Université
du Québec
en Outaouais

Conflict Detection in Call Control Using First-order Logic Model Checking

Ahmed Layouni
Luigi Logrippo
Ken J.Turner – University of Stirling

luigi@uqo.ca
http://w3.uqo.ca/luigi/

Changing views of FI

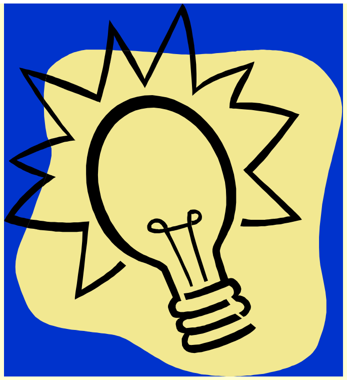
- Process-based view:
  - Early research on FI was based on the idea that Fis were the result of ***complex interleavings*** of features
    - See Feature Interaction contexts
- Logic-based view:
  - Later it became understood that many or most FIs are the result of ***logical inconsistencies*** in the specification of features we are composing

User Policies

- With the flexibility provided by IP, features will increasingly be directed by *user policies*
- In a policy directed system, features
  - Acquire logical complexity
  - While losing state complexity
- Hence the logic-based view becomes dominant

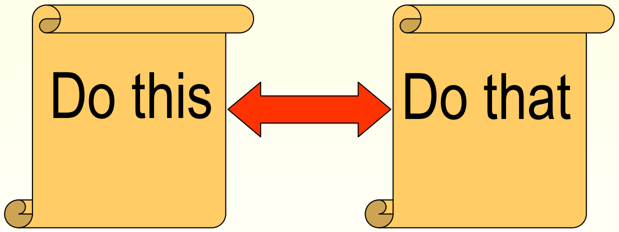
Policy systems as ECA systems

- Event (trigger, signal, stimulus)
- Condition (consultation of data base)
- Action(s) to be performed



Main idea

- *Feature interactions* are the result of logic flaws
  - Inconsistency of specs
- E.g. for the same event and condition, execute different actions



FIs as inconsistencies

- There is FI when there is inconsistency between:
  - Two simultaneous actions of one or several agents
    - They lead to inconsistent results
  - An action and a following action
    - Where the first makes the second impossible
    - Or the second contradicts the first
  - An action and the requirements of a user
  - Actions and systems requirements
- Inconsistency of actions may be visible only after complex domain-dependent considerations
- It is usually a fact provided as human input to FI detection tools

This idea is present in a number of works

- Within an explicit logic framework:
  - Felty and Namjoshi, FIW 2000
  - Various papers of Aiguier and Le Gall, e.g. Formal Methods 2006 (LNCS 4085)
- More generally talking about ‘conflicts’, ‘broken assumptions’, etc.
  - Kolberg, Magill, Wilson, IEEE Comm., 2003
  - Gorse, Logrippo, Sincennes, originally in Gorse’s Master’s thesis of 2000 and eventually published in SoSym 2006
  - Metzger et al., FIW 2003 and 2005
  - Turner, Blair 2006
  - Etc.

In fact, from the beginning

- Seminal paper by Cameron et al. identifies as main causes of FI:
  - Violation of assumptions
    - a clear case of inconsistency
  - Limitation of network support
    - inconsistency between concurrent claims of resources

FI symptoms according to Gorse, Logrippo, Sincennes

- Basic cases of FI:
  - Features leading to different results
    - Non-contradicting ones (non-determinism)
    - Contradicting ones
  - A feature enables another, with contradicting results
  - A feature enables another, which directly or indirectly enables the first (infinite loop)

Connections…

- Considerable work exists on
  - Consistency in software requirements
  - Consistency of viewpoints in requirements
- These connections have not yet been fully exploited within FI research

How do we know about the conflicts

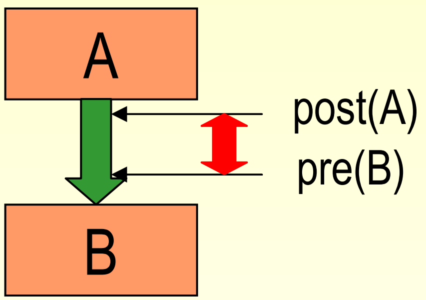
- This can be obvious, in cases where there is a straight contradiction
  - A and not A
    - But this is rarely the case
- In many cases, contradiction is a result of domain-dependent considerations
  - E.g. *accept call* contradicts *disconnect*

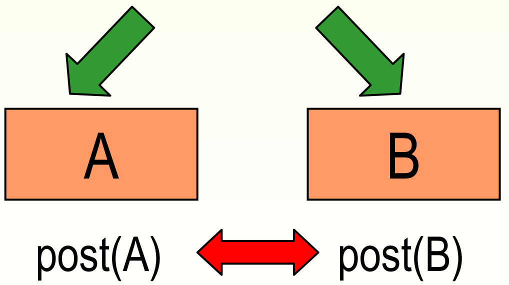
Next step of analysis: Considering pre- & post-conditions

- Wu and Schulzrinne (ICFI-Leicester) have moved forward by
  - Introducing the idea of **conflicts** between **pre-** and **post-conditions** of actions
  - Determining action conflicts on the basis of their pre- and post-conditions
  - This can provide information also on possible FI *resolution*

Interactions of pre- and post-conditions

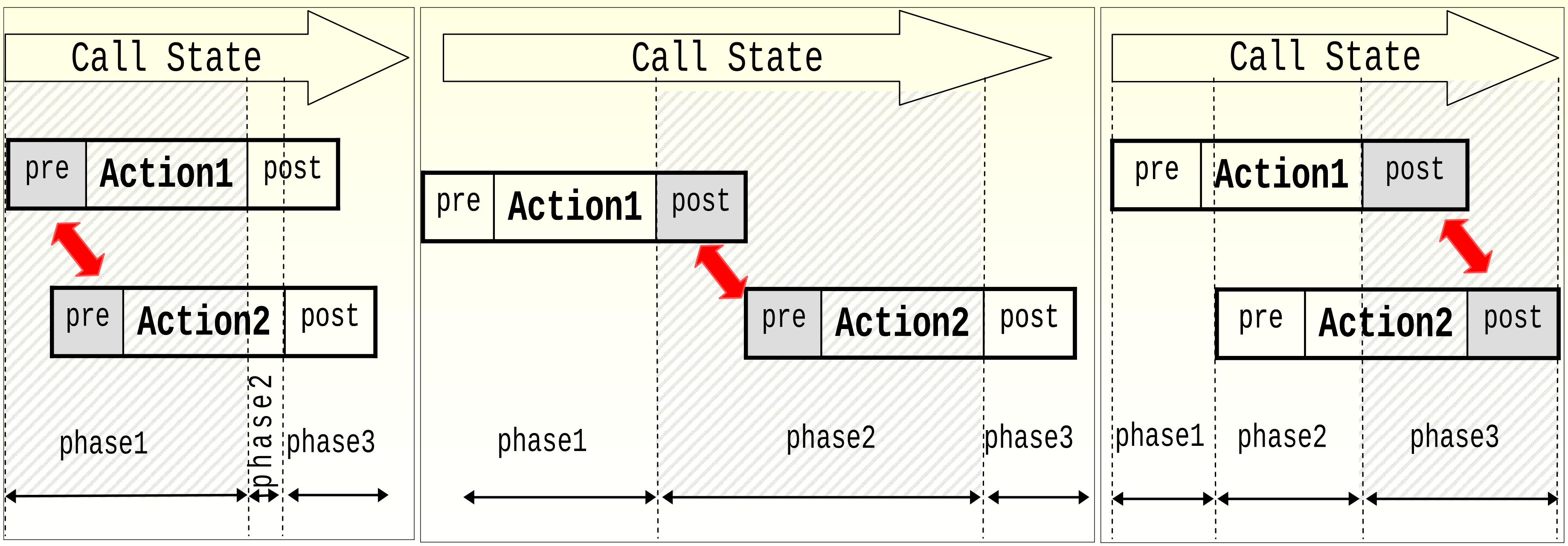
- Enable(A,B)   (positive interaction)
  - the post-condition of A implies the pre-condition of B
- Disable(A,B) (negative interaction)
  - The post-condition of A does not imply the pre-condition of B
- Conflict of post-conditions: (negative interactions)
  - The expected postconditions of two actions conflict directly
    - Special case: they request the same resources
  - The expected postconditions of two actions conflict because of parameters





Three types of conflicts

Concurrency conflict **Disabling conflict** **Results conflict**



How to choose pre- and post-condition:

APPEL case study

- Software systems are complex and every action is the result of, also produces, complex conditions
- Only few elements can be expressed in logic statements that are meant for analysis
- These elements must be chosen in terms of broad generalizations
- The choice of these elements is vital for producing a useful analysis
- In terms of the characteristics of APPEL, we have chosen to focus on two elements:
  - Call states
  - Media state

How to determine conflicts

- Similarly, conflicts must be determined in terms of broad generalizations
  - E.g. if one action requests a resource of a certain type, then it might disable another action that requires the same type of resources

APPEL Actions

- connect_to *initiates a new and independent call*
- reject_call *rejects a call*
- forward_to *changes the destination of the call*
- fork_to *adds an alternative leg to the call*
- add_party *adds a new party to an existing call*
- remove_party *removes a party from the call*
- add_medium *adds a new medium to the call*
- remove_medium *removes a medium from the call*
- remove_default *removes the def. medium from the call*
- disconnect *disconnects the call*

APPEL Example 1

- reject_call **concurrent with** add_party
  - Precondition for reject_call:
    - CallSetup state
  - Precondition for add_party:
    - MidCall state
  - State **conflict** for these two actions

If a feature or a combination of feature requires simultaneous execution of these actions, this won't be possible because of state conflict

APPEL Example 2

- remove_party **concurrent with** fork_to
  - Resulting media state by remove_party:
    - DefaultAvailable
  - Resulting media state by fork_to:
    - DefaultReserved
  - Resource **conflict** for these two actions

APPEL Example 3

- add_party followed by forward_to
  - Resulting call state by AddCaller vs precondition call state by ForwardTo:
    - MidCall vs CallSetup **Conflict**
  - Resulting media state by AddParty vs precondition media state by ForwardTo:
    - DefaultReserved vs DefaultReserved **OK**

Now for a systematic analysis

Pre-and post-conditions of call actions

Action	Pre-conditions		Post-conditions	
	Connection State	Media State	Connection State	Media State
connect_to	NoCall	DefaultAvail	CallSetup	DefaultReserv
reject_call	CallSetup	DefaultReserv	NoCall	DefaultAvail
forward_to	CallSetup	DefaultReserv	CallForwarded	DefaultAvail
fork_to	CallSetup	DefaultReserv	CallForked	DefaultReserv
add_party	MidCall	DefaultAvail	PartyAddedToCall, MidCall	DefaultReserv
remove_party	MidCall, PartyAddedToCall	DefaultReserv	MidCall	DefaultAvail
add_medium	MidCall	MediumAvail	MidCall	MediumReserv
remove_medium	MidCall	MediumReserv	MidCall	MediumAvail
remove_default	MidCall	DefaultReserv	MidCall	DefaultAvail
disconnect	MidCall	DefaultReserv	NoCall	DefaultAvail

Connection state incompatibilities:
the system cannot be in two different states

Connection State 1	Connection State 2
NoCall	MidCall
NoCall	CallSetup
CallSetup	MidCall
CallSetup	NoCall
MidCall	NoCall
MidCall	CallSetup

Media state incompatibilities
concurrency conflict

Precondition media state Action1	Precondition media state Action2
DefaultAvailable	DefaultReserved
DefaultReserved	DefaultAvailable
MediumAvailable	MediumReserved
MediumReserved	MediumAvailable

Example of conflict:

two actions that cannot be executed in parallel

Action	Pre-conditions		Post-conditions	
	Connection State	Media State	Connection State	Media State
connect_to	NoCall	DefaultAvail	CallSetup	DefaultReserv
reject_call	CallSetup	DefaultReserv	NoCall	DefaultAvail
forward_to	CallSetup	DefaultReserv	CallForwarded	DefaultAvail
fork_to	CallSetup	DefaultReserv	CallForked	DefaultReserv
add_party	MidCall	DefaultAvail	PartyAddedToCall, MidCall	DefaultReserv
remove_party	MidCall, PartyAddedToCall	DefaultReserv	MidCall	DefaultAvail
add_medium	MidCall	MediumAvail	MidCall	MediumReserv
remove_medium	MidCall	MediumReserv	MidCall	MediumAvail
remove_default	MidCall	DefaultReserv	MidCall	DefaultAvail
disconnect	MidCall	DefaultReserv	NoCall	DefaultAvail

Two actions that cannot follow each other

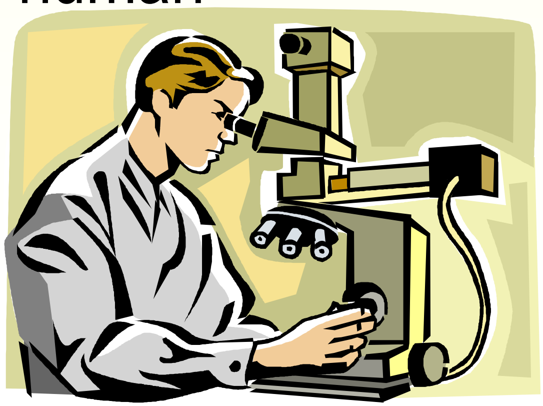
Action	Pre-conditions		Post-conditions	
	Connection State	Media State	Connection State	Media State
connect_to	NoCall	DefaultAvail	CallSetup	DefaultReserv
reject_call	CallSetup	DefaultReserv	NoCall	DefaultAvail
forward_to	CallSetup	DefaultReserv	CallForwarded	DefaultAvail
fork_to	CallSetup	DefaultReserv	CallForked	DefaultReserv
add_party	MidCall	DefaultAvail	PartyAddedToCall, MidCall	DefaultReserv
remove_party	MidCall, PartyAddedToCall	DefaultReserv	MidCall	DefaultAvail
add_medium	MidCall	MediumAvail	MidCall	MediumReserv
remove_medium	MidCall	MediumReserv	MidCall	MediumAvail
remove_default	MidCall	DefaultReserv	MidCall	DefaultAvail
disconnect	MidCall	DefaultReserv	NoCall	DefaultAvail

Extent of analysis

- 10 actions x 10 actions x 6 predicates:
  - 600 cases were considered
    - Analysis is **complete** within the framework of our abstractions
- Quite a number of potential interactions was discovered between the 10 actions
  - See Fig. 6 in paper

Symptoms of conflicts

- Due to the inability to formalize all elements of a domain, action inconsistency is usually a *symptom*
  - Based on knowledge of expected systems behavior
  - Detection is tentative
  - Detection tool identifies possible conflict scenarios and interaction must be confirmed by human inspection



Granularity

- This analysis has coarse granularity
  - Relatively to what one could envisage…
    - But still better than other techniques
- Improvements possible:
  - More detailed analysis of pre- and post-conditions
  - Parameters, addresses

FI Resolution

- This approach provides little immediate help for FI resolution
- However, it might eventually, because the more information is available regarding the reasons for interaction, the more we can address it appropriately
- Research topic…

How to detect

- Specifications must be precise!
  - Sometimes they are already sufficiently precise, e.g. in a XML-based language
- Constraint Logic Programming
  - Given a set of logic constraints, CPL tools can tell whether
    - There is a solution, constraints are satisfiable
    - There is no solution, in fact there is a counterexample
- First order model checking
  - A related technique

Alloy

- Formal language and related tool developed at MIT
  - Daniel Jackson
- Tool is a *first-order logic model checker* with FINITE MODELS
  - Note difference wrt temporal logic model checkers
- Alloy's front end:
  - A logic-based language
- Alloy's engine:
  - an efficient Constraint Satisfaction (SAT) algorithm
- Alloy includes many interesting concepts, and it would not be possible to present it well in few minutes

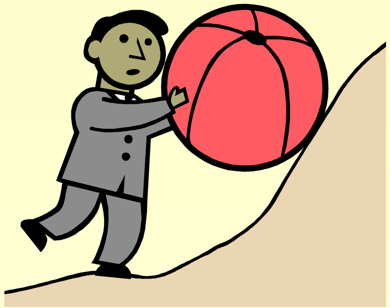
Alloy specifications

- Alloy allows to specify a set of constraints in any of, or a combination of
  - Logical style (1^{st} order pred calculus)
  - Relational style
  - 'Navigational' style
- Very expressive user language

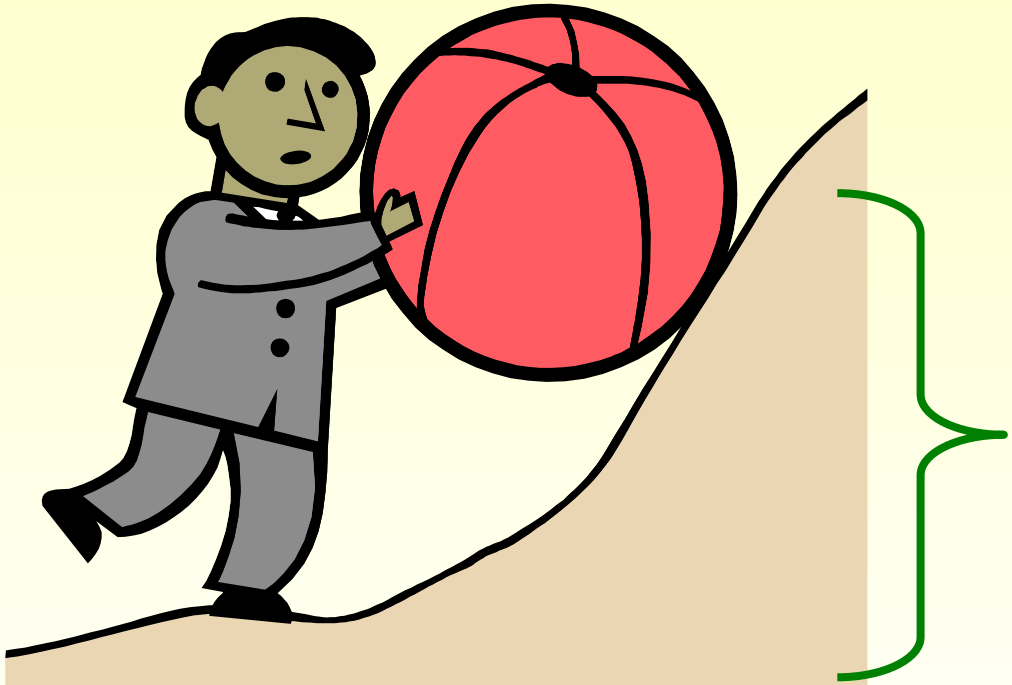
Some elements of Alloy

- **Facts, Predicates, Functions**: describe the system, in terms of constraints
- **Assertions**: state properties that are believed to be true of the system
- **Check**: checks a given assertion, trying to find a **counterexample**
- **Run**: runs a given predicate, trying to find an **example**
  - Run and check have to specify how many instances should be created for each type: FINITE MODELS

How Alloy works



- Alloy expresses the constraints in terms of boolean expressions and then tries to solve these by invoking off-the-shelf SAT solvers
- This problem is *NP-complete*, however improvements in efficiency of SAT solvers allows many non-trivial problems to be treated
- Current solvers can handle
  - thousands of boolean vars,
  - hundreds of expressions
    - But much depends on the type of the expressions



Feasible part of the curve

First order logic – overkill?

- Yes, for our specific problem
  - We do simple comparisons
- However, in general, pre- post-conditions can be arbitrarily complex logic statements
  - Approach will need first order logic in order to be generalized

Conclusions

- Complex designs require the composition of complex features
  - With user control of what will happen in different situation (user *policies*)
- Introduction of these features requires sophisticated methods to detect different situations of feature conflicts
- Model checkers and constraint logic programming provide tools to detect potential conflicts

Merci! – Questions?