Formal Methods for Interactive Systems

Part 8 — Cognitive Architectures

Antonio Cerone
United Nations University
International Institute for Software Technology
Macau SAR China
email: `antonio@iist.unu.edu`
web: `www.iist.unu.edu`

Cognitive Models

of the user

Cognitive Models

of the user

- competence models represent expected behaviour

Cognitive Models

of the user

- competence models represent expected behaviour
- performance models represent and analyse routine behaviour

Cognitive Models

of the user

- competence models represent expected behaviour
- performance models represent and analyse routine behaviour

deal with three different levels:

- goal and task hierarchies: GOMS, Cognitive Complexity Theory (CCT)

Cognitive Models

of the user

- competence models represent expected behaviour
- performance models represent and analyse routine behaviour

deal with three different levels:

- goal and task hierarchies: GOMS, Cognitive Complexity Theory (CCT)
- human understanding: BNF, Task-action Grammar (TAG)

Cognitive Models

of the user

- competence models represent expected behaviour
- performance models represent and analyse routine behaviour

deal with three different levels:

- goal and task hierarchies: GOMS, Cognitive Complexity Theory (CCT)
- human understanding: BNF, Task-action Grammar (TAG)
- physical/device: Keystroke-level Model (KLM)

Architectural Aspects

Cognitive models incorporate implicit and explicit models of cognitive processing

Architectural Aspects

Cognitive models incorporate implicit and explicit models of cognitive processing

- GOMS: divide and conquer using subgoals
- CCT: production rules in LTM matched agains STM contents
- KLM: motor and mental operators based on Model Human processor

Architectural Aspects

Cognitive models incorporate implicit and explicit models of cognitive processing

- GOMS: divide and conquer using subgoals
- CCT: production rules in LTM matched agains STM contents
- KLM: motor and mental operators based on Model Human processor

are Architectural Aspects
$\Longrightarrow$ aim to some performance analysis

Architectural Aspects

Cognitive models incorporate implicit and explicit models of cognitive processing

- GOMS: divide and conquer using subgoals
- CCT: production rules in LTM matched agains STM contents
- KLM: motor and mental operators based on Model Human processor

are Architectural Aspects

$\Longrightarrow$ aim to some performance analysis

but seldom deals with user observation and perception $\Longrightarrow$ mainly competence models

Error Detection

Weakness of cognitive models:

errors must be explicitly defined in the model

$\Longrightarrow$ no error detection

Error Detection

Weakness of cognitive models:
errors must be explicitly defined in the model
$\Longrightarrow$ no error detection

ATC Example: failure decomposition was given rather than detected

Error Detection

Weakness of cognitive models:
errors must be explicitly defined in the model
$\Longrightarrow$ no error detection

ATC Example: failure decomposition was given rather than detected

Users behave rationally
$\Longrightarrow$ make persistent errors

Rational Behaviour

based on

Rational Behaviour behaviour that is intended to achieve a specific goal

in AI: Knowledge-level System $=$ Agent behaves in Environment

Rational Behaviour

based on

Rational Behaviour behaviour that is intended to achieve a specific goal

in AI: Knowledge-level System = Agent behaves in Environment

in contrast with

Computational Behaviour behaviour defined by an algorithm without an explicit goal

Computational Architectures

- algorithm describes the problem solution

Computational Architectures

- algorithm describes the problem solution
- $\Longrightarrow$ represented in a programming language

Computational Architectures

- algorithm describes the problem solution
- $\Longrightarrow$ represented in a programming language
- compilation $\Longrightarrow$ problem space $+$ actions to traverse represented in the machine architecture

Computational Architectures

- algorithm describes the problem solution
- $\Longrightarrow$ represented in a programming language
- compilation $\Longrightarrow$ problem space $+$ actions to traverse represented in the machine architecture
- execution terminates once a desired state reached

Computational Architectures

- algorithm describes the problem solution
- $\Longrightarrow$ represented in a programming language
- compilation $\Longrightarrow$ problem space $+$ actions to traverse represented in the machine architecture
- execution terminates once a desired state reached

Machine does not formulate the problem space (the goal is not programmed)

Cognitive Architectures

- goal to achieve
- $\Longrightarrow$ represented as a set of goal states
- rational behaviour $\Longrightarrow$ to select appropriate operators to generate new states starting form the initial state
- goal achieved once a goal state is reached

based on problem space theory, developed by Newell and Simon [Newell et a. 91]

Cognitive Activities

- goal formulation creates the initial state and use perception sense changes in the external environment which are relevant to the goal of the agent

Cognitive Activities

- goal formulation creates the initial state and use perception sense changes in the external environment which are relevant to the goal of the agent
- operation selection to transform a given state to one closer to a goal state $\Longrightarrow$ rational behaviour

Cognitive Activities

- goal formulation creates the initial state and use perception sense changes in the external environment which are relevant to the goal of the agent
- operation selection to transform a given state to one closer to a goal state $\Longrightarrow$ rational behaviour
- operation application changes the states of the agent and the environment

Cognitive Activities

- goal formulation creates the initial state and use perception sense changes in the external environment which are relevant to the goal of the agent
- operation selection to transform a given state to one closer to a goal state $\Longrightarrow$ rational behaviour
- operation application changes the states of the agent and the environment
- goal completion when the new state is a goal state the agent becomes inactive

Knowledge Role

- goal formulation
- operation selection
- operation application
- goal completion

Knowledge Role

- goal formulation
- operation selection
- operation application
- goal completion

Steps are triggered by knowledge availability

Knowledge Role

- goal formulation
- operation selection
- operation application
- goal completion

Steps are triggered by knowledge availability

knowledge availability $\Longrightarrow$ recursion:
new space problem invoked with
goal $=$ find needed knowledge

SOAR

Executable Cognitive Architecture developped by Allen Newell, John Laird and Paul Rosembloom in 1983 [Newell et a. 87]

Used by the University of Michigan

http://sitemaker.umich.edu/soar/

PUM

Programmable User Models

Psychologically Constrained Architecture which an interface designer is invited to program to simulate a user performing a range of tasks with a proposed interface

[Young et a. 89]

PUM Philosophy

The interface designer

- must program the PUM respecting the constraints
  $\Longrightarrow$ driven interface design
  $\Longrightarrow$ explicit “user program”

PUM Philosophy

The interface designer

- must program the PUM respecting the constraints
  $\Longrightarrow$ driven interface design
  $\Longrightarrow$ explicit “user program”
- run the model
  ($\Longrightarrow$ “user program” is executable)
  to make predictions
  $\Longrightarrow$ show source of predictions and strategy options

PUM Purposes

- Outcome:
  predictive evaluation to tell the designer usability of a proposed design before it is actually built

PUM Purposes

- Outcome:
  predictive evaluation to tell the designer usability of a proposed design before it is actually built
- Benefits for the designer:
  - to draw the designer attention to issues of usability
  - to provide a way of reasoning about usability

PUMA: PUM Applications

Research Project aim to

- Bring the PUM metodology into industrial design
- Using formal methods to effectively implement PUM

PUMA Research Group Website:

http://www.cs.mdx.ac.uk/puma/

Detecting User Errors

Work by Curzon and Blandford [Curzon et al. 01]:

- PUM defined in the HOL Theorem Prover $\Longrightarrow$ Generic User Model described by sets of non-deterministic rules

Detecting User Errors

Work by Curzon and Blandford [Curzon et al. 01]:

- PUM defined in the HOL Theorem Prover
  $\Longrightarrow$ Generic User Model described by sets of non-deterministic rules
- Programming performed using SML
  $\Longrightarrow$ target the Generic User Model to a particular design

Detecting User Errors

Work by Curzon and Blandford [Curzon et al. 01]:

- PUM defined in the HOL Theorem Prover
  $\Longrightarrow$ Generic User Model described by sets of non-deterministic rules
- Programming performed using SML
  $\Longrightarrow$ target the Generic User Model to a particular design
- Automated Correctness Proof

Detecting User Errors

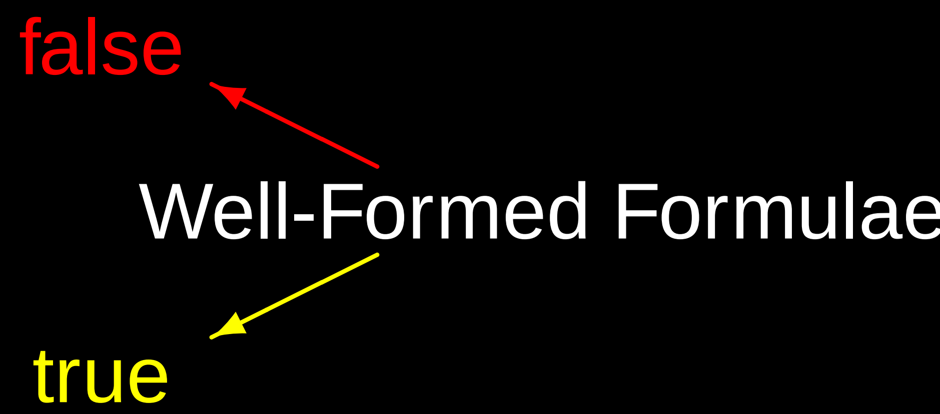
Work by Curzon and Blandford [Curzon et al. 01]:

- PUM defined in the HOL Theorem Prover
  $\Longrightarrow$ Generic User Model described by sets of non-deterministic rules
- Programming performed using SML
  $\Longrightarrow$ target the Generic User Model to a particular design
- Automated Correctness Proof
- Informal reasoning to detect errors

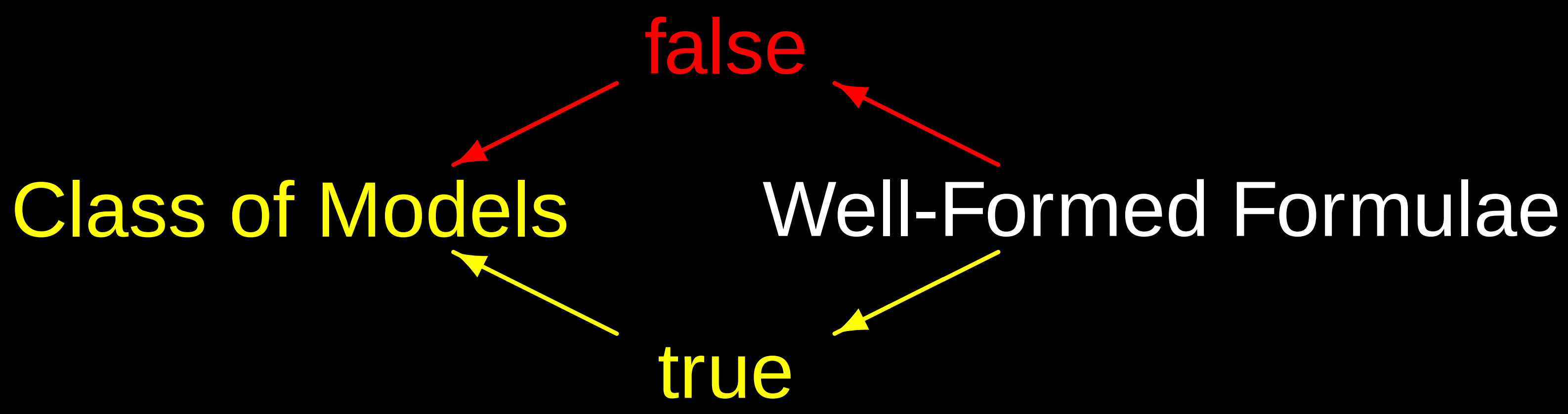
Theorem-proving

Well-Formed Formulae

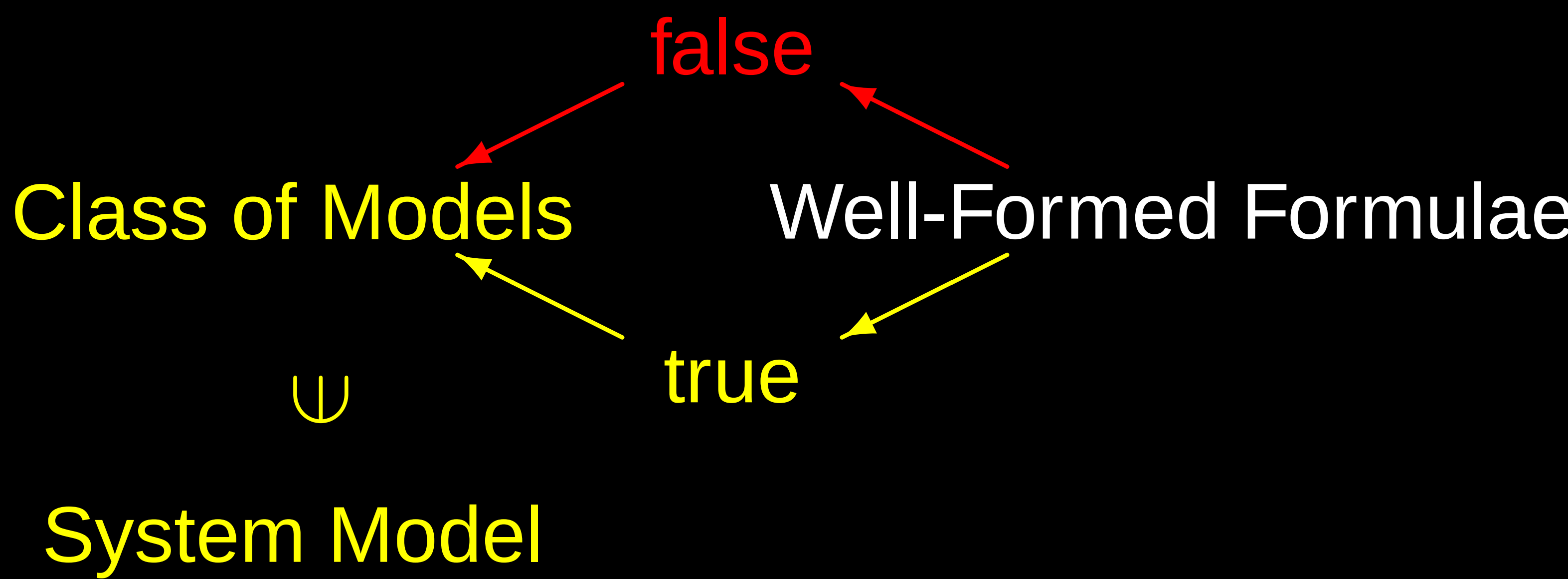
Theorem-proving



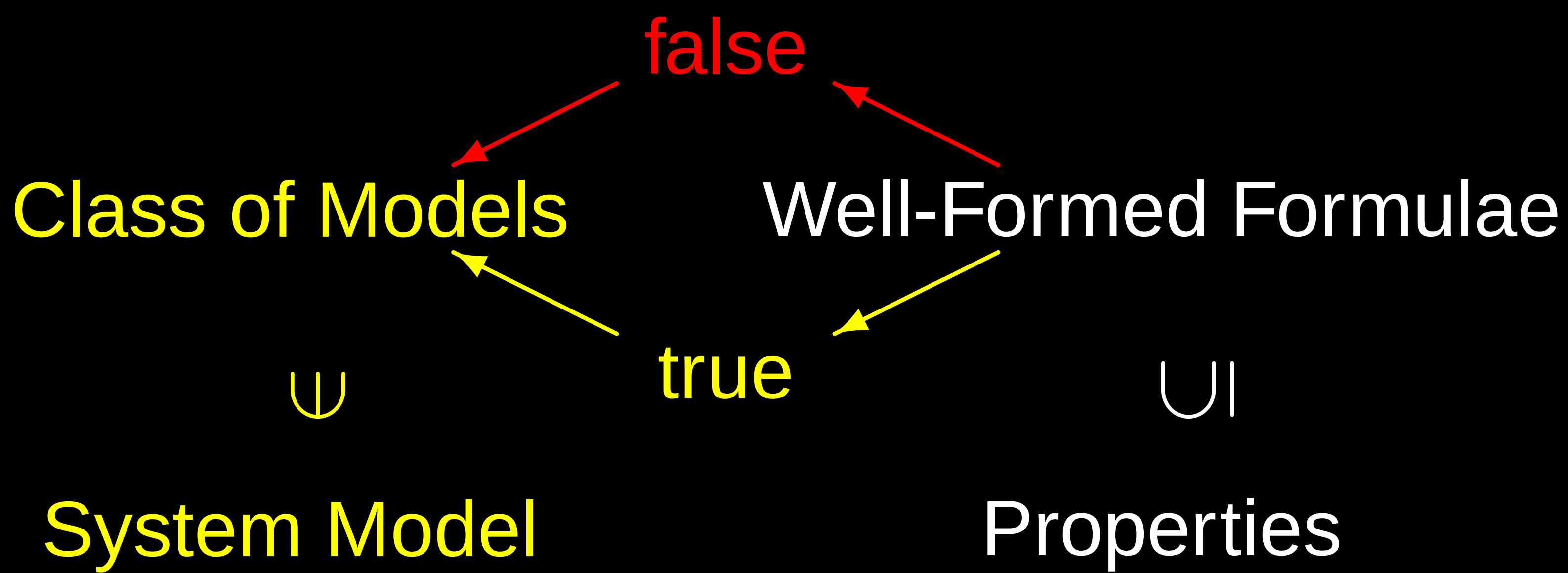
Theorem-proving



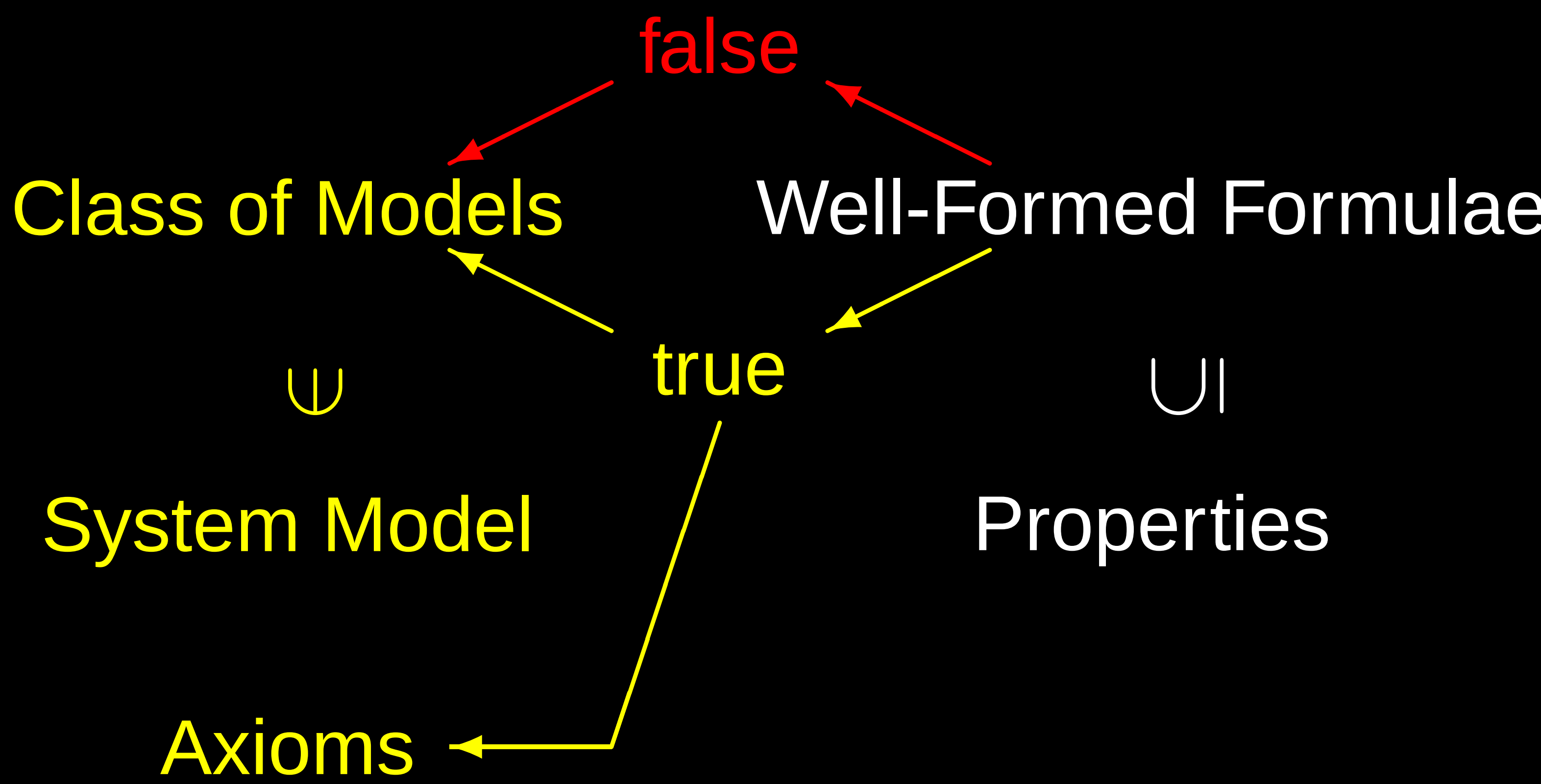
Theorem-proving



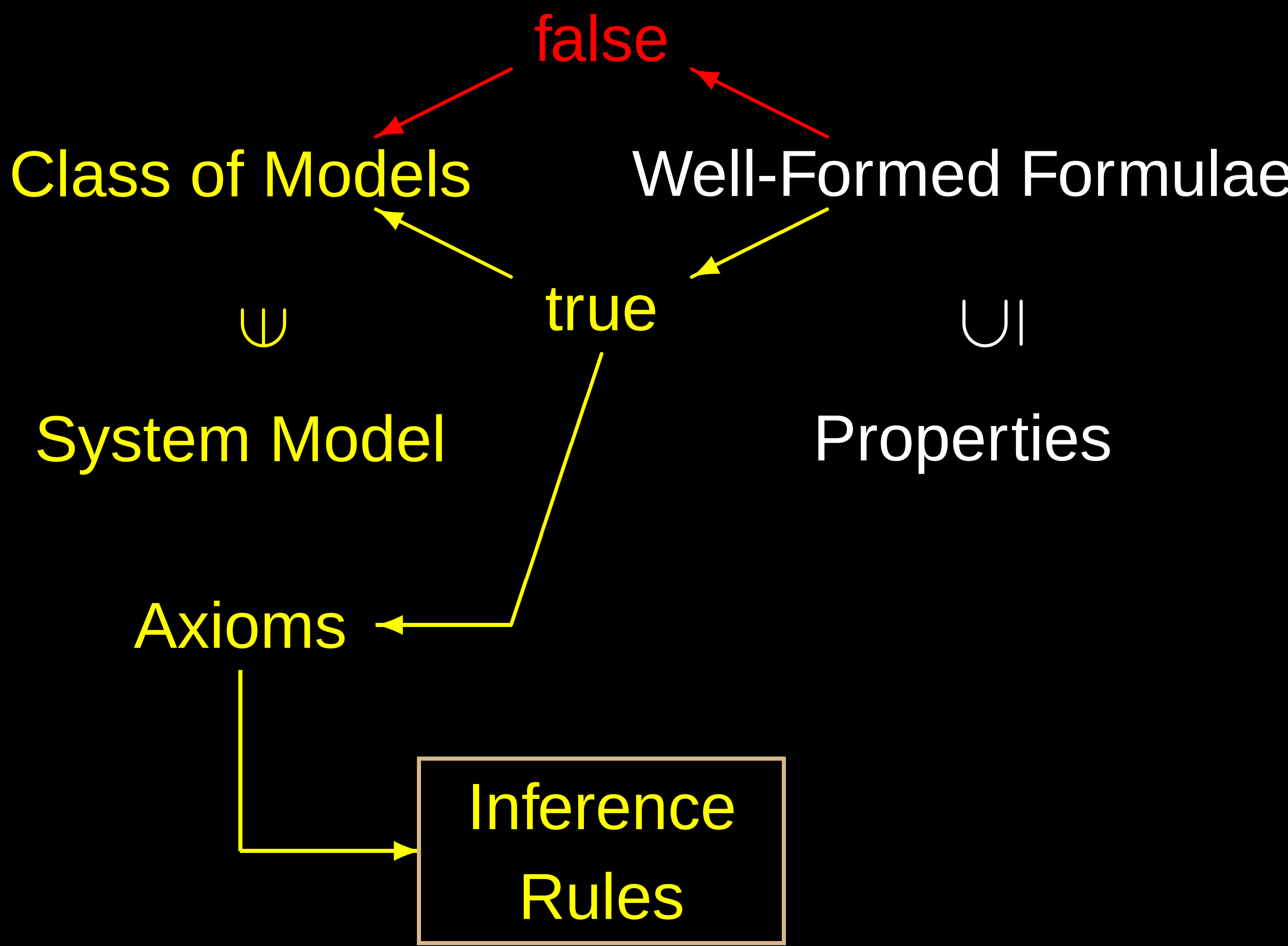
Theorem-proving



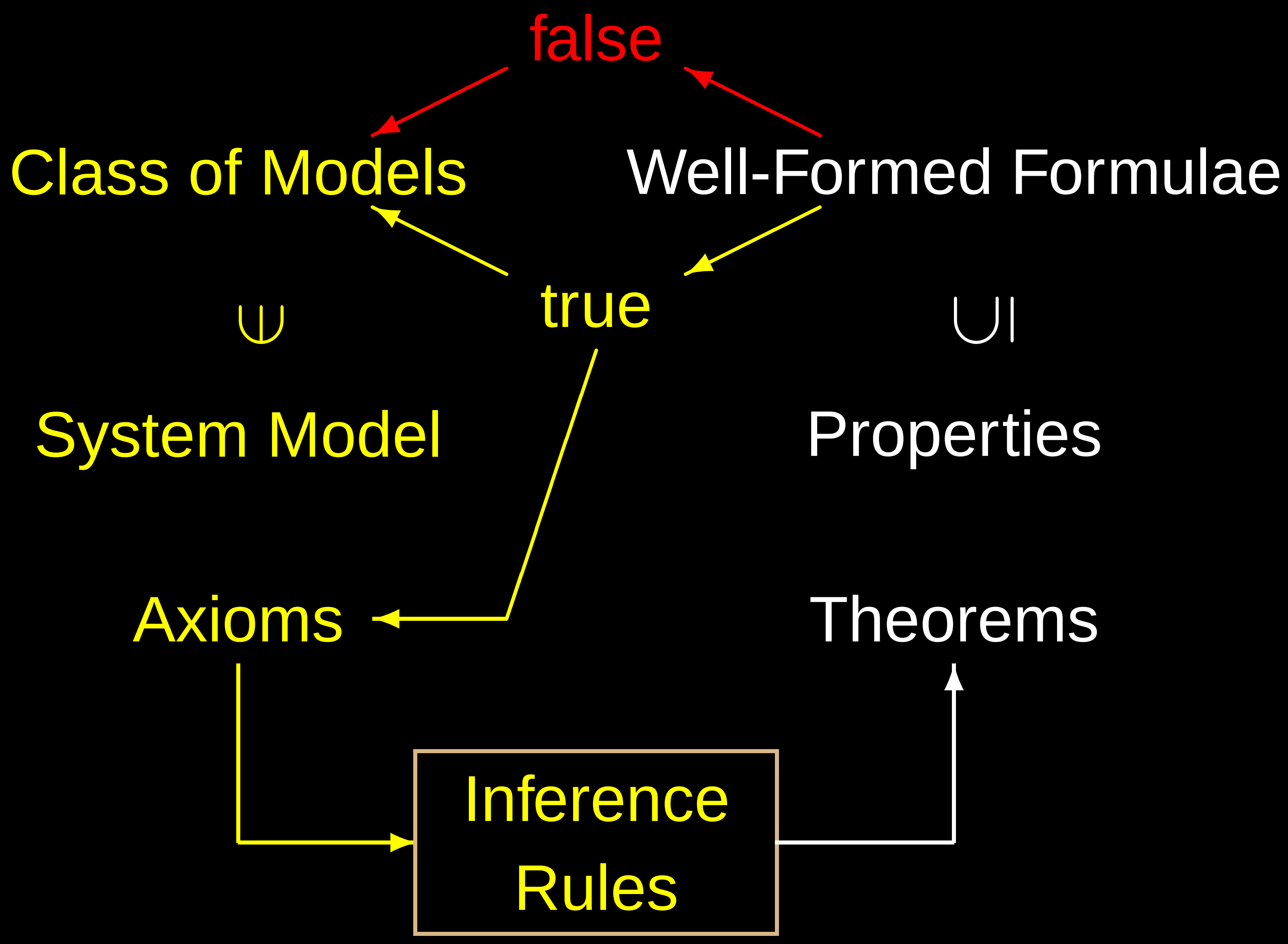
Theorem-proving



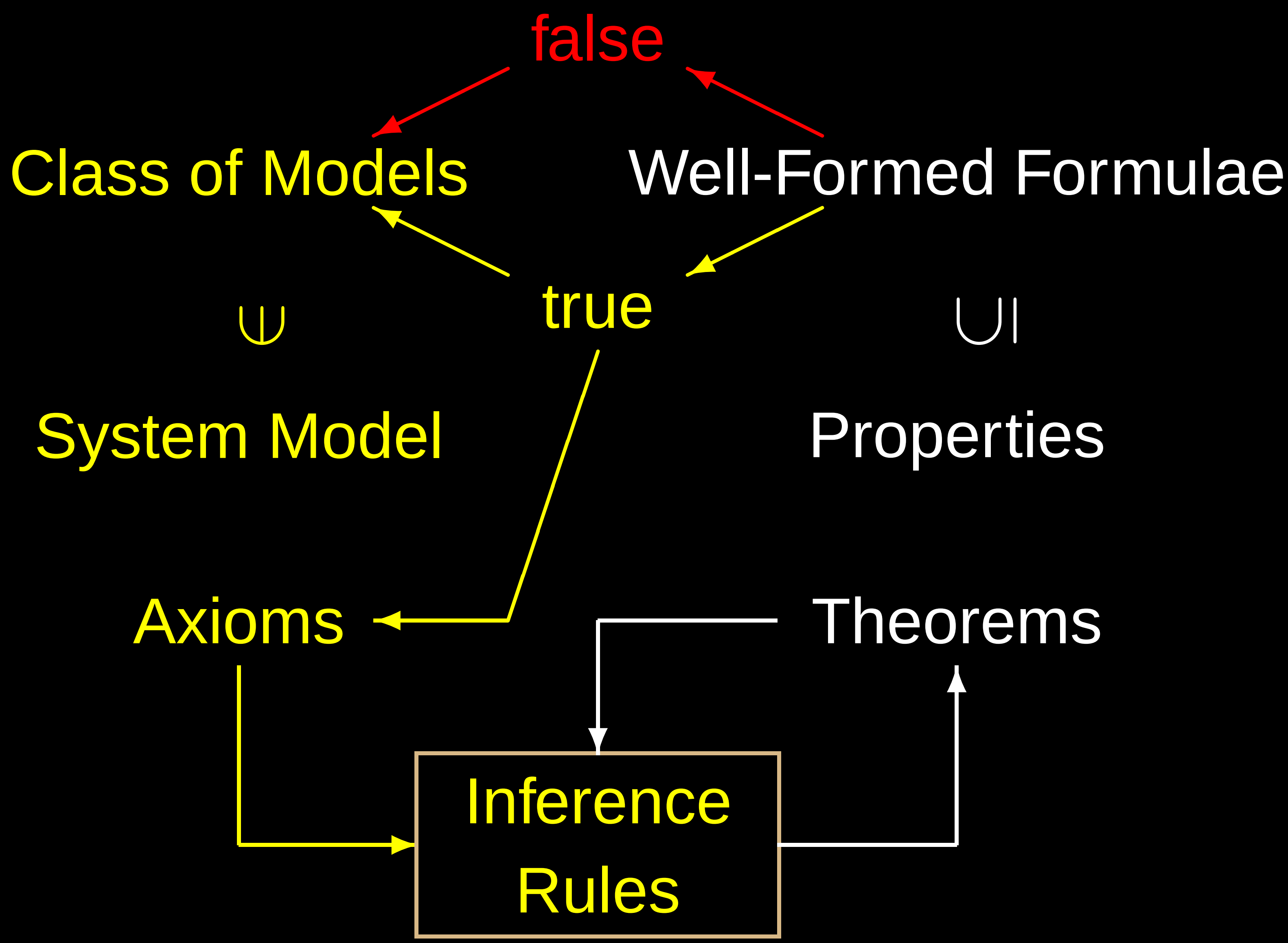
Theorem-proving



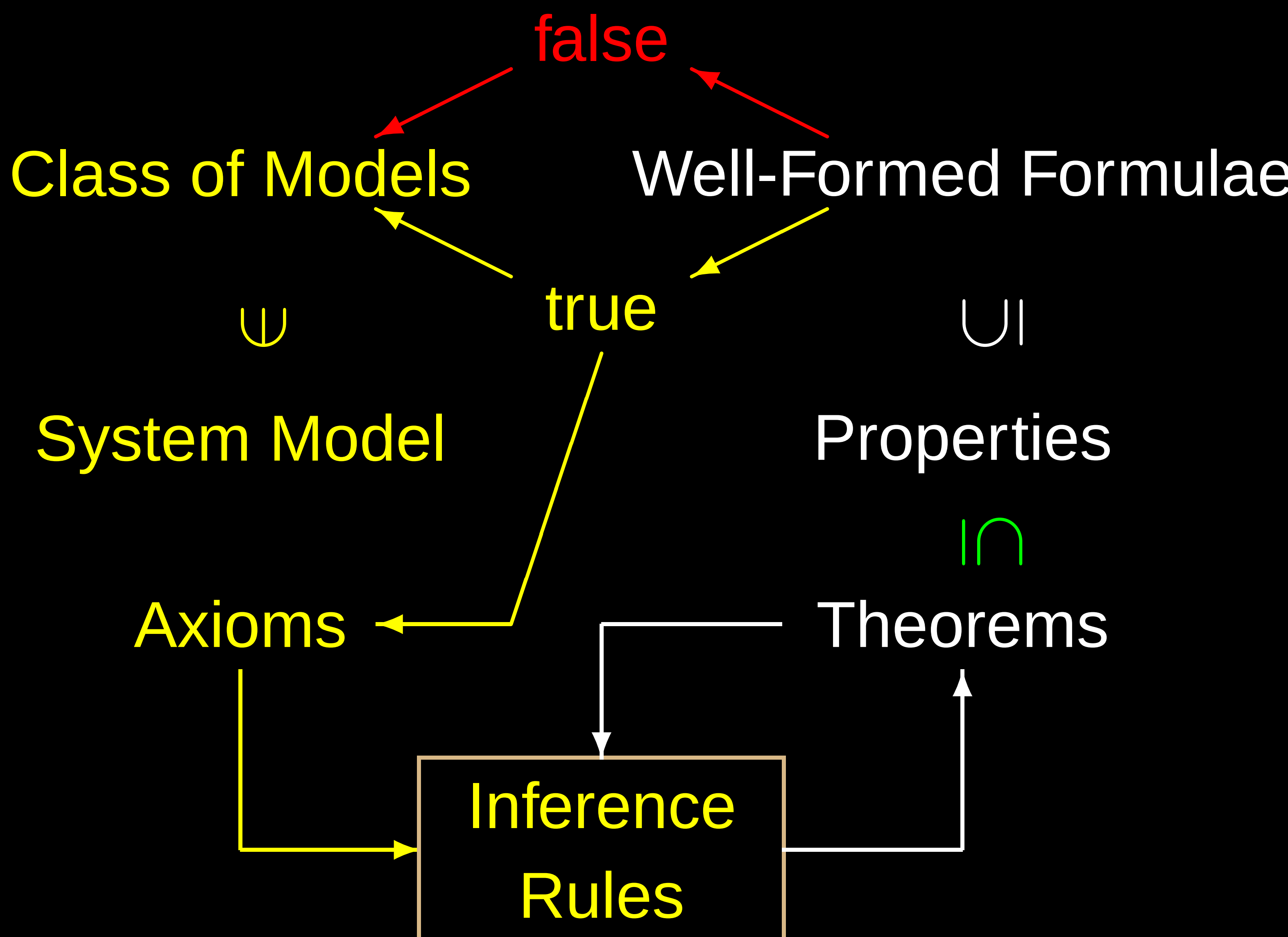
Theorem-proving



Theorem-proving



Theorem-proving



Theorem-proving: Pros & Cons

- Advantages

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems
    - Complex data structure

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems
    - Complex data structure
- Disadvantages

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems
    - Complex data structure
- Disadvantages
  - Semidecidability

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems
    - Complex data structure
- Disadvantages
  - Semidecidability
  - Verification procedure not fully automated

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems
    - Complex data structure
- Disadvantages
  - Semidecidability
  - Verification procedure not fully automated
  - Tools difficult to use

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems
    - Complex data structure
- Disadvantages
  - Semidecidability
  - Verification procedure not fully automated
  - Tools difficult to use
  - No scalability

Theorem-proving: Pros & Cons

- Advantages
  - Maximum Modelling Expressivity
    - Infinite State Systems
    - Complex data structure
- Disadvantages
  - Semidecidability
  - Verification procedure not fully automated
  - Tools difficult to use
  - No scalability
  - Does not allow debugging

Model-Checking: Pros & Cons

- Advantages

Model-Checking: Pros & Cons

- Advantages
  - Decidability

Model-Checking: Pros & Cons

- Advantages
  - Decidability
  - May be fully automated

Model-Checking: Pros & Cons

- Advantages
    - Decidability
    - May be fully automated
    - Better usability tools

Model-Checking: Pros & Cons

- Advantages
  - Decidability
  - May be fully automated
  - Better usability tools
  - Good scalability

Model-Checking: Pros & Cons

- Advantages
  - Decidability
  - May be fully automated
  - Better usability tools
  - Good scalability
  - Allows debugging

Model-Checking: Pros & Cons

- Advantages
  - Decidability
  - May be fully automated
  - Better usability tools
  - Good scalability
  - Allows debugging
- Disadvantages

Model-Checking: Pros & Cons

- Advantages
  - Decidability
  - May be fully automated
  - Better usability tools
  - Good scalability
  - Allows debugging
- Disadvantages
  - Limited Expressivity

Model-Checking: Pros & Cons

- Advantages
  - Decidability
  - May be fully automated
  - Better usability tools
  - Good scalability
  - Allows debugging
- Disadvantages
  - Limited Expressivity
    - Finite State Systems

Model-Checking: Pros & Cons

- Advantages
    - Decidability
    - May be fully automated
    - Better usability tools
    - Good scalability
    - Allows debugging
- Disadvantages
    - Limited Expressivity
        - Finite State Systems
        - Limited data structure

Sets of Rules

to describe

- reactive behaviour: user reacts to a stimulus that clearly indicates that a particular action should be taken (independently of the goal)

Sets of Rules

to describe

- reactive behaviour: user reacts to a stimulus that clearly indicates that a particular action should be taken (independently of the goal)
- communication goals: user knows a task-dependent mental list of information that must be communicated to the device

Sets of Rules

to describe

- reactive behaviour: user reacts to a stimulus that clearly indicates that a particular action should be taken (independently of the goal)
- communication goals: user knows a task-dependent mental list of information that must be communicated to the device
- completion: subsidiary tasks generated in achieving the goal

Sets of Rules

to describe

- reactive behaviour: user reacts to a stimulus that clearly indicates that a particular action should be taken (independently of the goal)
- communication goals: user knows a task-dependent mental list of information that must be communicated to the device
- completion: subsidiary tasks generated in achieving the goal
- abort: no rational action is available

Sets of Rules

to describe

- reactive behaviour: user reacts to a stimulus that clearly indicates that a particular action should be taken (independently of the goal)
- communication goals: user knows a task-dependent mental list of information that must be communicated to the device
- completion: subsidiary tasks generated in achieving the goal
- abort: no rational action is available

nondeterministic rules $\Longrightarrow$ rule disjunction

Reactive Behaviour

$$(stimulus\ t) \wedge NEXT\ reaction\ t$$

Reactive Behaviour

$$(stimulus\ t) \wedge NEXT\ reaction\ t$$

NEXT does not require that the action is taken on the next cycle, but rather that it is taken before any other user action

Reactive Behaviour

$$(\mathit{stimulus}\ t) \wedge \mathit{NEXT}\ \mathit{reaction}\ t$$

To target the generic user model to a particular device, it is applied to a concrete list in SML

$$[(\mathit{stimulus}_1, \mathit{reaction}_1);\ \ldots;\ (\mathit{stimulus}_n, \mathit{reaction}_n)]$$

Reactive Behaviour

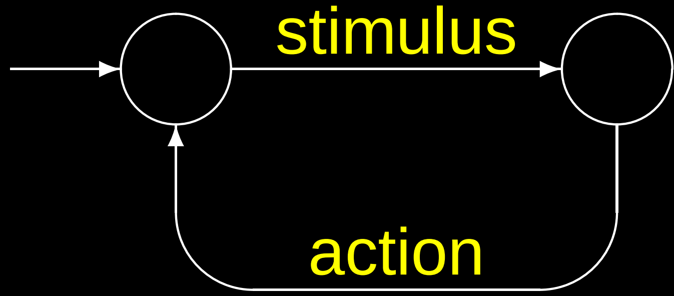
$$(stimulus\ t)\ \wedge\ NEXT\ reaction\ t$$

$$[(stimulus_1, reaction_1);\ \ldots;\ (stimulus_n, reaction_n)]$$

Example: $[(light, push\_button);\ (wait\_msg, pause);\ (card\_msg, take\_card);\ (cash\_msg, take\_cash)]$

Reactive Behaviour

$$(stimulus\ t) \wedge NEXT\ reaction\ t$$

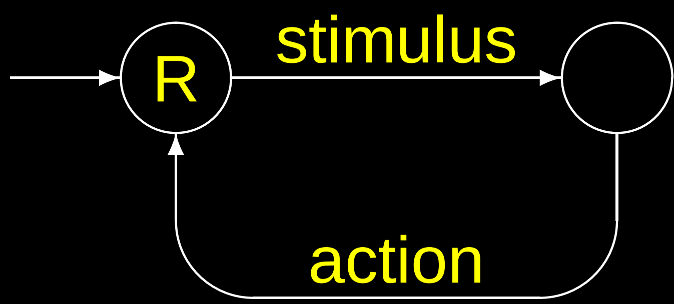


$$[(stimulus_1, reaction_1);\ \ldots;\ (stimulus_n, reaction_n)]$$

Example: $[(light, push\_button);\ (wait\_msg, pause);$
$(card\_msg, take\_card);\ (cash\_msg, take\_cash)]$

Reactive Behaviour

$$(stimulus\ t) \wedge NEXT\ reaction\ t$$



$R_1 = R/f_1$ where $f_1(\text{stimulus}) = \text{light}$
$f_1(\text{reactions}) = \text{push\_button}$

$$[(stimulus_1, reaction_1); \ldots; (stimulus_n, reaction_n)]$$

Example: $[(light, push\_button); (wait\_msg, pause); (card\_msg, take\_card); (cash\_msg, take\_cash)]$

Communication Goals

$$\sim (goalachieved\ t) \wedge (guard\ t) \wedge NEXT\ action\ t$$

Communication Goals

$\sim (goalachieved\ t) \wedge (guard\ t) \wedge NEXT\ action\ t$

Example: $[(has\_card, insert\_card); (TRUE, insert\_pin)]$

Communication Goals

$\sim (goalachieved\ t) \wedge (guard\ t) \wedge NEXT\ action\ t$

Example: $[(has\_card, insert\_card); (TRUE, insert\_pin)]$
Goal: $HasGotCash$

Communication Goals

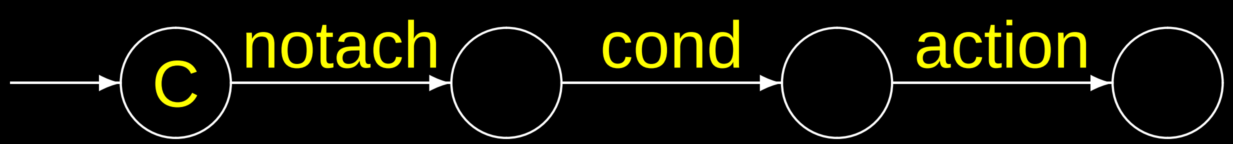
$$\sim (goalachieved\ t) \wedge (guard\ t) \wedge NEXT\ action\ t$$

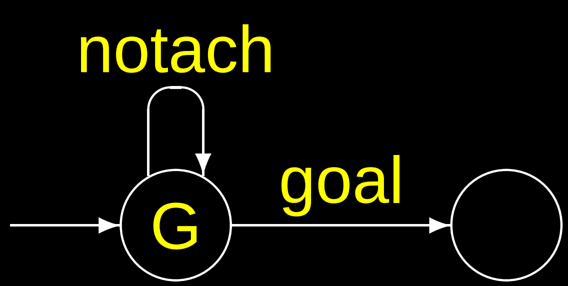
Example: $[(has\_card, insert\_card); (TRUE, insert\_pin)]$
Goal: $HasGotCash$

Communication Goals

$$\sim (goalachieved\ t) \wedge (guard\ t) \wedge NEXT\ action\ t$$

Example: $[(has\_card, insert\_card); (TRUE, insert\_pin)]$
Goal: $HasGotCash$

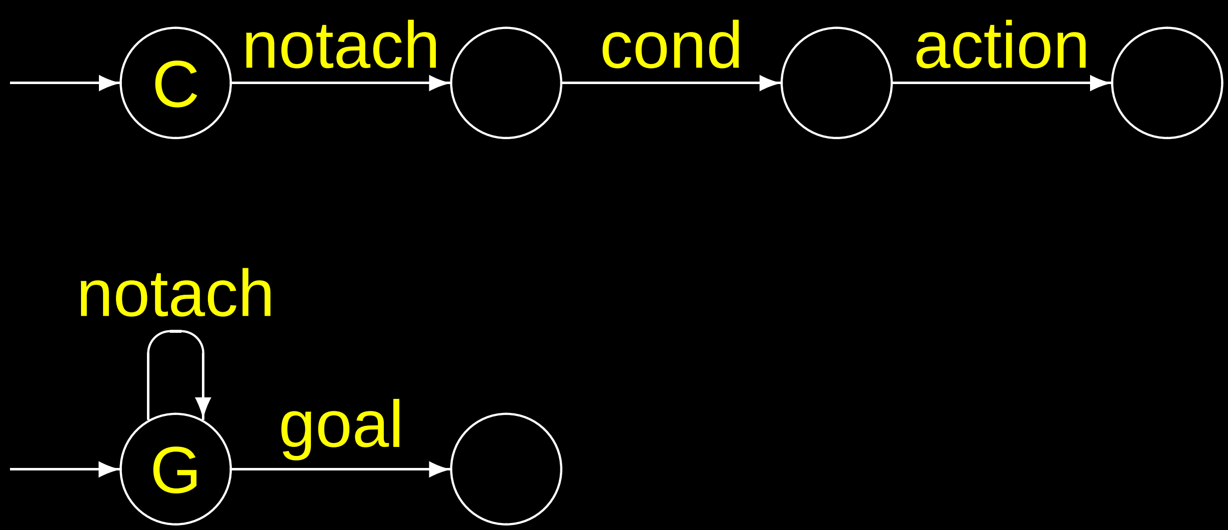




Completion

$$\mathit{if}\ (((\mathit{invariant}\ t) \wedge (\mathit{goalachieved}\ t)) \vee ((\mathit{finished}\ (t-1))$$
$$\mathit{then\ action\ finished}\ t$$
$$\mathit{else}\ \text{—}\mathit{disjunction\ of\ nondeterministic\ rules}\text{—}$$

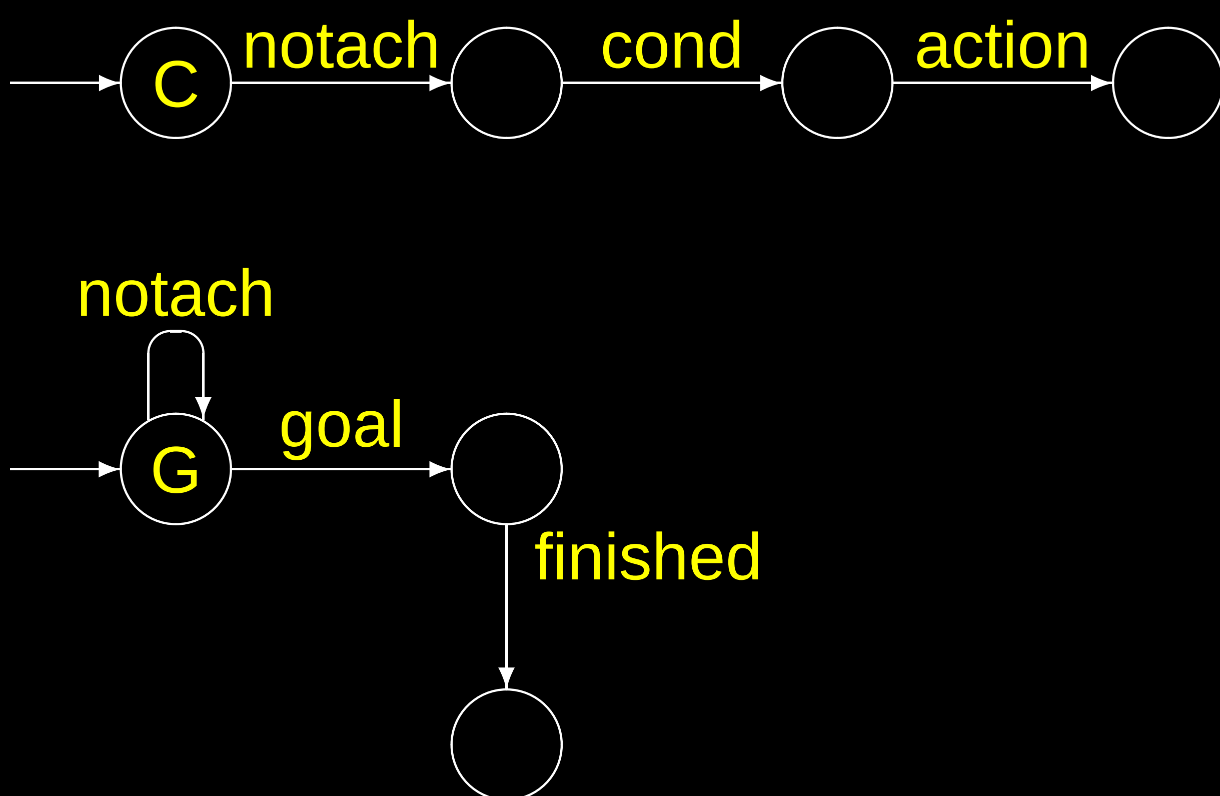
Invariant: $\mathit{VALUE\ possession}\ t \geq \mathit{VALUE\ possession}\ 1$



Completion

$$\textit{if } (((\textit{invariant } t) \wedge (\textit{goalachieved } t)) \vee ((\textit{finished } (t-1))$$
$$\textit{then action finished } t$$
$$\textit{else} \textit{ — disjunction of nondeterministic rules —}$$

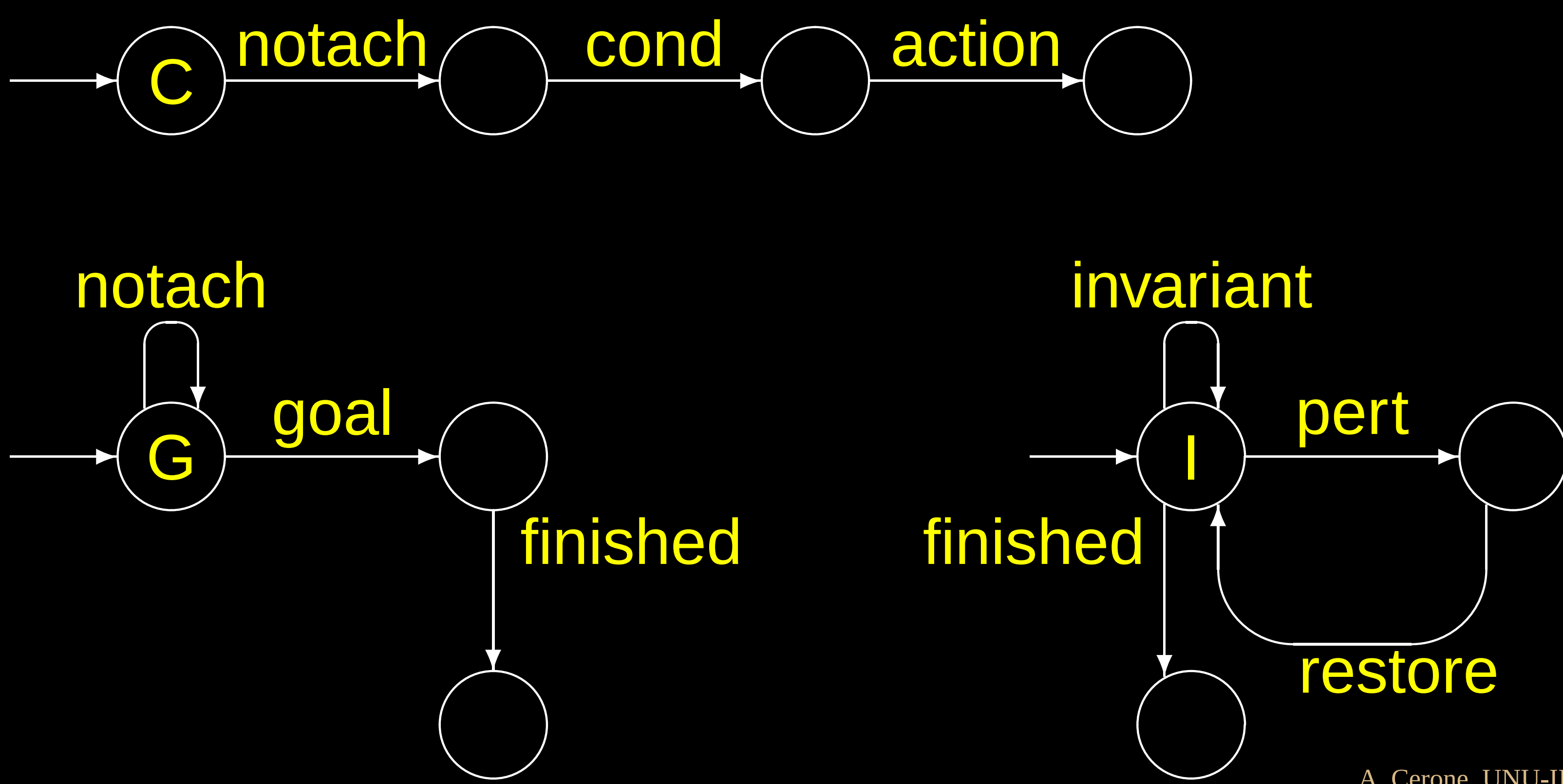
Invariant: $\textit{VALUE possession } t \geq \textit{VALUE possession } 1$



Completion

$\mathit{if}\ (((\mathit{invariant}\ t) \wedge (\mathit{goalachieved}\ t)) \vee ((\mathit{finished}\ (t-1))$
$\mathit{then\ action\ finished}\ t$
else — *disjunction of nondeterministic rules* —

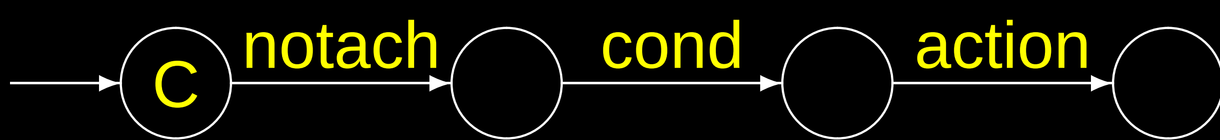
Invariant: $\mathit{VALUE\ possession}\ t \geq \mathit{VALUE\ possession}\ 1$

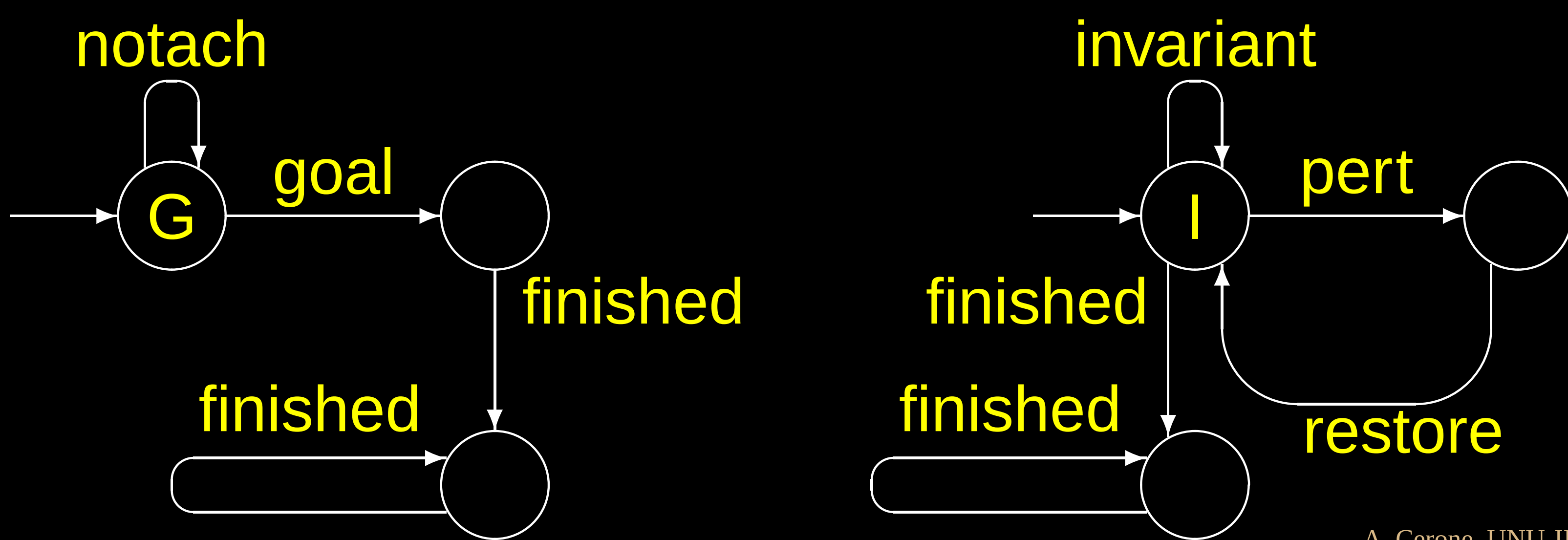


Completion

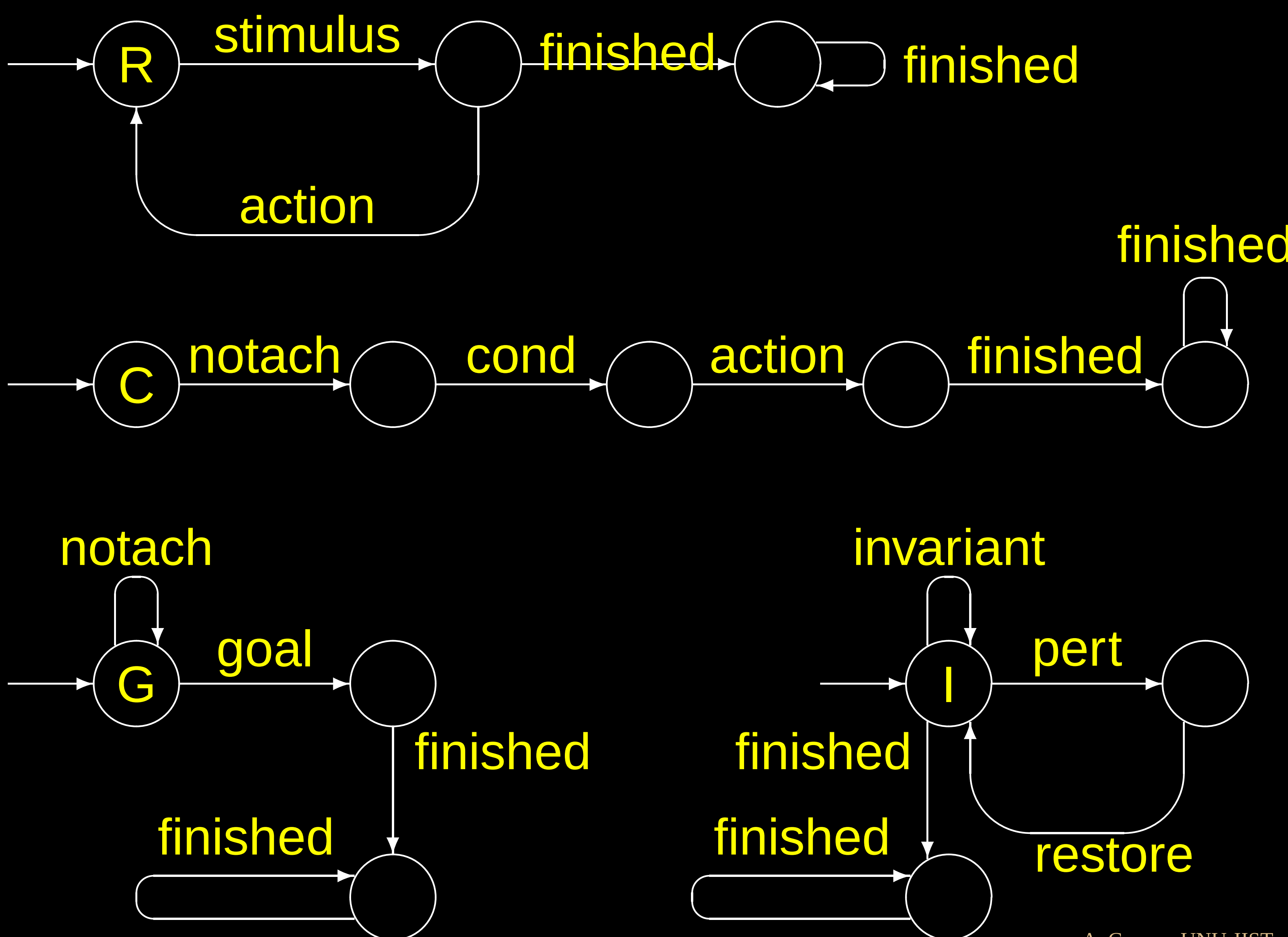
$$\textit{if } (((\textit{invariant } t) \wedge (\textit{goalachieved } t)) \vee ((\textit{finished } (t-1))$$
$$\textit{then action finished } t$$
$$\textit{else} \text{ — } \textit{disjunction of nondeterministic rules} \text{ — }$$

Invariant: $\mathit{VALUE}\ \mathit{possession}\ t \geq \mathit{VALUE}\ \mathit{possession}\ 1$





CSP Architecture



User Model

HOL

User Model

HOL

- Rule disjunction

User Model

- HOL
  - Rule disjunction
- CSP

User Model

HOL

Rule disjunction

CSP

$$
\begin{aligned}
\mathsf{U} \quad = \quad & \mathsf{R}_1 \;\|\; \ldots \;\|\; \mathsf{R}_m \;\| \\
& ((\mathsf{C}_1 \;\|\; \mathsf{G}) \,|[\{goal, finished\}]|\, \ldots \\
& \ldots \,|[\{goal, finished\}]|\, (\mathsf{C}_n \;\|\; \mathsf{G})) \;\| \\
& \mathsf{I}_1 \;\|\; \ldots \;\|\; \mathsf{I}_s
\end{aligned}
$$

User Model

HOL

Rule disjunction

CSP

$$
\begin{aligned}
\mathsf{U} \;=\; & \mathsf{R}_1 \;\|\; \ldots \;\|\; \mathsf{R}_m \;\| \\
& ((\mathsf{C}_1 \;\|\; \mathsf{G}) \,|[\{goal, finished\}]|\, \ldots \\
& \ldots \,|[\{goal, finished\}]|\, (\mathsf{C}_n \;\|\; \mathsf{G})) \;\| \\
& \mathsf{I}_1 \;\|\; \ldots \;\|\; \mathsf{I}_s
\end{aligned}
$$

What's missing?

EXERCISE

How to guarantee that all reactive behaviours and communication goals are performed atomically within the user model?

Formal Verification

HOL — Correctness Theorem

Formal Verification

HOL — Correctness Theorem

$$\begin{array}{l}
\forall\ state\ traces\ . \\
\quad initial\ state\ \wedge \\
\quad device\ specification\ \wedge \\
\quad user\ model \\
\qquad \supset\ \exists\, t\ .\ \ (invariant\ t)\ \wedge \\
\qquad\qquad\qquad (goalachieved\ t)
\end{array}$$

Formal Verification

HOL — Correctness Theorem

$$
\begin{array}{l}
\forall\ state\ traces\ .\\
\quad initial\ state\ \wedge\\
\quad device\ specification\ \wedge\\
\quad user\ model\\
\qquad \supset \exists\, t\, .\ (invariant\ t)\ \wedge\\
\qquad\qquad\qquad (goalachieved\ t)
\end{array}
$$

CSP

Formal Verification

HOL — Correctness Theorem

$$\begin{array}{l} \forall\ \mathit{state}\ \mathit{traces}\ . \\ \quad \mathit{initial}\ \mathit{state}\ \wedge \\ \quad \mathit{device}\ \mathit{specification}\ \wedge \\ \quad \mathit{user}\ \mathit{model} \\ \qquad \supset \exists\, t\ .\ (\mathit{invariant}\ t)\ \wedge \\ \qquad\qquad\quad (\mathit{goalachieved}\ t) \end{array}$$

CSP

– Overall system

$\mathsf{OverallSystem} = \mathsf{U}\ \|\ \mathsf{Device}$

Formal Verification

HOL — Correctness Theorem

$$\begin{array}{l} \forall\ \mathit{state}\ \mathit{traces}\ . \\ \quad \mathit{initial}\ \mathit{state}\ \wedge \\ \quad \mathit{device}\ \mathit{specification}\ \wedge \\ \quad \mathit{user}\ \mathit{model} \\ \qquad \supset\ \exists\, t\, .\ \ (\mathit{invariant}\ t)\ \wedge \\ \qquad\qquad\qquad (\mathit{goalachieved}\ t) \end{array}$$

CSP

– Overall system

$\mathsf{OverallSystem} = \mathsf{U}\ \|\ \mathsf{Device}$

– Temporal Logic Formula

$\Box\Diamond\,\mathsf{finished}$ checked on OverallSystem

Classes of User Errors

Errors detected by attempts to prove the task completion error

Classes of User Errors

Errors detected by attempts to prove the task completion error

Classes of errors defined in terms of their cognitive causes rather than their effects:

- post-completion errors
- communication-goal errors
- device delay errors

Post-completion Errors

User terminates an interation with outstanding tasks remaining

Post-completion Errors

User terminates an interation with outstanding tasks remaining

It emerges because of a rule allowing the user to stop once the goal is achieved

Post-completion Errors

User terminates an interation with outstanding tasks remaining

It emerges because of a rule allowing the user to stop once the goal is achieved

Design Principle: goal cannot be achieved until after the interaction invariant is restored

Post-completion Errors

User terminates an interation with outstanding tasks remaining

It emerges because of a rule allowing the user to stop once the goal is achieved

Design Principle: goal cannot be achieved until after the interaction invariant is restored

Error still present if a warning after goal achieved remind the user to do the completions tasks

Communication Errors

User discharges communication goals in an order different from the one required by the device

Communication Errors

User discharges communication goals in an order different from the one required by the device

It emerges because of communication rule removed too early $\Longrightarrow$ activate abort rule

Communication Errors

User discharges communication goals in an order different from the one required by the device

It emerges because of communication rule removed too early $\Longrightarrow$ activate abort rule

Design Principle: the device must not require a specific order for the communication goal actions

Communication Errors

User discharges communication goals in an order different from the one required by the device

It emerges because of communication rule removed too early $\Longrightarrow$ activate abort rule

Design Principle: the device must not require a specific order for the communication goal actions

Error still present if a message tell the user the right order

Device Delay Errors

User discharges outstanding communication goals during device delay

Device Delay Errors

User discharges outstanding communication goals during device delay

It emerges because of communication rule removed too early $\Longrightarrow$ activate abort rule

Device Delay Errors

User discharges outstanding communication goals during device delay

It emerges because of communication rule removed too early $\Longrightarrow$ activate abort rule

Design Principle: the device delay can only occur when there is no outstanding communicatin goal in the presence of a “wait” warning causing a “pause” reaction

TP vs. MC

- Theorem Proving

TP vs. MC

- Theorem Proving
  - automated correctness proof

TP vs. MC

- Theorem Proving
  - automated correctness proof
  - informal reasoning to detect errors

TP vs. MC

- Theorem Proving
  - automated correctness proof
  - informal reasoning to detect errors
- Model Checking more promising

TP vs. MC

- Theorem Proving
  - automated correctness proof
  - informal reasoning to detect errors
- Model Checking more promising
  - general correctness: $\Box\Diamond$ finished

TP vs. MC

- Theorem Proving
  - automated correctness proof
  - informal reasoning to detect errors
- Model Checking more promising
  - general correctness: $\Box\Diamond$ finished
  - post-completion correctness: $\Box\Diamond$ invariant

TP vs. MC

- Theorem Proving
  - automated correctness proof
  - informal reasoning to detect errors
- Model Checking more promising
  - general correctness: $\Box\Diamond$ finished
  - post-completion correctness: $\Box\Diamond$ invariant
  - automatically generated counterexample allows error detection and correction

Examination — Architectures

SOAR and PUMA

- Seminars
  - Theorem proving and informal reasoning for usability studies
  - The SOAR cognitive architecture
- Reports
  - CSP model of the PUMA cognitive architecture

Examinations

Seminar 1 — Full OCM for ATC

Topic: Operator Choice Model for ATC

Full OCM model for the ATC

- D. Leadbetter, P. Lindsay, A. Hussey, A. Neal and M. Humphreys
  Towards Towards Model Based Prediction of Human Error Rates in Interactive Systems, 2000
- A. Hussey, D. Leadbetter, P. Lindsay, A. Neal and M. Humphreys
  Modelling and Hazard Identification in an Air-Traffic Control User-Interface, 2000
- S. Connelly, P. Lindsay, A. Neal and M. Humphreys
  A formal model of cognitive processes for an Air Traffic Control Task, 2001

Seminar 2 — Mode Confusion

Topic: Mode Confusion

Formal analysis of mode confusion

- S. P. Miller and J. N. Potts
  Detecting Mode confusion Through formal Modelling and Analysis,
  1999
- N. Leveson, L. D. Pinnel, S. D. Sandys, S. Koga and J. D. Reese
  Analysing Software Specification for Mode Confusion Potential,
  1998
- R. W. Butler, S. P. Miller, J. N. Potts and V. A. Carreno
  A Formal Methods Approach to the Analysis of Mode Confusion,
  1998

Seminar 3 — PUMA

Topic: PUMA Work

Theorem proving and informal reasoning for usability studies

- P. Curzon and A. Blandford
  Detecting Multiple Classes of User Errors, 2001
- P. Curzon and A. Blandford
  From a Formal User Model to Design Rules, 2002
- P. Curzon and A. Blandford
  Formally Justifying User-Centred Design Rules: a Case Study on Post-completion Error, 2004

Seminar 4 — SOAR

Topic: SOAR

The SOAR cognitive architecture (for one or more PhD students)

- SOAR Home Page
  http://sitemaker.umich.edu/soar/
- The SOAR Tutorial
  http://sitemaker.umich.edu/soar/documentation_and_links
  Part 1 | Part 2 | Part 3 | Part 4 | Part 5 | Part 6

Report 1 — FM of Full ATC

Topic: Operator Choice Model

Formal model of the full OCM for ATC

using CSP or other formalism, possibly running simulation using a tool

- D. Leadbetter, P. Lindsay, A. Hussey, A. Neal and M. Humphreys
  Towards Model Based Prediction of Human Error Rates in Interactive Systems, 2000
- S. Connelly, P. Lindsay, A. Neal and M. Humphreys
  A formal model of cognitive processes for an Air Traffic Control Task, 2001
- Antonio Cerone, Simon Connelly and Peter Lindsay.
  Formal Analysis of Operator Behavioural Patterns in Interactive Systems, submitted

Report 2 — Cooperative TM

Topic: Task Models

Formal Analysis of Cooperative Task Models

Discussion of the papers' differences and limitations and propose possible extensions

- F. Paternò, C. Santoro and S. Thamassebi
  Formal models for Cooperative Tasks: Concepts and an Application for En-route Air Traffic Control
- V. M. R. Penichlet, F. Paternò, J. A. Gallud and M. D. Lozano
  Collaborative Social Structures and Task Modelling Integration
- D. Pinelle and C. Gutwin
  Task Analysis for Groupware Usability Evaluation: Modeling Shared Workplace Tasks with Mechanics of cCllaboration

Report 3 — PUMA

Topic: PUMA Work

CSP model of the PUMA cognitive architecture

Build a case study and formally analyse it with a tool (e.g. CWNC)

- P. Curzon and A. Blandford
  Using a Verification System to Reason about Post-Completion Errors, 2000
- P. Curzon and A. Blandford
  Reasoning about Order Errors in Interaction, 2000
- P. Curzon and A. Blandford
  Detecting Multiple Classes of User Errors, 2001

References

Cognitive Architecturess

- []:

[Newel et al. 91]

Allen Newell, Gregg Yost, John Laird, Paul Rosembloom, Ekkehard Altmann.
Formaulating the problem-space computational model.
In R. F. Rashid (ed) *CMU Computer Science: a 25th Anniversary Commemorative*, Chapter 11, ACM Press, 1991.

- Problem Space
- Cognitive Architectures

[Newel et al. 87]

Allen Newell, John Laird, Paul Rosembloom.
SOAR: an architecture for general intelligence.
Artificial Intelligence 33, 1987, pages 1–64.

- SOAR

[Young et al. 89]

Richard Young, T. R. G. Green, Tony Simon. *Programmable user models for predictive evaluation of interface design*. In K. Bice and G. Lewis (eds) *Proceedings of CHI'89: Human Factors in Computing Systems*, ACM Press, 1989, pages 15–19.

- PUM

[Curzon et al. 01]

Paul Curzon, Ann Blandford.
Detecting multiple classes of user errors.
In *Proceedings of EHCI'01*, LNCS 2254,
Springer, 2001, pages 57–71.

- Detecting Errors