基于Event-B的隔离内核模型及验证技术

赵永望
zhaoyw@buaa.edu.cn
北京航空航天大学 计算机学院

2013年11月

报告提纲

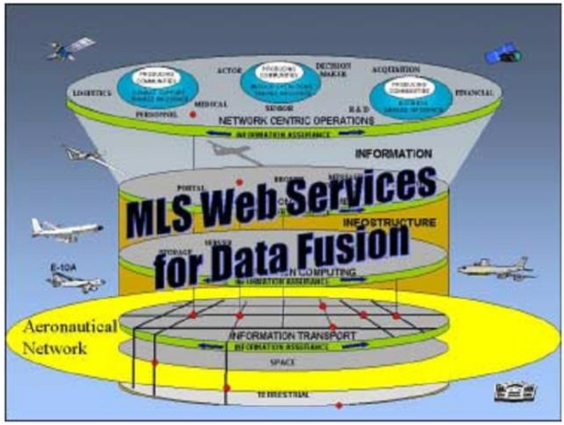
- 1. 什么是隔离内核
- 2. 隔离内核验证的研究现状
- 3. 总体思路
- 4. 隔离内核模型
- 5. 隔离内核验证方法
- 6. 小结

隔离内核 Separation Kernel

- 由美国斯坦福研究院(SRI)的John Rushby于1981年提出

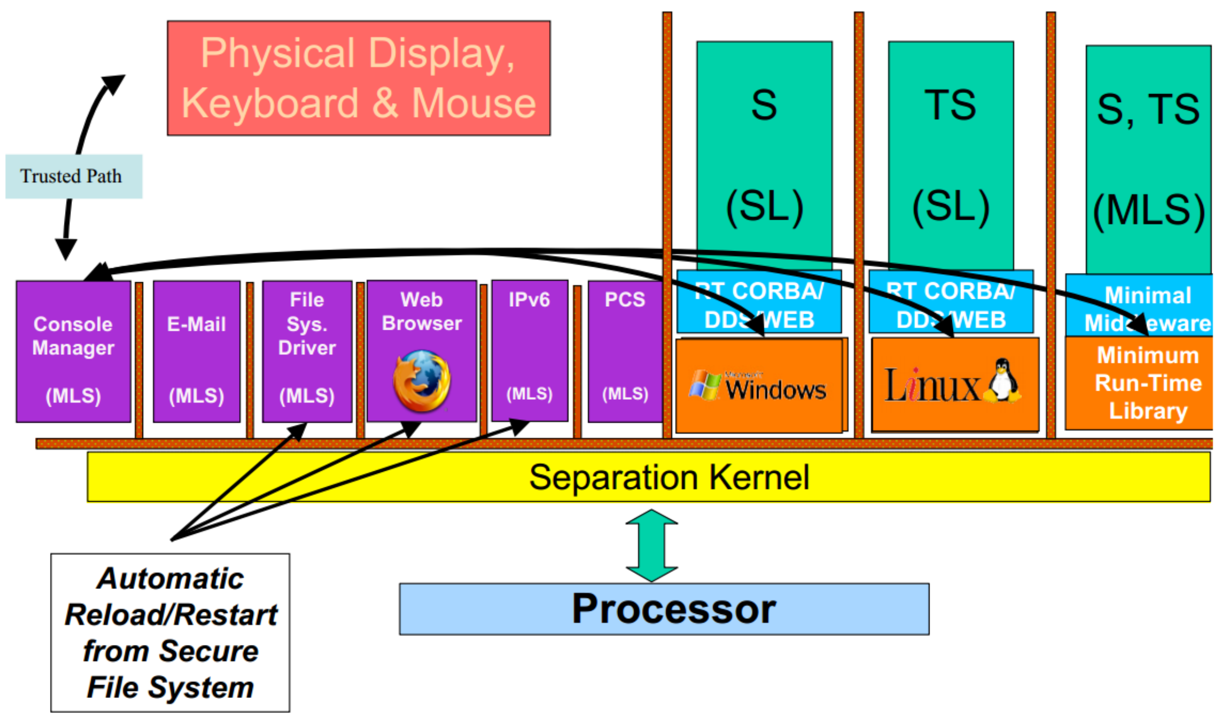
- 最初目的：通过分区将系统的不同部分隔离开，分区之间只能通过可控的方式进行安全通信，以支持各部分（内核、中间件、上层应用）独立验证/认证，而且多级安全应用共存于一个系统中

- 美国空军研究实验室与美国国家安全局 MILS 计划
  - Multiple Independent Levels of Security/Safety
  - 融合Security和Safety相关技术
    - Safety(RTCA DO-178B Level A, ARINC-653)
    - Security(Common Criteria, High Robustness, DCID 6/3 Separation
  - 为武器系统、通信设施、指挥控制平台等提供MSLS /MLS计算、Web和网络服务



MILS架构

- **广泛应用于各类安全关键系统**
  - F22, F35, FCS, JTRS, DDG-1000, CDS
- **包括3层**
  - 隔离内核、中间件和应用



- 隔离内核
  - ✓ 通过分区，隔离进程空间
  - ✓ 分区间数据/信息/控制的安全传输
  - ✓ 规模较小：~4000行代码
- 中间件
  - ✓ 分区内操作系统、库或通信中间件
  - ✓ 提供安全的端到端对象消息流
  - ✓ 维护应用组件
  - ✓ 设备驱动、文件it、网络协议栈、CORBA、DDS等
- 应用
  - ✓ 实现任务功能

隔离内核产品

- 硬件隔离内核 (CPU)
  - Rockwell Collins的AAMP7处理器
- 软件隔离内核(OS)
  - 风河WindRiver的VxWorks MILS平台
  - 绿山Green Hills的Integrity-178B
  - Lynux Works的LynuxOS-178
  - SYSGO的PiksOS

西安631所下一代实时操作系统内核就是MILS架构下的隔离内核(结合Hypervisor)

Common Criteria Evaluation Assurance Levels
- EAL 7: Formally Verified Design and Tested
- EAL 6: Semi-formally Verified Design and Tested
- EAL 5: Semi-formally Designed and Tested
- EAL 4: Methodically Designed, Tested and Reviewed
- EAL 3: Methodically Tested and Checked
- EAL 2: Structurally Tested
- EAL 1: Functionally Tested

序号	产品	CC认证	DO-178B认证
1	AAMP7处理器	EAL 7	Level A
2	VxWorks MILS	EAL 6+	Level A
3	Integrity-178B	EAL 6+	Level A
4	LynuxOS-178	EAL 7	Level A
5	PiksOS	??	??

DO-178B Software Assurance Levels
- Level A: Catastrophic Failure Protection
- Level B: Hazardous/Severe Failure Protection
- Level C: Major Failure Protection
- Level D: Minor Failure Protection
- Level E: Minimal Failure Protection

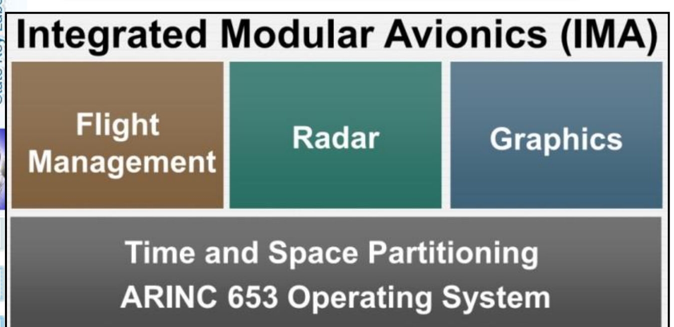
综合化航电 IMA和ARINC653

SWaP需求

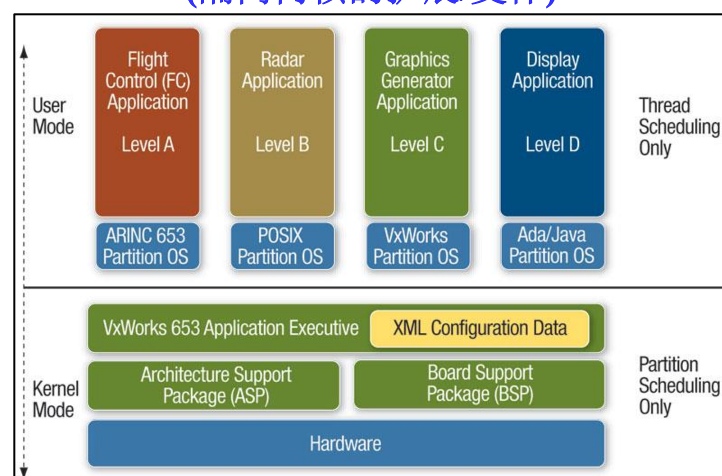
- SWaP: Size, Weight, and Power

分区内核、分区实时操作系统
(隔离内核的扩展/变体)

- ✓ 时空分区
- ✓ 分区间分区内通信
- ✓ 健康监控



MILS架构
应用
隔离内核



隔离内核验证的现状

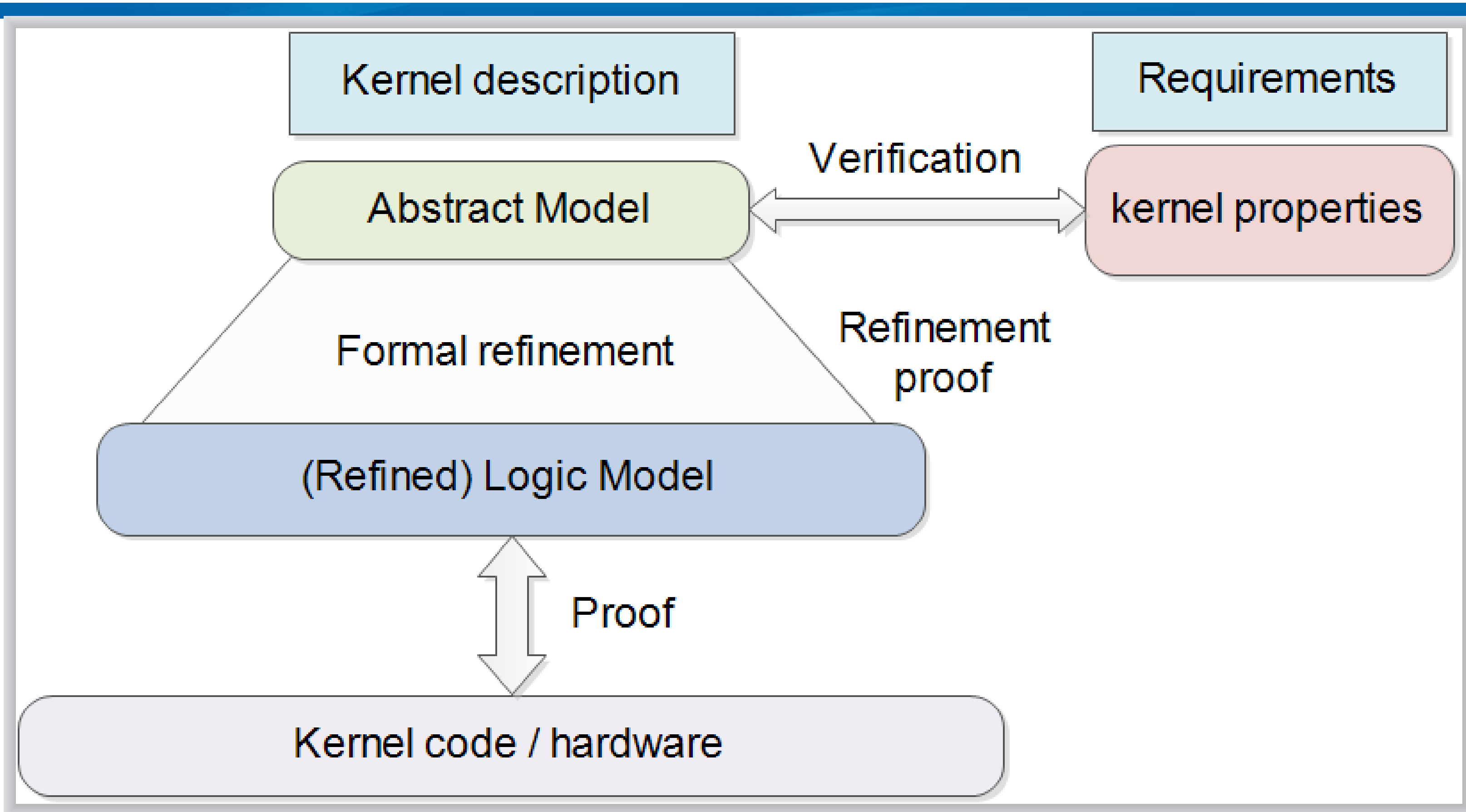
1. Partitioning Kernel Protection Profile (PKPP)
   - 2003年，洛克希德马丁，空军研究实验室，OpenGroup草案
2. Separation Kernel Protection Profile (PKPP)
   - 2007年，美国国家安全局正式颁布
3. 美国国防部：a Mathematically Analyzed Separation Kernel (MASK) 项目
4. Rockwell Collins公司的GWV安全策略及信息流安全验证
5. 美国海军研究中心ED separation kernel形式验证
6. 英国约克大学Grand Challenge Verified Software项目下Separation Kernel形式模型与验证
7. 美国Critical Software公司Partitioning Kernel的B模型及验证
8. 德国Verisoft XT项目对PikeOS的Separation Kernel验证
9. 澳大利亚NICTA：seL4作为Separation Kernel，验证安全性

研究目标

- 针对现有隔离内核只在设计层面、或部分代码层面的形式化验证现状
- 提出一种隔离内核的形式化验证方法
  - 验证内核实现的功能正确性
  - 验证内核实现的关键性质
- 可对隔离内核的源码进行形式化验证
- 方法和模型可重用性强
  - 隔离内核版本升级
  - 同样适用其他隔离内核的验证

隔离内核的安全性质集
隔离内核形式化参考模型

总体思路

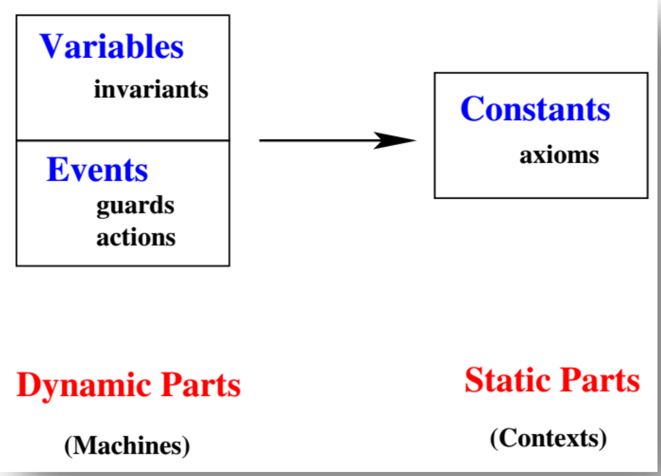


Refinement及证明

- B方法：refinement及proof obligation
- Proof obligation的正确性：定理证明器Coq

Event-B简介

- 由Jean-Raymond Abrial提出
  - 1980s,提出Z语言
  - 1990s,提出B语言
    - 太复杂
  - 2000s, 提出Event-B
    - 替代经典B
- 基于集合论、基本算术、一阶逻辑
- Event-B模型
  - 离散迁移系统：离散状态及迁移(事件触发)
  - 状态：带公理(axiom)的常量(constant)、带不变式(invariant)的变量(variable)
  - 事件：包括卫式(guard)和动作(action)，卫式表明事件触发条件，动作表明状态被修改的方式
  - 常量、变量、卫式和动作均采用集合论表达式来书写



Event-B应用状况

- **Space**
  - **Space Systems Finland**
    - Bepi Colombo (ESAs first mission to Mercury)
    - On-Board Satellite Software: Attitude and Orbit Control System (AOCS)
  - **SSF**
- **Transportation**
  - **Siemens, Systerel: Communication-based Train Control (CBTC)**
  - **Battelle and ClearSy: automation of the Flushing and Culver metro lines in New York**
  - **Alstom: Dynamic Trusted Railway Interlocking**
- **Automotive**
  - **Bosch: Cruise-control system, Engine start-stop system**
- **其他**
  - **SAP: Service choreography modelling**
  - **Critical Software Technologies: Integrated Secondary Flight Display, used on-board commercial and military aircraft; Smart Energy Grids**
  - **XMOS: XCore Microprocessor指令集验证**
  - **The CCF Display and Information System, air traffic controllers based at the London Terminal Control Centre**
  - **STMicroelectronics: modelling and generation of a VHDL code for a smartcard-based microcircuit**
  - **Dependable Systems Forum (DSF) project in Japan (NTT-Data, Fujitsu, Hitachi, NEC, Toshiba, and SCSK)**

Context、Machine和Event

```
context
    < context_identifier >
extends  ⋆
    < context_identifier >
    ...
sets  ⋆
    < set_identifier >
    ...
constants  ⋆
    < constant_identifier >
    ...
axioms  ⋆
    < label >: < predicate >
    ...
theorems  ⋆
    < label >: < predicate >
    ...
end
```

```
machine
    < machine_identifier >
refines  ⋆
    < machine_identifier >
sees  ⋆
    < context_identifier >
    ...
variables
    < variable_identifier >
    ...
invariants
    < label >: < predicate >
    ...
theorems  ⋆
    < label >: < predicate >
    ...
events
    < event >
    ...
variant  ⋆
    < variant >
end
```

```
< event_identifier > ≙
  status
    {normal, convergent, anticipated}
  refines  ⋆
    < event_identifier >
    ...
  any  ⋆
    < parameter_identifier >
    ...
  where  ⋆
    < label >: < predicate >
    ...
  with  ⋆
    < label >: < witness >
    ...
  then  ⋆
    < label >: < action >
    ...
  end
```

Substitution

Kind	Generalized Substitution
Deterministic	$x := E(v)$
Empty	skip
Non-deterministic	**any** t **where** $P(t,v)$ **then** $x := F(t,v)$ **end**

Event-B模型的语义

- **事件的语义**
  - 系统动态行为由卫式命令（事件）表示

WHEN *guard* **THEN** *action* **END**

- **Machine的语义**
  - 系统总体行为是系统事件的无限循环

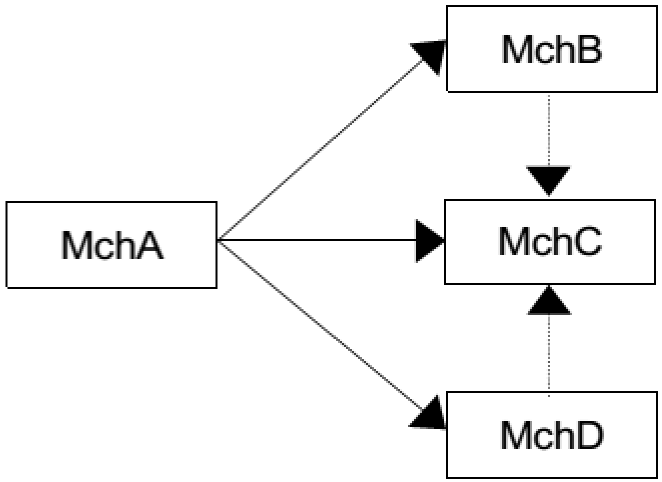
```
forever do
        Event1 or
        Event2 or
        Event3 or ...
end
```

  - 模型不变式定义了一个允许状态的集合，每个事件都必须保持不变式

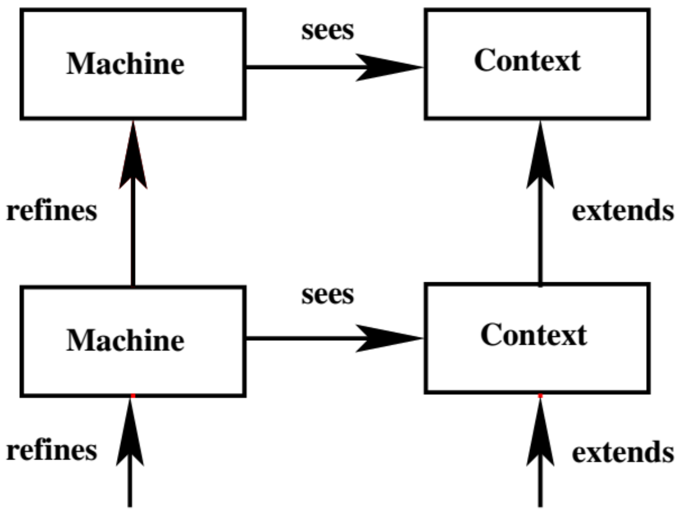
软件开发环境国家重点实验室 State Key Laboratory of Software Development Environment

模型关系

- 经典B方法 – 过于复杂
  - Component : machine, implementation
  - 关系: Include, see, use, import, refine, extend, promote



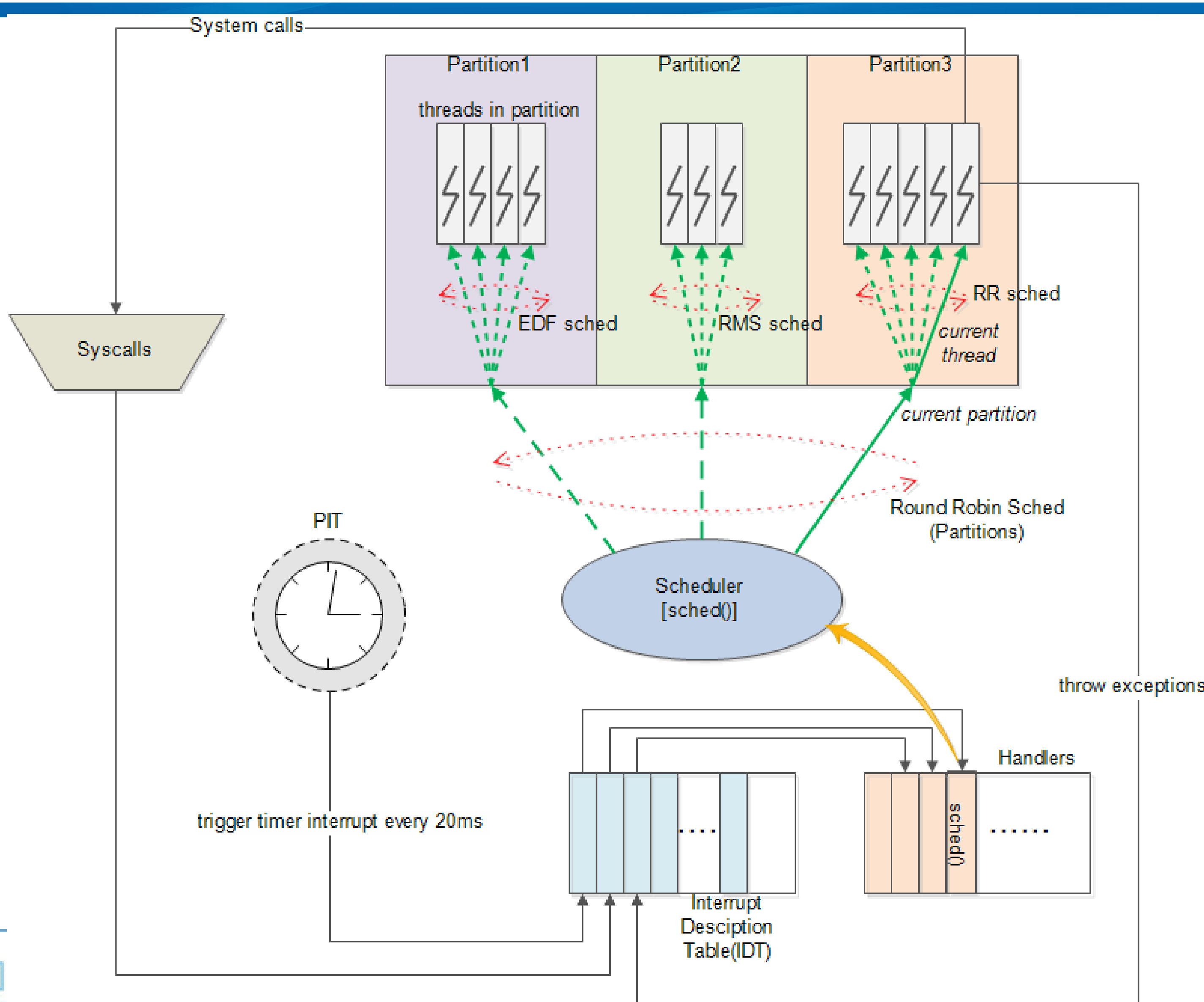
- Event-B方法 – 简洁
  - Component : Context, Machine
  - 关系: extend, see, refine



隔离内核模型

- **隔离内核模型**
  - 分区管理、分区内进程管理、时间管理、内存管理、分区间/分区内通信、健康监控
- **关键性质**
  - Spatial separation
    - 数据隔离
    - 内存隔离
  - Temporal separation
    - 两级调度
  - Information flow control
    - 分区间通信及安全性

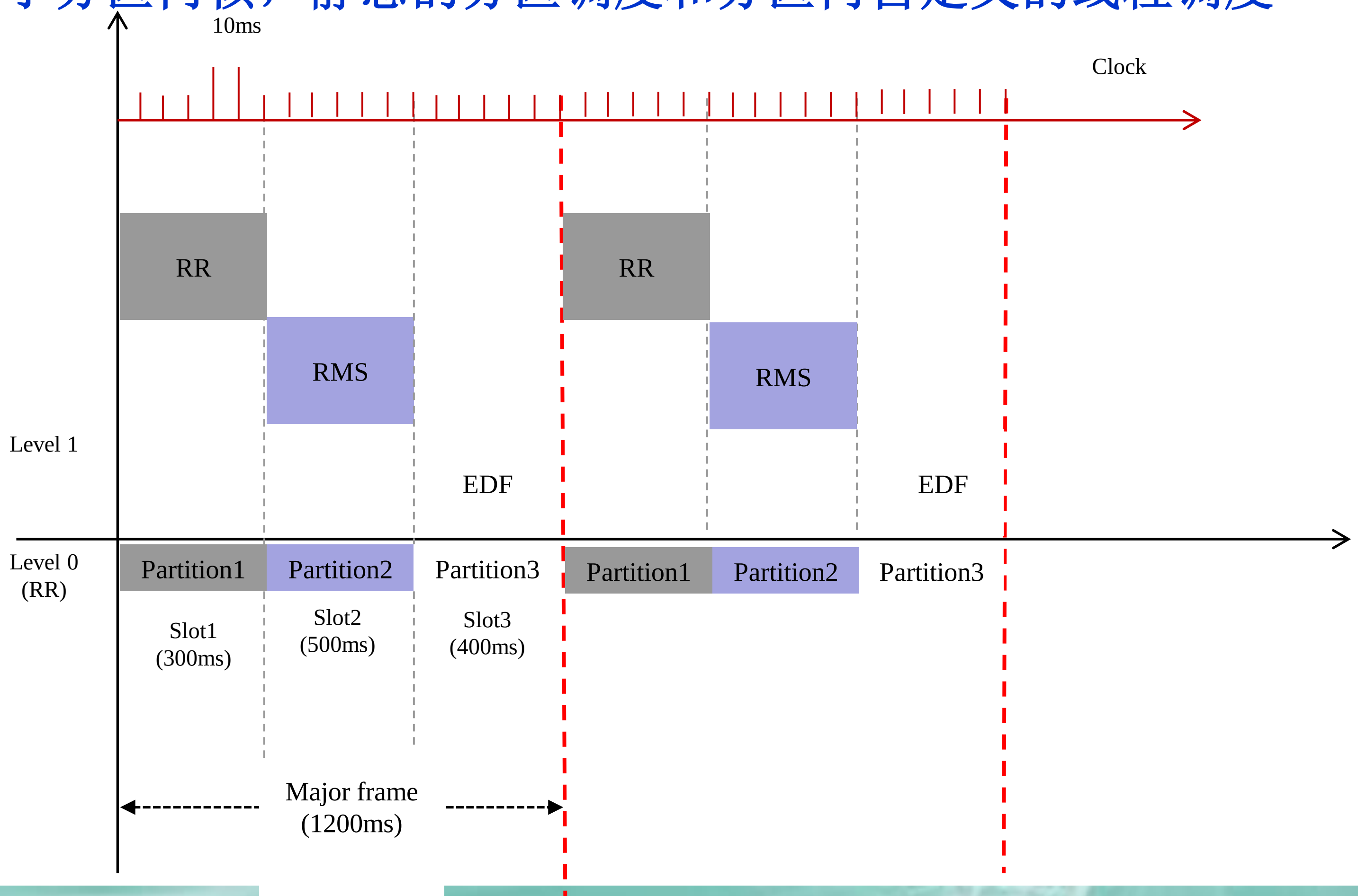
隔离内核执行



System calls
Partition1
Partition2
Partition3
threads in partition
EDF sched
RMS sched
RR sched
current thread
Syscalls
current partition
Round Robin Sched (Partitions)
PIT
Scheduler [sched()]
throw exceptions
Handlers
sched()
trigger timer interrupt every 20ms
Interrupt Desciption Table(IDT)

隔离内核顶层模型

- 从顶层的视角，隔离内核的实质是**分区管理**和**分区调度**
  - 分区管理包括分区内线程，调度是两级调度
  - 对于分区内核，静态的分区调度和分区内自定义的线程调度



隔离内核模型的求精过程

refinement

顶层模型
- 分区、线程和调度

第1层
- 分区、线程状态及转换

第2层
- 两级调度

第3层
- 分区、线程管理

第4层
- 分区、线程及数据的内存模型

第5层
- 分区间/分区内通信

第6层
- 健康监控

顶层Event-B模型

```
context Kernel_c01

sets Partitions
    AllThreads // the thread including used and unused threads
    PartitionModes ThreadStates

constants PM_IDLE PM_NORMAL PM_WARM_START
        PM_COLD_START // partition operating mode, according to ARINC653
        TS_Dormant TS_Ready TS_Waiting TS_Suspend TS_WaitandSuspend TS_Running

axioms
  @axm00 finite(Partitions) // ∧ finite(Threads)
  @axm01 partition(PartitionModes,{PM_IDLE}, {PM_NORMAL}, {PM_WARM_START},{PM_COLD_START})
  @axm02 partition(ThreadStates,{TS_Dormant}, {TS_Ready}, {TS_Waiting}, {TS_Suspend},
                         {TS_WaitandSuspend}, {TS_Running})
  @axm03 card(Partitions) ≥ 2 // at least two partition, the kernal partition and a
application partition
end
```

顶层Event-B模型

```
machine Kernel_m01
sees Kernel_c01

variables Threads
        threadsOfPartition partitionstate
        threadstate
        current_partition
        current_thread

invariants
  @inv00 Threads ∈ AllThreads → BOOL
  @inv1 partitionstate ∈ Partitions → PartitionModes
  @axiom1 threadsOfPartition ∈ AllThreads ↠ Partitions
  @inv2 threadstate ∈ AllThreads → ThreadStates
  @inv3 current_partition ∈ Partitions
  @inv4 current_thread ∈ AllThreads ∧ Threads(current_thread) = TRUE ∧
threadsOfPartition(current_thread) = current_partition
  @inv31 partitionstate(current_partition) = PM_NORMAL
  @inv41 threadstate(current_thread) = TS_Running
  @inv5 card(partitionstate ▷ {PM_NORMAL}) ≥ 1
  @inv6 card(threadstate ▷ {TS_Running}) ≤ 1
```

软件开发环境国家重点实验室
State Key Laboratory of Software Development Environment

顶层Event-B模型

```
event schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
  then
    @act1 threadstate ≔ (threadstate   (threadstate∼[{TS_Running}] × {TS_Ready}))   {t ↦ TS_Running}
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

第2层Event-B模型

在顶层模型基础上，增加分区和线程的状态转换

```
event schedule refines schedule
  any p t
  where
   @axm1 p ∈ Partitions
   @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
   @axm2 threadsOfPartition(t) = p
   @axm3 partitionstate(p) = PM_NORMAL
   @axm4 threadstate(t) = TS_Ready
   @axm5 t ≠ current_thread
  then
   @act0 threadstate :| threadstate'(t) = TS_Running ∧ threadstate'(current_thread) = TS_Ready
   @act5 current_partition ≔ p
   @act6 current_thread ≔ t
 end
event partition_cold_restart
end
 event partition_warm_restart
end
 event partition_go
end
 event partition_shutdown
end
 event thread_ready // put a thread into READY except the situation of runing --> ready, which has been in the event of "schedule"
end
 event thread_stop // put a thread into DORMANT
end
 event thread_stopself // put a thread into DORMANT
end
 event thread_waiting // put a thread into WAITING
end
 event thread_waitingfrmruning // put a thread into WAITING
end
 event thread_suspend
end
 event thread_suspendself
end
 event thread_waitandsuspend
end
```

第3层Event-B模型

- 在第2层模型基础上，增加两级调度

```
context Kernel_c03
/*two level scheduling with defined partition time frame, we also add clock*/
extends Kernel_c01

constants
schedulingFrameofPartition /*the scheduling time frame of each partition*/
schedulingFrames /*the scheduling time frames*/
majorFrame /*the total time of all partitions*/

axioms
@axm00 schedulingFrames ∈ ℙ1(ℕ×ℕ) /*each <x |->y> : schedulingFrames, x is the start time of the frame and y is the end time*/
@axm01 ∀x•(x ∈ dom(schedulingFrames) ⇒ x ≤ schedulingFrames(x)) /*the end time should be larger than the start time of each frame*/
@axm02 ∀x,y•(x ∈ dom(schedulingFrames) ∧ y ∈ dom(schedulingFrames) ⇒ (x ≥schedulingFrames(y) ∨ y ≥ schedulingFrames(x) )) /*scheduling frames are disjoint*/
@axm03 schedulingFrameofPartition ∈ schedulingFrames ↠ Partitions /*in a major frame, each partition should have one or more than one scheduling frame*/
@axm04 majorFrame ∈ ℕ1 /*majorFrame should be larger than sum of all schedulingframe. there may be spare time space between scheduling frames*/

end
```

第3层Event-B模型

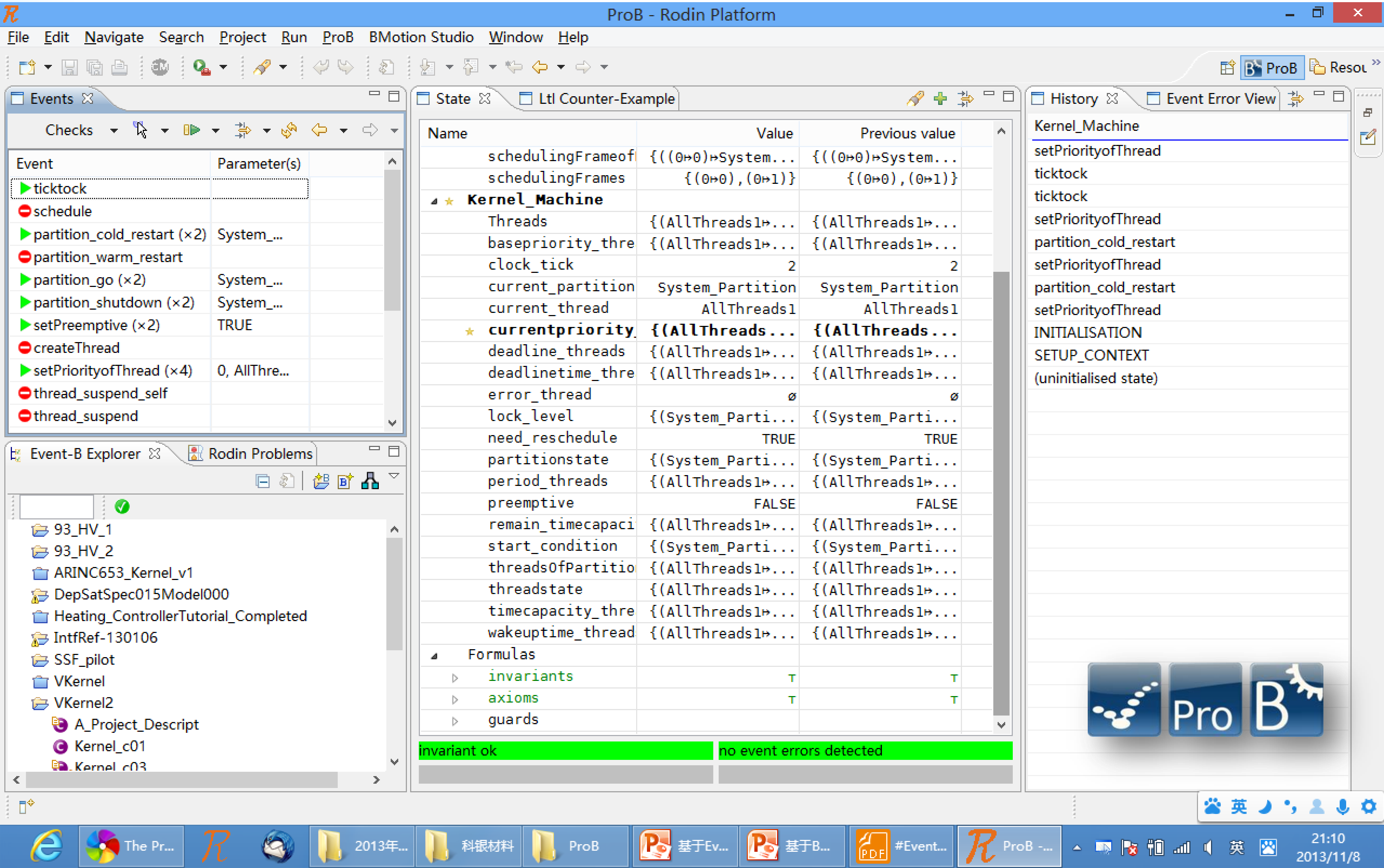
- 在第2层模型基础上，增加两级调度

```
event schedule extends schedule
 when
    @axm6 ∃x,y,n•(x∈ℕ1 ∧ y ∈ ℕ1 ∧ n ∈ ℕ1 ∧ x↦y ∈ schedulingFrames
    ∧ schedulingFrameofPartition(x ↦y) = p
    ∧ (x + n*majorFrame) < clock_tick ∧ clock_tick < (y + n*majorFrame))
/*current tick is in the scheduling frame of the partition*/
 end
```

第4层Event-B模型

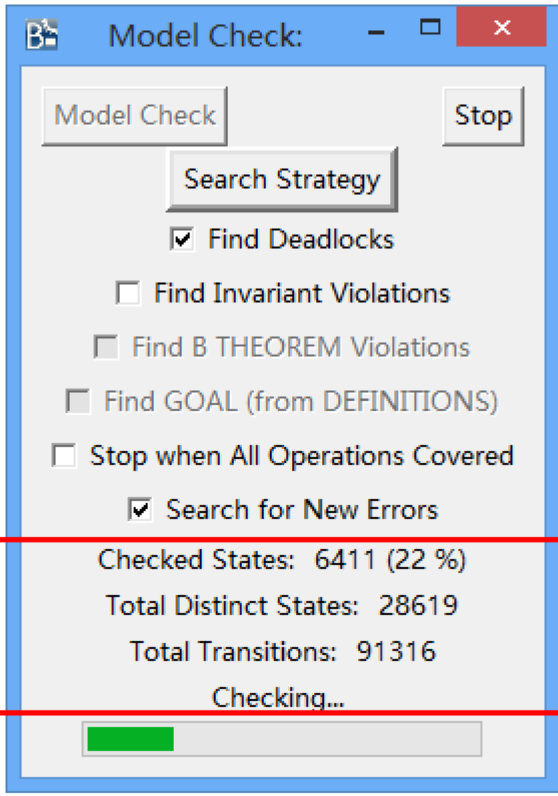
- 在第3层模型基础上，增加分区管理和线程管理

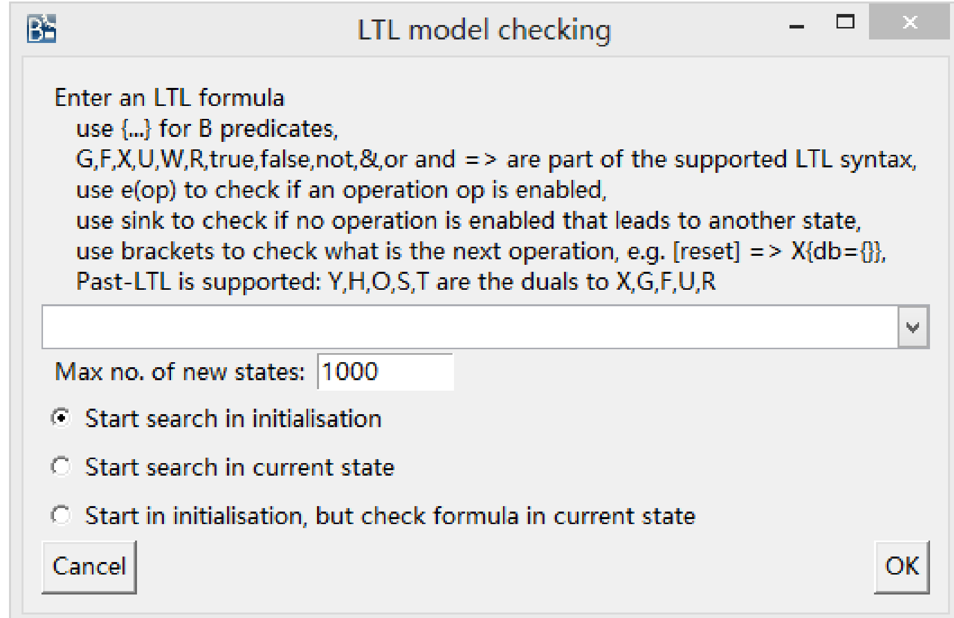
隔离内核的验证—模拟

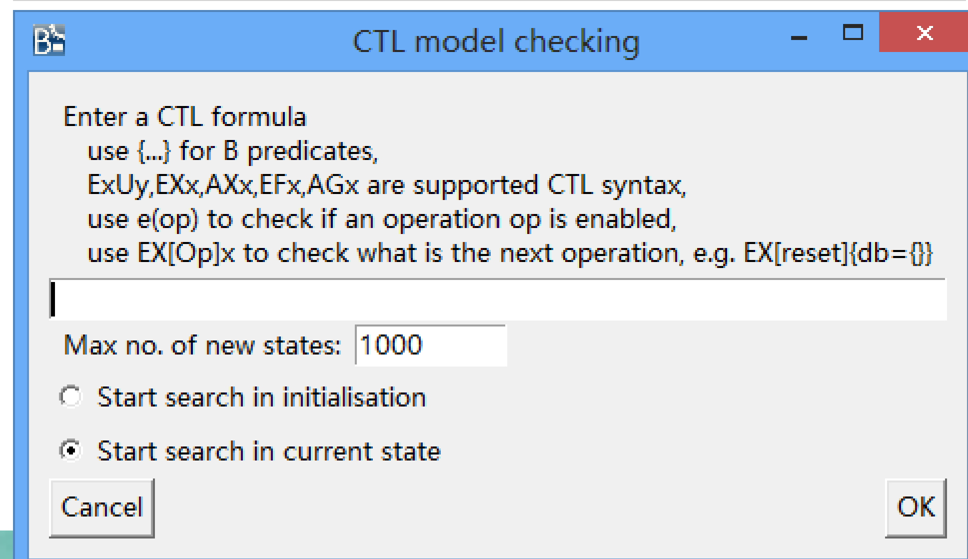


ProB - Rodin Platform
File Edit Navigate Search Project Run ProB BMotion Studio Window Help
Events
Checks
Event
Parameter(s)
ticktock
schedule
partition_cold_restart (×2)
System_...
partition_warm_restart
partition_go (×2)
System_...
partition_shutdown (×2)
System_...
setPreemptive (×2)
TRUE
createThread
setPriorityofThread (×4)
0, AllThre...
thread_suspend_self
thread_suspend
Event-B Explorer
Rodin Problems
93_HV_1
93_HV_2
ARINC653_Kernel_v1
DepSatSpec015Model000
Heating_ControllerTutorial_Completed
IntfRef-130106
SSF_pilot
VKernel
VKernel2
A_Project_Descript
Kernel_c01
Kernel_c03
State
Ltl Counter-Example
Name
Value
Previous value
schedulingFrameof
schedulingFrames
Kernel_Machine
Threads
basepriority_thre
clock_tick
2
current_partition
System_Partition
current_thread
AllThreads1
currentpriority_
deadline_threads
deadlinetime_thre
error_thread
lock_level
need_reschedule
TRUE
partitionstate
period_threads
preemptive
FALSE
remain_timecapaci
start_condition
threadsOfPartitio
threadstate
timecapacity_thre
wakeuptime_thread
Formulas
invariants
axioms
guards
invariant ok
no event errors detected
History
Event Error View
Kernel_Machine
setPriorityofThread
ticktock
ticktock
setPriorityofThread
partition_cold_restart
setPriorityofThread
partition_cold_restart
setPriorityofThread
INITIALISATION
SETUP_CONTEXT
(uninitialised state)
Pro B
ProB
Resou
The Pr...
2013年...
科银材料
ProB
基于Ev...
基于B...
#Event...
ProB -...
21:10
2013/11/8

隔离内核的验证—模型检测







隔离内核的验证—演绎推理(Deductive reasoning)

- **正确性的必要条件**
  - 需证明的命题，也称证据(proof)命题/证据义务(proof obligation)
  - 需证明的证据命题，可由Event-B工具自动生成

- **对于一个Machine**
  - 事件的可行性Feasibility
  - 事件的不变式保持Invariant preservation
  - Machine的无死锁deadlockfree

- **对于一个Refined Machine**
  - 事件的可行性Feasibility
  - 事件的不变式保持Invariant preservation
  - Machine的无死锁deadlockfree
  - 事件的求精正确性Correct refinement
    - 事件的卫式条件被加强或保持不变
    - 事件的动作模拟被求精事件的动作（求精后每个迁移都被抽象模型所允许）
    - 使用关联不变式(gluing invariant)来连接新旧模型的状态

可行性

- s: sets
- c: constants
- v: event执行之前的变量值
- v’: event执行之后的变量值

- P(s, c): 常量的性质 ---- context中的axioms
- I (s, c, v): 不变式---- machine中的invariants
- G(s, c, v): 事件的卫式条件 ---- event起始的where条件
- R(s, c, v, v’): 事件的前后谓词 ---- event执行前后保持的性质

软件开发环境国家重点实验室 State Key Laboratory of Software Development Environment

举例

```
context Kernel_c01

sets Partitions
    AllThreads // the thread including used and unused threads
    PartitionModes ThreadStates

constants PM_IDLE PM_NORMAL PM_WARM_START
        PM_COLD_START // partition operating mode, according to ARINC653
        TS_Dormant TS_Ready TS_Waiting TS_Suspend TS_WaitandSuspend TS_Running

axioms
  @axm00 finite(Partitions) // ∧ finite(Threads)
  @axm01 partition(PartitionModes,{PM_IDLE}, {PM_NORMAL}, {PM_WARM_START},{PM_COLD_START})
  @axm02 partition(ThreadStates,{TS_Dormant}, {TS_Ready}, {TS_Waiting}, {TS_Suspend},
                        {TS_WaitandSuspend}, {TS_Running})
  @axm03 card(Partitions) ≥ 2 // at least two partition, the kernal partition and a
application partition
end
```

举例

```
machine Kernel_m01
sees Kernel_c01

variables Threads
        threadsOfPartition partitionstate
        threadstate
        current_partition
        current_thread

invariants
  @inv00 Threads ∈ AllThreads → BOOL
  @inv1 partitionstate ∈ Partitions → PartitionModes
  @axiom1 threadsOfPartition ∈ AllThreads ↠ Partitions
  @inv2 threadstate ∈ AllThreads → ThreadStates
  @inv3 current_partition ∈ Partitions
  @inv4 current_thread ∈ AllThreads ∧ Threads(current_thread) = TRUE ∧
threadsOfPartition(current_thread) = current_partition
  @inv31 partitionstate(current_partition) = PM_NORMAL
  @inv41 threadstate(current_thread) = TS_Running
  @inv5 card(partitionstate ▷ {PM_NORMAL}) ≥ 1
  @inv6 card(threadstate ▷ {TS_Running}) ≤ 1
```

可行性

event schedule

  any *p t*

  where

   @axm1 *p* ∈ *Partitions*

   @axm11 *t* ∈ *AllThreads* ∧ *Threads*(*t*) = *TRUE*

   @axm2 threadsOfPartition(*t*) = *p*

   @axm3 partitionstate(*p*) = *PM_NORMAL*

  then

   @act1 threadstate ≔ (threadstate   (threadstate∼[{TS_Running}]   {TS_Ready}))   {*t* ↦ *TS_Running*}

   @act5 current_partition ≔ *p*

   @act6 current_thread ≔ *t*

end

P(s, c)
I (s, c, v)
G(s, c, v)

∃v’:R(s, c, v, v’)=T

不变式保持

```
event schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
  then
    @act1 threadstate ≔ (threadstate   (threadstate∼[{TS_Running}]   TS_Ready}))   {t ↦
TS_Running}
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

P(s, c)
I (s, c, v)
G(s, c, v)
R(s, c, v, v')

I (s, c, v')

无死锁

```
event schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
  then
    @act1 threadstate ≔ (threadstate    (threadstate∼[{TS_Running}] × {TS_Ready}))    {t ↦ TS_Running}
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

P(s, c)
I (s, c, v)

↓

G(s, c, v)

可行性和不变式保持的命题

- 对于一般性事件，证明可行性和不变式保持的两个命题

$P(s,c) \land I(s,c,v) \land G(s,c,v) \Rightarrow \exists v' \cdot R(s,c,v,v')$	FIS
$P(s,c) \land I(s,c,v) \land G(s,c,v) \land R(s,c,v,v') \Rightarrow I(s,c,v')$	INV

  - P(s, c)常量的性质
  - I (s, c, v)不变式
  - G(s, c, v)事件的卫式条件
  - R(s, c, v, v')事件的前后谓词: 事件执行后保持的性质

Generalized Substitution	Before-after Predicate
$x := E(v)$	$x' = E(v) \land y' = y$
skip	$v' = v$
any t **where** $P(t,v)$ **then** $x := F(t,v)$ **end**	$\exists t \cdot (P(t,v) \land x' = F(t,v)) \land y' = y$

可行性和不变式保持命题

- 对于一个Machine的初始化事件，证明可行性和不变式保持的两个命题

$P(s,c) \;\Rightarrow\; \exists v' \cdot RI(s,c,v')$	INI_FIS
$P(s,c) \wedge RI(s,c,v') \;\Rightarrow\; I(s,c,v')$	INI_INV

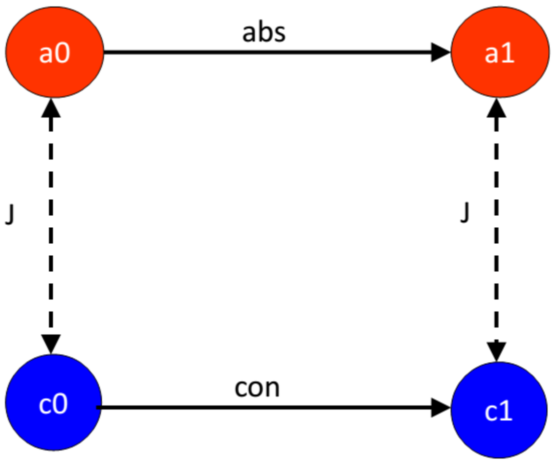
无死锁的命题

- 对于一个Machine，无死锁的命题

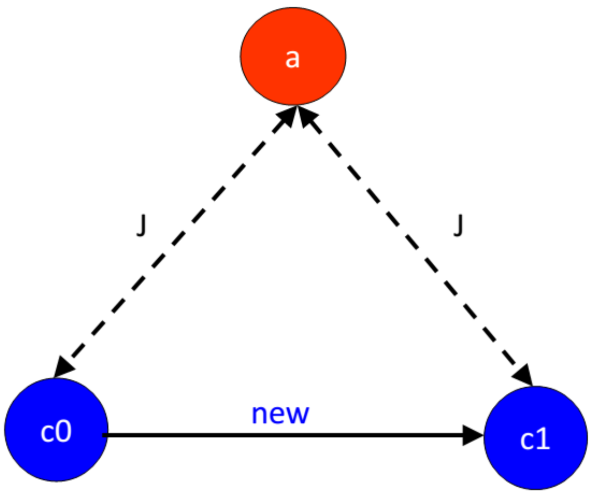
$P(s,c) \wedge I(s,c,v) \Rightarrow G_1(s,c,v) \vee \ldots \vee G_n(s,c,v)$	DLKF

事件求精的类型

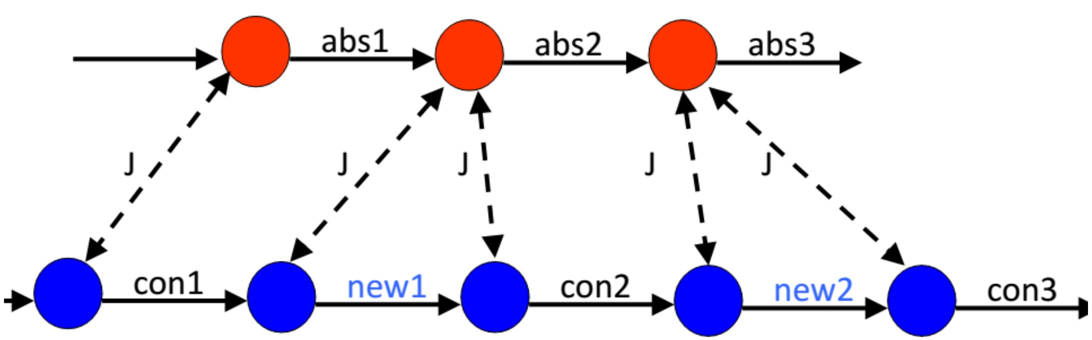
Simulation: maintaining a gluing relation



New concrete events refine skip (stuttering step)



Refining traces



求精的正确性

```
event schedule
   any p t
   where
     @axm1 p ∈ Partitions
     @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
     @axm2 threadsOfPartition(t) = p
     @axm3 partitionstate(p) = PM_NORMAL
   then
     @act1 threadstate ≔ (threadstate     (threadstate~[{TS_Running}] × {TS_Ready}))     {t ↦ TS_Running}
     @act5 current_partition ≔ p
     @act6 current_thread ≔ t
end
```

- ✓ P(s, c)常量的性质
- ✓ I (s, c, v)不变式
- ✓ G(s, c, v): 抽象事件的卫式条件
- ✓ R(s, c, v, v’): 抽象事件的前后谓词
- ✓ J (s, c, v, w): 关联(gluing)不变式

```
event schedule refines schedule
   any p t
   where
     @axm1 p ∈ Partitions
     @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
     @axm2 threadsOfPartition(t) = p
     @axm3 partitionstate(p) = PM_NORMAL
     @axm4 threadstate(t) = TS_Ready
     @axm5 t ≠ current_thread
   then
     @act0 threadstate :| threadstate'(t) = TS_Running ∧ threadstate'(current_thread) = TS_Ready
     @act5 current_partition ≔ p
     @act6 current_thread ≔ t
end
```

- ✓ H(s, c, w): 具体事件的卫式条件
- ✓ S(s, c, w, w’): 具体事件的前后谓词

求精的正确性—可行性

```
event schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
  then
    @act1 threadstate ≔ (threadstate   (threadstate~[{TS_Running}] × {TS_Ready}))   {t ↦ TS_Running}
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

- ✓ P(s, c)常量的性质
- ✓ I (s, c, v)不变式
- ✓ G(s, c, v): 抽象事件的卫式条件
- ✓ R(s, c, v, v'): 抽象事件的前后谓词
- ✓ J (s, c, v, w): 关联(gluing)不变式

```
event schedule refines schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
    @axm4 threadstate(t) = TS_Ready
    @axm5 t ≠ current_thread
  then
    @act0 threadstate :| threadstate'(t) = TS_Running ∧ threadstate'(current_thread) = TS_Ready
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

- ✓ H(s, c, w): 具体事件的卫式条件
- ✓ S(s, c, w, w'): 具体事件的前后谓词

P(s, c)
I (s, c, v)
J (s, c, v, w)
H(s, c, w)

S(s, c, w, w')

求精的正确性—卫式保持

```
event schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
  then
    @act1 threadstate ≔ (threadstate    (threadstate~[{TS_Running}] × {TS_Ready}))    {t ↦ TS_Running}
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

- ✓ P(s, c)常量的性质
- ✓ I (s, c, v)不变式
- ✓ G(s, c, v): 抽象事件的卫式条件
- ✓ R(s, c, v, v'): 抽象事件的前后谓词
- ✓ J (s, c, v, w): 关联(gluing)不变式

P(s, c)
I (s, c, v)
J (s, c, v, w)
H(s, c, w)

↓

G(s, c, v)

```
event schedule refines schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
    @axm4 threadstate(t) = TS_Ready
    @axm5 t ≠ current_thread
  then
    @act0 threadstate :| threadstate'(t) = TS_Running ∧ threadstate'(current_thread) = TS_Ready
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

- ✓ H(s, c, w): 具体事件的卫式条件
- ✓ S(s, c, w, w'): 具体事件的前后谓词

求精的正确性—不变式保持

```
event schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
  then
    @act1 threadstate ≔ (threadstate    (threadstate~[{TS_Running}] × {TS_Ready}))    {t ↦  TS_Running}
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

- ✓ P(s, c)常量的性质
- ✓ I (s, c, v)不变式
- ✓ G(s, c, v): 抽象事件的卫式条件
- ✓ R(s, c, v, v'): 抽象事件的前后谓词
- ✓ J (s, c, v, w): 关联(gluing)不变式

```
event schedule refines schedule
  any p t
  where
    @axm1 p ∈ Partitions
    @axm11 t ∈ AllThreads ∧ Threads(t) = TRUE
    @axm2 threadsOfPartition(t) = p
    @axm3 partitionstate(p) = PM_NORMAL
    @axm4 threadstate(t) = TS_Ready
    @axm5 t ≠ current_thread
  then
    @act0 threadstate :| threadstate'(t) = TS_Running ∧ threadstate'(current_thread) = TS_Ready
    @act5 current_partition ≔ p
    @act6 current_thread ≔ t
end
```

- ✓ H(s, c, w): 具体事件的卫式条件
- ✓ S(s, c, w, w'): 具体事件的前后谓词

P(s, c)
I (s, c, v)
J (s, c, v, w)
H(s, c, w)
S(s, c, w, w')

$\exists v':(R(s, c, v, v') \wedge J(s, c, v', w'))$

求精的正确性

- 事件求精正确的命题(具体事件refines抽象事件)

$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w)$ $\Rightarrow$ $\exists w' \cdot S(s,c,w,w')$	FIS_REF
$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w)$ $\Rightarrow$ $G(s,c,v)$	GRD_REF
$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w) \wedge S(s,c,w,w')$ $\Rightarrow$ $\exists v' \cdot (R(s,c,v,v') \wedge J(s,c,v',w'))$	INV_REF

- G(s, c, v): 抽象事件的卫式条件
- H(s, c, w): 具体事件的卫式条件
- J (s, c, v, w): 关联不变式
- R(s, c, v, v’): 抽象事件的前后谓词
- S(s, c, w, w’): 具体事件的前后谓词

求精的正确性

- **新事件正确性的命题**
  - 每个新事件是对隐式的skip动作的求精
  - 新增的所有事件，必须互不偏离(diverge)

$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w)$ $\Rightarrow$ $\exists w' \cdot S(s,c,w,w')$	FIS_REF
$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w) \wedge S(s,c,w,w')$ $\Rightarrow$ $J(s,c,v,w')$	INV_REF

$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge H(s,c,w) \wedge S(s,c,w,w')$ $\Rightarrow$ $V(s,c,w) \in \mathbb{N} \wedge V(s,c,w') < V(s,c,w)$	WFD_REF

求精的正确性

- **新事件无死锁的命题**
  - 弱性无死锁

$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge G_i(s,c,v)$ $\Rightarrow$ $H_1(s,c,w) \vee \ldots \vee H_m(s,c,w) \vee N_1(s,c,w) \vee \ldots \vee N_n(s,c,w)$	W_DLK_E$_i$

  - 强性无死锁

$P(s,c) \wedge I(s,c,v) \wedge J(s,c,v,w) \wedge G_i(s,c,v)$ $\Rightarrow$ $H_i(s,c,w) \vee N_1(s,c,w) \vee \ldots \vee N_n(s,c,w)$	S_DLK_E$_i$

小结

- 完成了
  - 隔离内核的Event-B模型
  - 基于模拟和模型检测方法的验证
- 下一步工作
  - 采用演绎推理的方式验证隔离内核的Event-B模型

- Event-B建模思路与软件的区别

- Event-B中，不变式保持以事件为最小粒度

谢谢！

请批评指正！