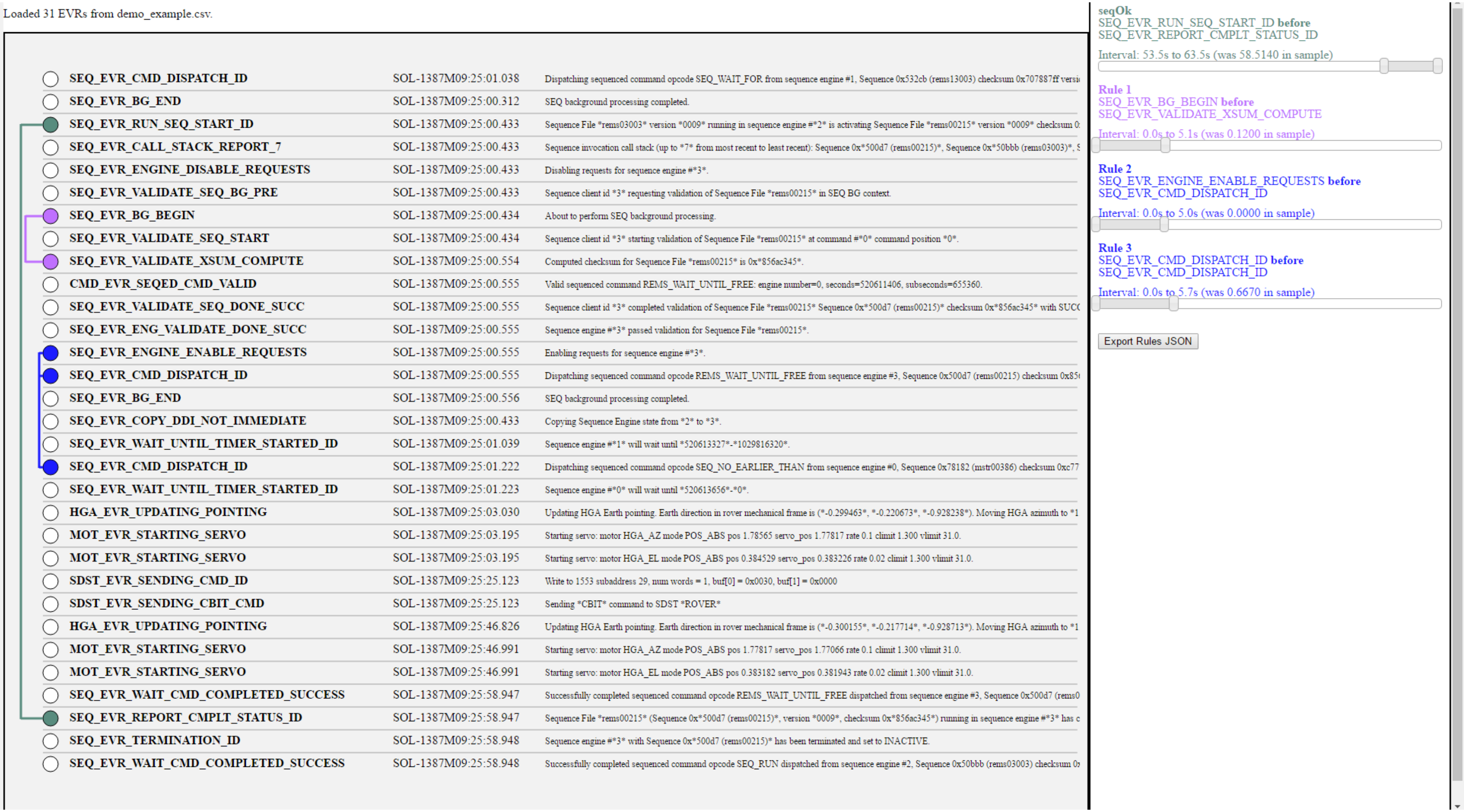
The following time line illustrates a scenario with two BOOTs, hence a double boot, and a downlink occurring during this period.

```
-----|---------|------------|---------|------->
BOOT_S    BOOT_E    |    BOOT_S      BOOT_E
                    |
              DOWNLINK
```

The intervals can be formalized with the following nfer specification:

```
BOOT :- BOOT_S before BOOT_E
  where BOOT_E.begin - BOOT_S.end <= 5000
  map {count -> BOOT_S.count}

DBOOT :- b1:BOOT before b2:BOOT
  where b2.begin - b1.end <= 10000
  map {count -> b1.count}
```

GUI for Entering Rules using a Log



Detailed design and programming by Nathaniel Guy

END