Applications of Advanced Technologies for High Integrity

L.Georgieva
lilia@macs.hw.ac.uk
Dependable Systems Group
Heriot Watt University
Edinburgh, UK

What is integrity?

- Ensuring system correctness

  If the user supplies expected input, system generates desired output

- Ensuring security

  If an attacker supplies unexpected input, system does not fail in certain ways

My research area

- Understanding of the issues involved in building high integrity software intensive systems.
- Providing both practical and theoretical insights into industrial strength tools and techniques that promote the development and modelling of high integrity software intensive systems.
- Reasoning techniques in AI with focus on high integrity.

Relevance of integrity

We consider cases where high integrity is needed:

- Security
- Data, knowledge and process management
- Meaning and disambiguation
- Achieving program efficiency and correctness
- Data transmission over a network and network modelling.

Today's talk

Tools for model analysis:

- Alloy
- Spin
- UPPAAL

Dynamic versus static models

- Security as code – a model where we define our Cyber security policies as Code and not as a static rule.
- Three areas of focus for Security as Code:
  - Security Testing
  - Vulnerability Scanning
  - Access Control and Policy Management
- Refers to various routines, subroutines and scripts.

Dynamic models in Alloy

- Basics of dynamic models
  - Modeling a system's states and state transitions
    - Modeling operations causing transitions
- Simple example of operations

Static Models

Alloy is used to define the allowable values of state components

- values of sets
- values of relations

- A model instance is a set of state component values that

- Satisfies the constraints defined by multiplicities, fact, “realism” conditions, …

Dynamic Models

- Static models allow us to describe the legal states of a dynamic system
- We also want to be able to describe the legal transitions between states

E.g.

- A message is sent before it can be received.

Example

Message Model

```
abstract sig Person {
    senders: set Person,
    recipients: set Person
}

sig Sender, Receiver extends Person {}

sig Signatory in Person {
    signed: one Signatory
}
```

Modeling State Transitions

- Alloy does not have an embedded notions of state transition.
- However, there are several ways to model dynamic aspects of a system
- A a general and relatively simple one is to:
  - introduce a Time signature expressing time and
  - add a time component to each relation that changes over time

Signatures are static

Message Model

```
abstract sig Person {
    sender: Person set -> Time,
    recipients: Person set -> Time,
    signatory: Person lone -> Time
}
sig Man, Woman extends Person {}
```


~~sig Signatory in Person {~~
~~signed: Signatory one -> Time~~
~~}~~

Signatures are static

Family Model: Another example

```
abstract sig Person {
    children: Person set -> Time,
    siblings: Person set -> Time,
    spouse: Person lone -> Time
}
sig Man, Woman extends Person {}
```


We want to add this relation, but where?

```
alive: Person set -> Time
```

Signatures are static

Family Model

```
abstract sig Person {
   children: Person set -> Time,
   siblings: Person set -> Time,
   spouse: Person lone -> Time
   alive: set Time
}
sig Man, Woman extends Person {}
```

Revising static constraints

```
fact static {
     -- People cannot be their own ancestors
     all t: Time | no p: Person |
    p in p.^(parents.t)

-- No one can have more than one father
-- or mother
all t: Time | all p: Person |
 lone (p.parents.t & Man)
and
 lone (p.parents.t & Woman)

...
```

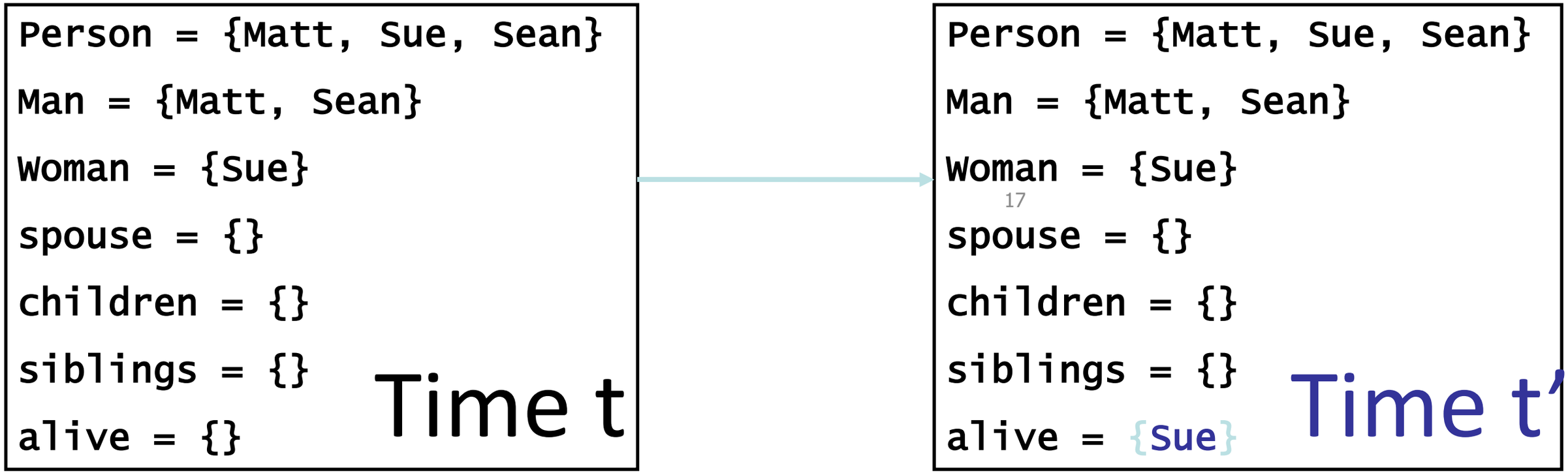
Revising static constraints

...

```
-- A person p's siblings are those people other
-- than p with the same parents as p
all t: Time | all p: Person |
  some p.parents.t implies
      p.siblings.t =
                ({q: Person | p.parents.t = q.parents.t} - p )
    else no p.siblings.t

-- Each married man (woman) has a wife (husband)
all t: Time | all p: Person |
  let s = p.spouse.t |
      (p in Man implies s in Woman) and
    (p in Woman implies s in Man)
    ...
```

Transitions

- **A person is born**
  - Add to alive relation
  - NB: No requirement that a person have parents



State Sequences

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {}
children = {}
siblings = {}
alive = {Sue}
```

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {}
siblings = {}
alive = {Sue, Matt}
```

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {}
children = {}
siblings = {}
alive = {}
```

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {(Matt,Sean), (Sue,Sean)}
siblings = {}
alive = {Sue, Matt, Sean}
```

Express a transition in Alloy

- A transition can be modeled as a predicate between two states:
  - the state right before the transition and
  - the state right after it

- We define it as predicate with (at least) two formal parameters: t, t’: Time

- Constraints over time t (resp., t’) model the state right before (resp., after) the transition

Express a transition in Alloy

- Pre condition constraints

  Describe the states to which the transition applies

- Post condition constraints
  - Describes the effects of the transition in generating the next state
- Frame condition constraints
  - Describes what does not change between pre-state and post-state of a transition

Distinguishing the pre, post and frame conditions in comment provides useful documentation

Example: Marriage

```
pred marriage [m: Man, w: Woman, t,t': Time, ] {
    -- precondition
        -- m and w must be alive before marriage
    m+w in alive.t
    -- neither one can be married
    no (m+w).spouse.t
    -- they must not be blood relatives
    not BloodRelatives [m, w, t]
    -- post-condition
    -- After marriage w is m's wife
    m.spouse.t' = w
    -- After marriage m is w's husband
    -- (redundant)
    -- frame condition ??
}
```

Frame condition

Which relations are untouched by marriage?

- 5 relations :
  - `children, parents, siblings`
  - `spouse`
  - `alive`
- `parents` and `siblings` relations are defined in terms of `children` relation
- Thus, the frame condition has only to consider `children`, `spouse` and `alive` relations

Example: Marriage

```
pred marriage [m: Man, w: Woman, t,t': Time]
{
   -- precondition
   m+w in alive.t
   no (m+w).spouse.t
   not BloodRelatives [m, w, t]
   -- post-condition
   m.spouse.t' = w
   -- frame condition
   noChildrenChangeExcept [none, t, t’]
   noSpouseChangeExcept [m+w, t, t’]
   noAliveChange [t, t’]
}
```

Instance of marriage

```
open ordering [Time] as T

…


pred marriageInstance {
    some t: Time |
  some m: Man | some w: Woman |
      let t' = T/next [t] |
        marriage [m, w, t, t’]
}
run { marriageInstance }
```

Specifying a transition system

- A transition system is a set of traces: sequences of time steps generated by the operators
- For every trace:
  - The first time step satisfies some initialization condition
  - Each pair of consecutive steps are related by an operation.

Specification of an initial step

```
pred init [t: Time] {
   no children.t
   no spouse.t
   no alive.t
}
```

Specification of a trace

```
fact Trace {

  init [T/first]

   all t: Time - T/last |

    let t' = T/next [t] |

      birth [t, t'] or

      one m: Man | one w: Woman |

       marriage [m, w, t, t'] or

       birthFromParents [m, w, t, t']

}

run {Trace and some Man and some Woman}
```

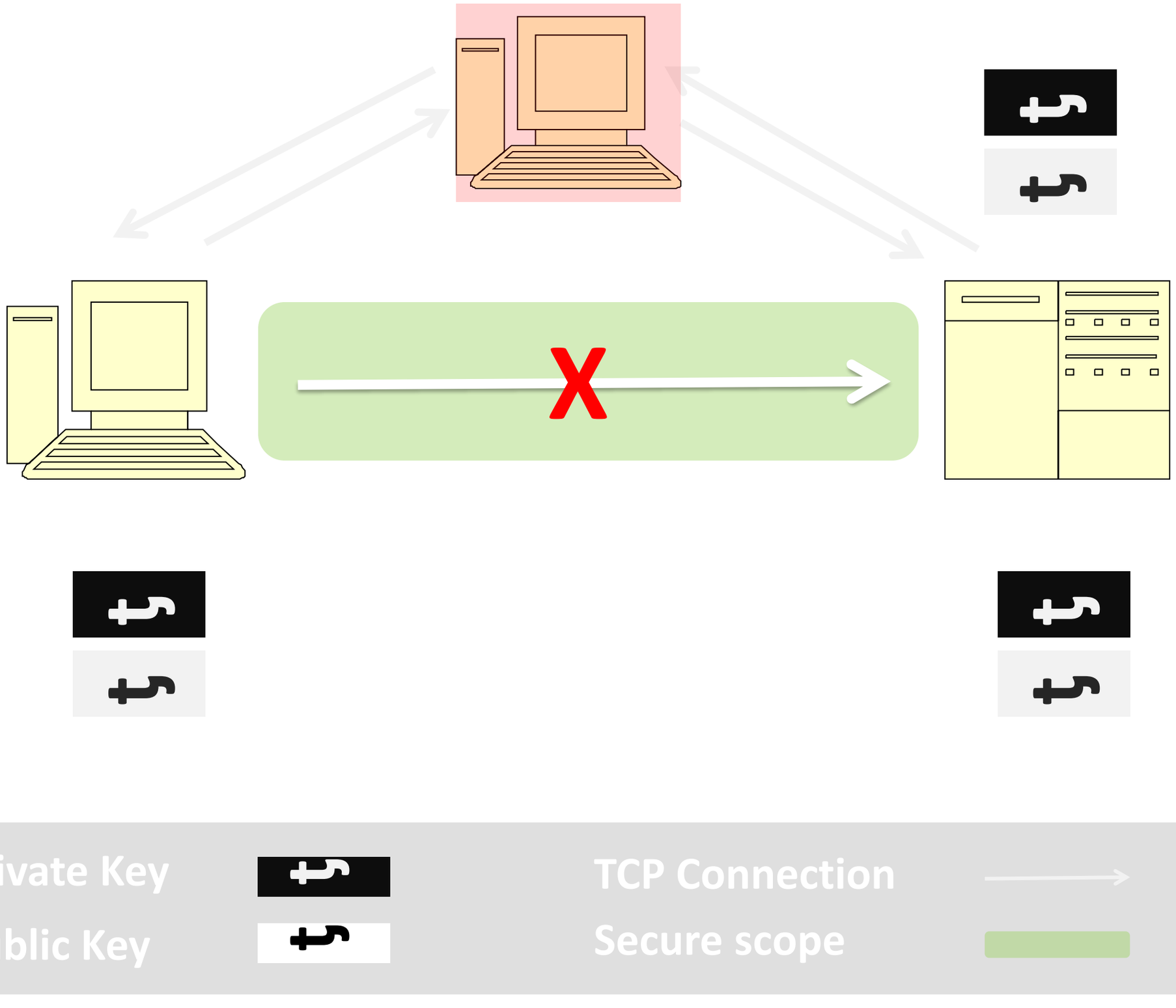
Realism constraint

```
run {
   marriageInstance
   birthInstance
   birthFromParentsInstance
} for 5
```

Man in the middle attack

- Man-In-The-Middle attack is the type of attack where attackers intrude into an existing connection to intercept the exchanged data pretending to be client to the server and server to the client.

- It involves intercepting encrypted messages , eavesdropping on a connection (passive) , and/or modifying data(active).


X
Private Key
Public Key
TCP Connection
Secure scope

Assert and Prove

```
assert RealAndVertualConnection
{
All
R_Client,V_Client,server:Actor,RC_t,VC_t,t0,t1:Time,RC_IP,VC_IP:IP,req
uest:Request |
(server.S_receive.Packet<:RC_IP= server.S_receive.Packet<:VC_IP)
And
(server.S_receive.Packet<:RC_t= server.S_receive.Packet<:VC_t)
Implies
R_Client.RC_sends.reuest->t0= server.S_receive.reuest->t0
And
V_Client.RC_sends.reuest->t0= server.S_receive.reuest->t0



}
```

Assert and Prove

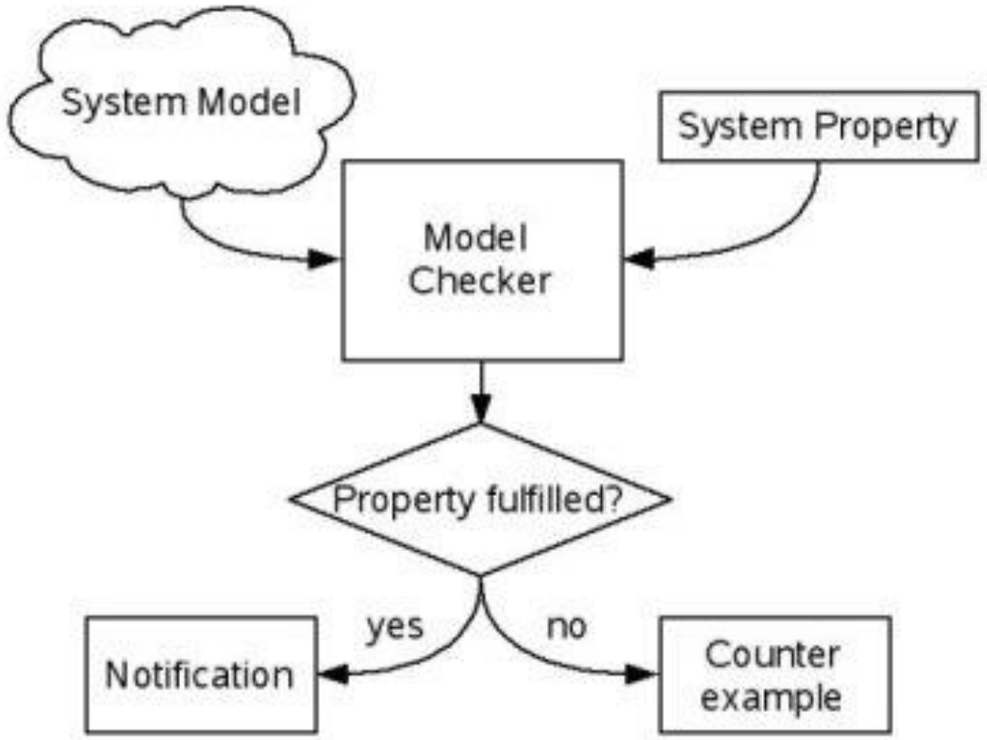
Else
(server.S_receive.Packet<:RC_IP= server.S_receive.Packet<:VC_IP)
And
(server.S_receive.Packet<:RC_t != V_Client.S_receive.Packet<:VC_t)
Implies
R_Client.RC_sends.reuest->t0 != server.S_receive.reuest->t1
And
V_Client.RC_sends.reuest->t0= server.S_receive.reuest->t0

}

```
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
2157 vars. 474 primary vars. 3796 clauses. 46ms.
No counterexample found. Assertion may be valid. 2ms.
```

Model Checking

An automated technique that,given a finite-state model of a system and a logical property, systematically checks whether this property holds for a given initial state in that model.



Edmund M. Clarke. The birth of model checking. In 25 Years of Model Checking, volume 5000 of Lecture Notes in Computer Science, pages 1{26, 2008.

Model Checking or model Analysis

- The checking process is automatic.
- Model checking is faster than traditional verification techniques (i.e. testing), and therefore it saves time and cost.
- Using logics can express many properties needed for reasoning about concurrent systems.

SPIN Model checker

- It is a model checker for the temporal logic LTL.
- Aimed at verification of protocols and software.
- Provides a graphical user interface (ispin) to the model checker and to an interactive simulator.

G.J. Holzmann. The SPIN model checker: Primer and reference manual.
Addison-Wesley Professional, 2004.

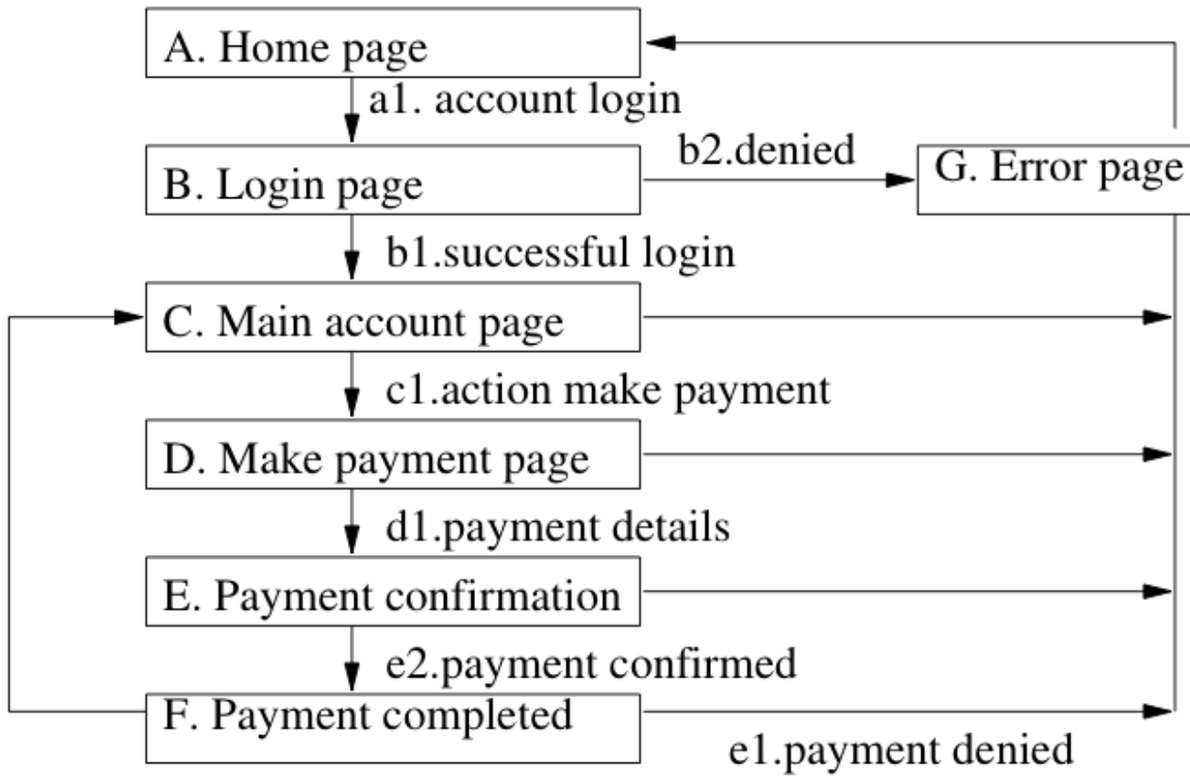
PROMELA Syntax

```
mtype = { ... };
chan = { ... };
<type> = { ... };

proctype SampleProcess1(<args>) {
  ...
}
```

- The structure of a PR
  MEta LAnguage) pro

```
proctype SampleProcess1(<args>) {
  ...
}

init {
  main body
}
```

- Investigated how to model:
  1.Authentication
- **2.Session management**
- **3 Navigation behaviour in an application.**



Modelling Web applications

- In page transitions; web pages are treated as states and; page transitions as a state transition.
- The Internal states represent the business logic, determined by input values .
- The internal state occurs synchronously with the page transition.

Modelling Authentication

- We modelled a security protocol at the start of the session.
- The user send his login credentials.

Modelling Session management

- Session is maintained throughout the communication.
- Non-deterministic (Timeouts) are given during the session.

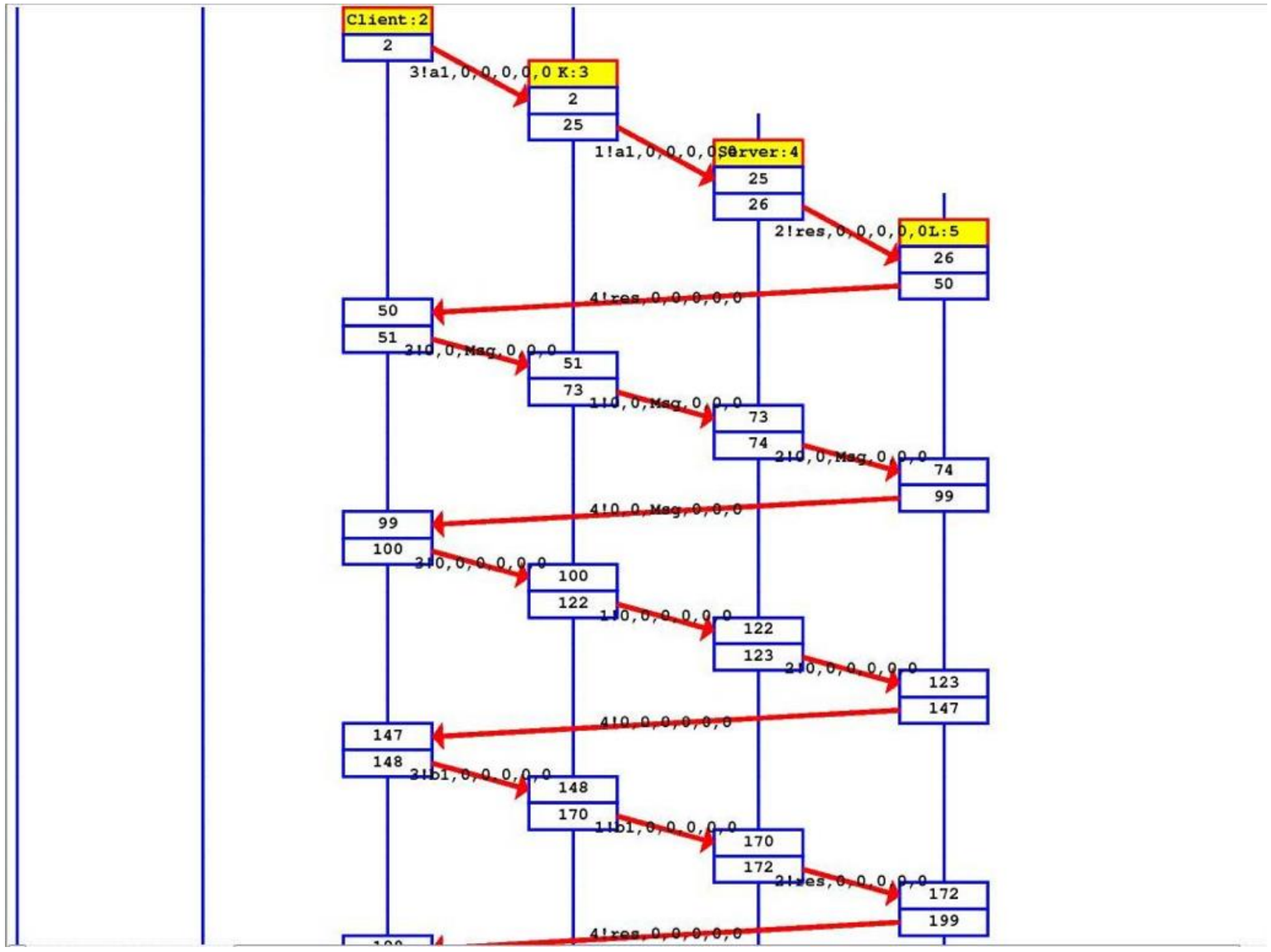
Representation In Promeal

A client is sending a request to login; the server receives it and reply.

```
chan Client ToServer = [0] of {mtype};
chan ServerToClient = [0] of {mtype};
mtype = {loginReq , ACK};

active proctype Pages()  {

HomePage:do ::       if       :: Client ToServer ! loginReq
-> ServerToCilent ? ACK->goto loginPage;       fi;
od; }

active proctype InternalState() {
do ::
      if
      :: ClientToServer ? loginReq ->
atomic {ServerToChanel!ACK;goto SloginPage};
fi;
od;
}
```

Model Simulation

- SPIN's Simulation chart
- Online-banking model.



DTSpin

- We extend our Promela model with discrete time macros.
- This will give us the ability to construct realistic web applications models.

Reference: http://www.win.tue.nl/~dragan/DTSpin/

UPPAAL model checker

- UPPAAL is used to verify real time systems.
- Models are defined graphically.

http://www.uppaal.com/

SPIN vs UPPAAL

- Design time in UPPAAL is less than building the Promela model.
- Same model Processes are defined in both tools.
- In Spin early modelling faults can be detected via the “ Message Sequence Charts”