Introduction to Model Checking

Radu Mateescu
Inria – Univ. Grenoble Alpes – LIG

What is model checking?

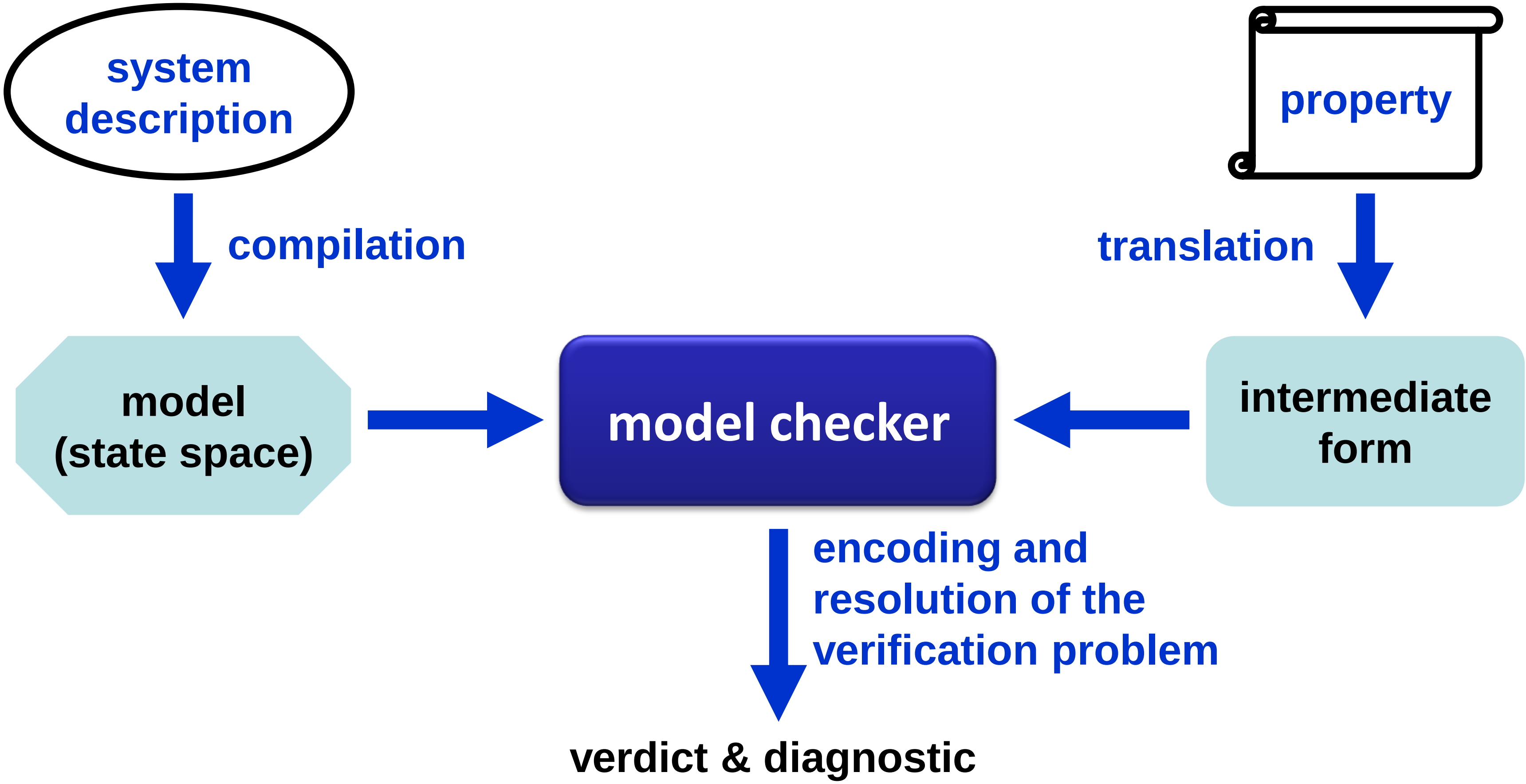
"*Model checking is the method by which a desired **behavioral property** of a **reactive system** is verified over a given system (the **model**) through exhaustive enumeration (**explicit** or **implicit**) of all the **states** reachable by the system and the **behaviors** that traverse through them*."

Amir Pnueli

Foreword to *Model Checking*

[Clarke-Grumberg-Peled-00]

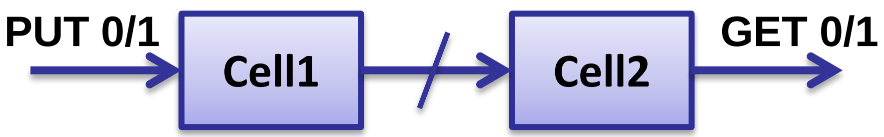
Basic model checking flow



Running example

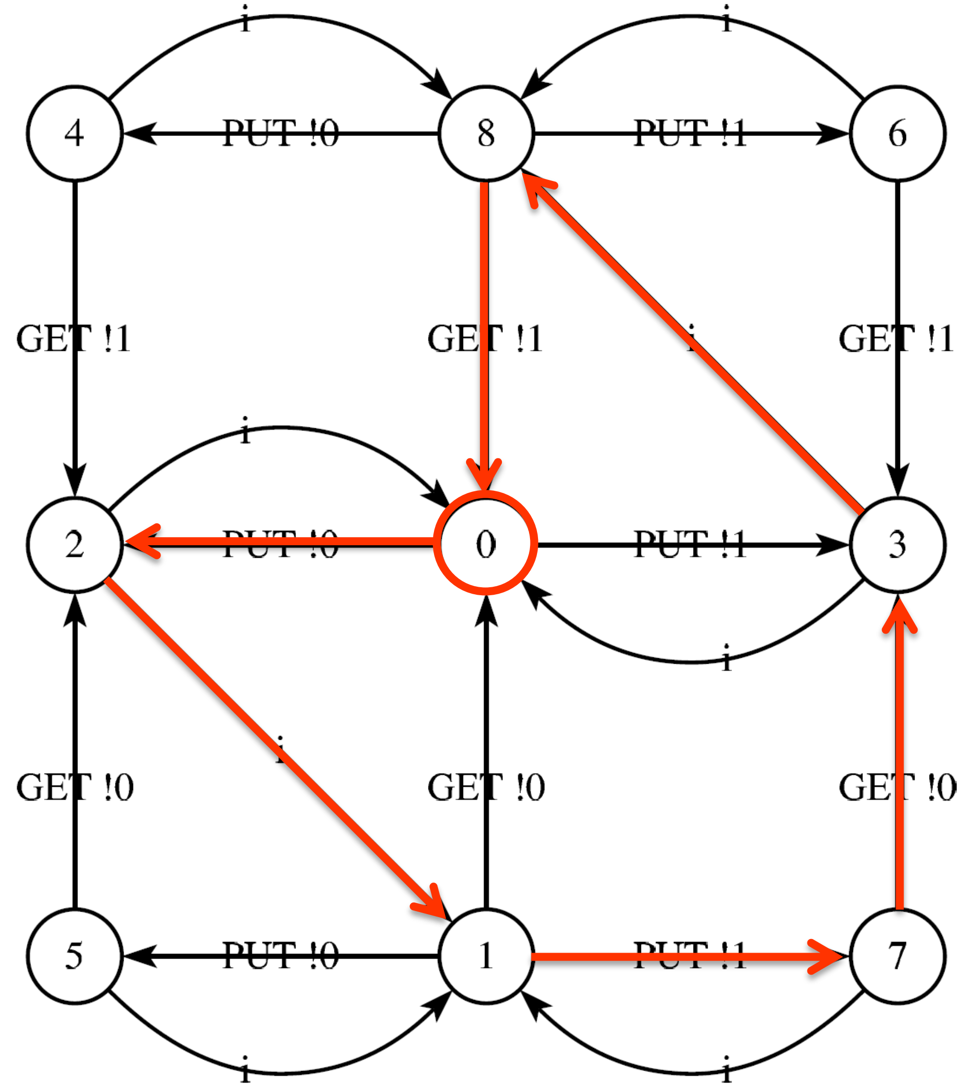
(action-based version)

- Two-cell buffer with unreliable transmission



- 9 states, 20 transitions

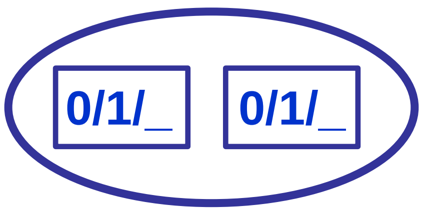
action-based setting
(Labelled Transition System)



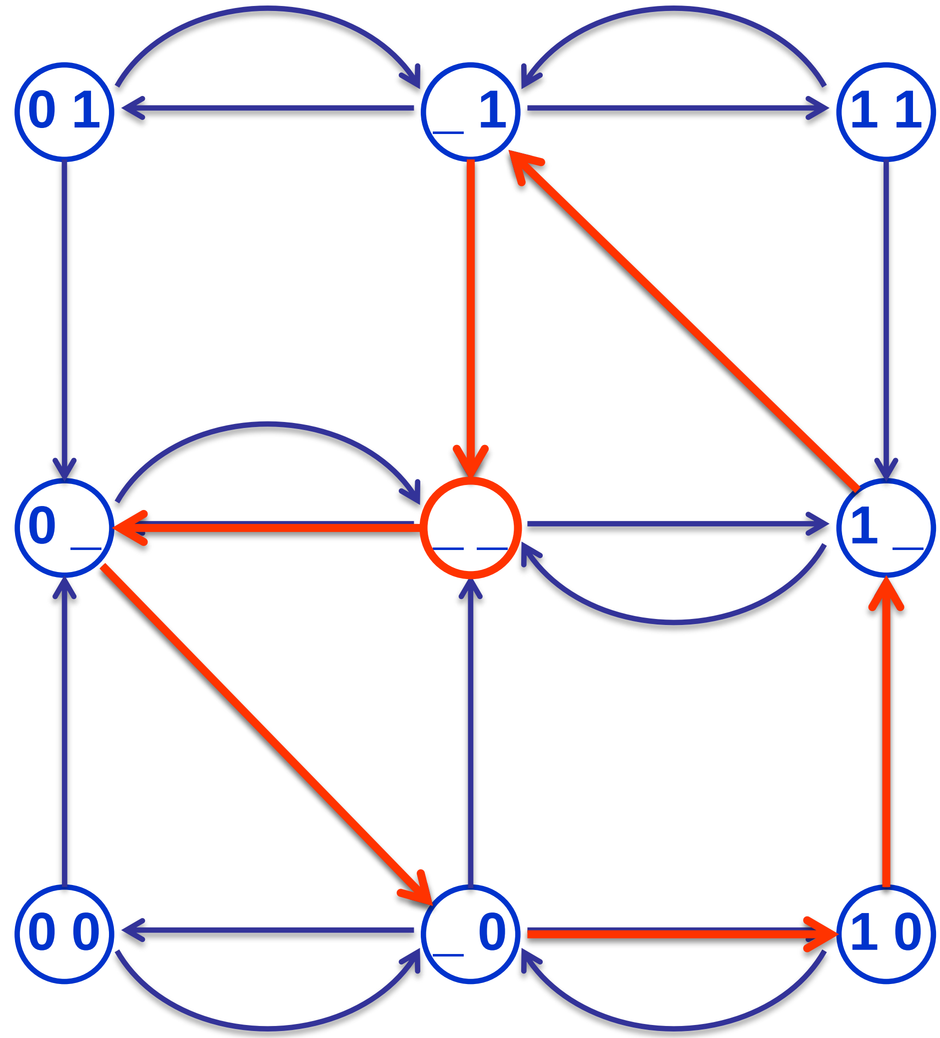
Running example

(state-based version)

- Keep the contents of states and the transitions between them



state-based setting
(Kripke structure)



States *vs* actions

State-based

- White box spec style
- Predicates on state variables
- Stuttering equivalence
- Partial order reductions

Action-based

- Black box spec style
- Predicates on actions/events
- Weak bisimulations
- Compositionality (congruences w.r.t. ||)

Kripke transition systems (KTS)
state variables and actions

Specification of temporal properties

- Temporal logic [Pnueli-77]:

  formalism for describing evolutions of program states over (logical) time

  – Atomic propositions over states
  – Propositional logic operators (or, and, not, …)
  – Tense operators (neXt, Until, Previous, Since, Once, …)
  – Interpreted on state spaces

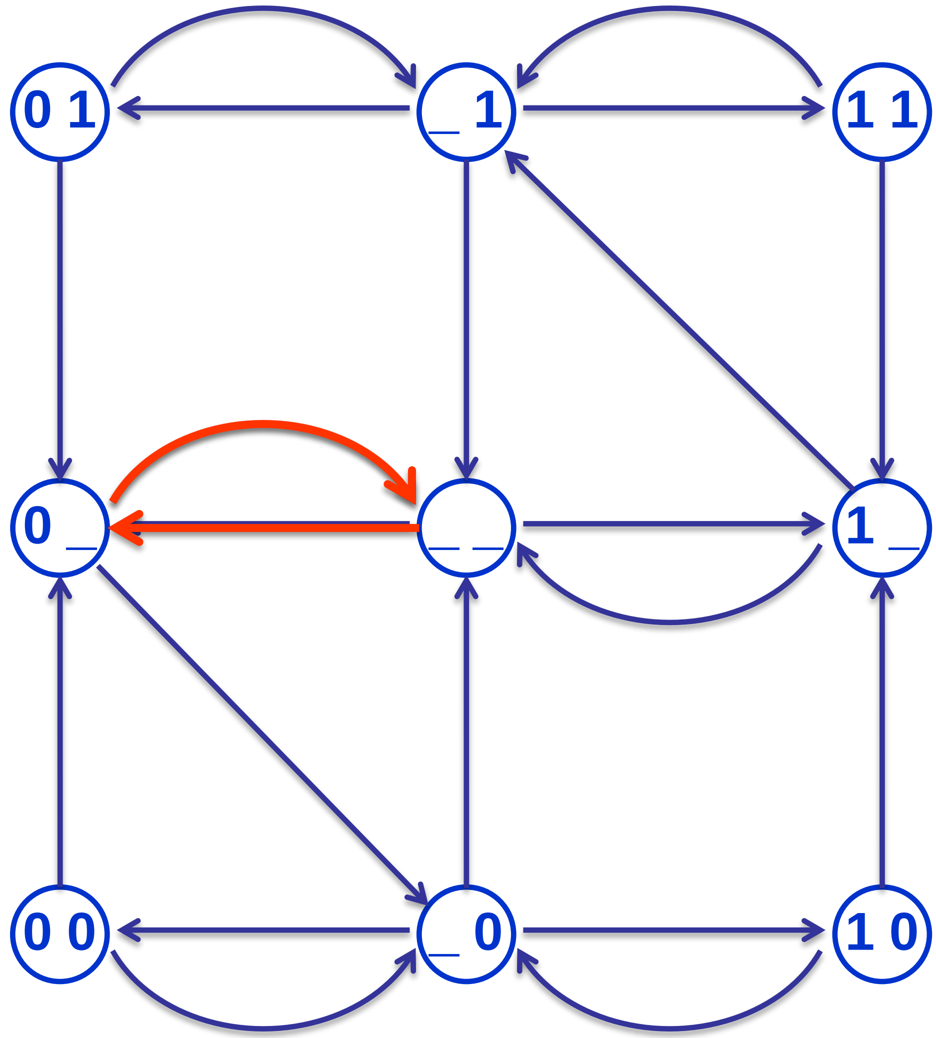
- High-level specification style:

  abstraction and ***modularity***

Properties on states and branches

(CTL – Computation Tree Logic)

- **X** φ, **E** [φ_1 **U** φ_2], **A** [φ_1 **U** φ_2]
  EF φ = **E** [true **U** φ]
  (potentiality)
  AG φ = ¬ **EF** ¬ φ
  (invariance)
  AF φ = **A** [true **U** φ]
  (inevitability)
  EG φ = ¬ **AF** ¬ φ
  (trajectory)
- **AG** (s0∗ => **EF** s∗0) ***ok***
- **AG** (s0∗ => **AF** s∗0) ***ko***



Properties on states and paths

(LTL – Linear Temporal Logic)

- **X** ψ, ψ_1 **U** ψ_2

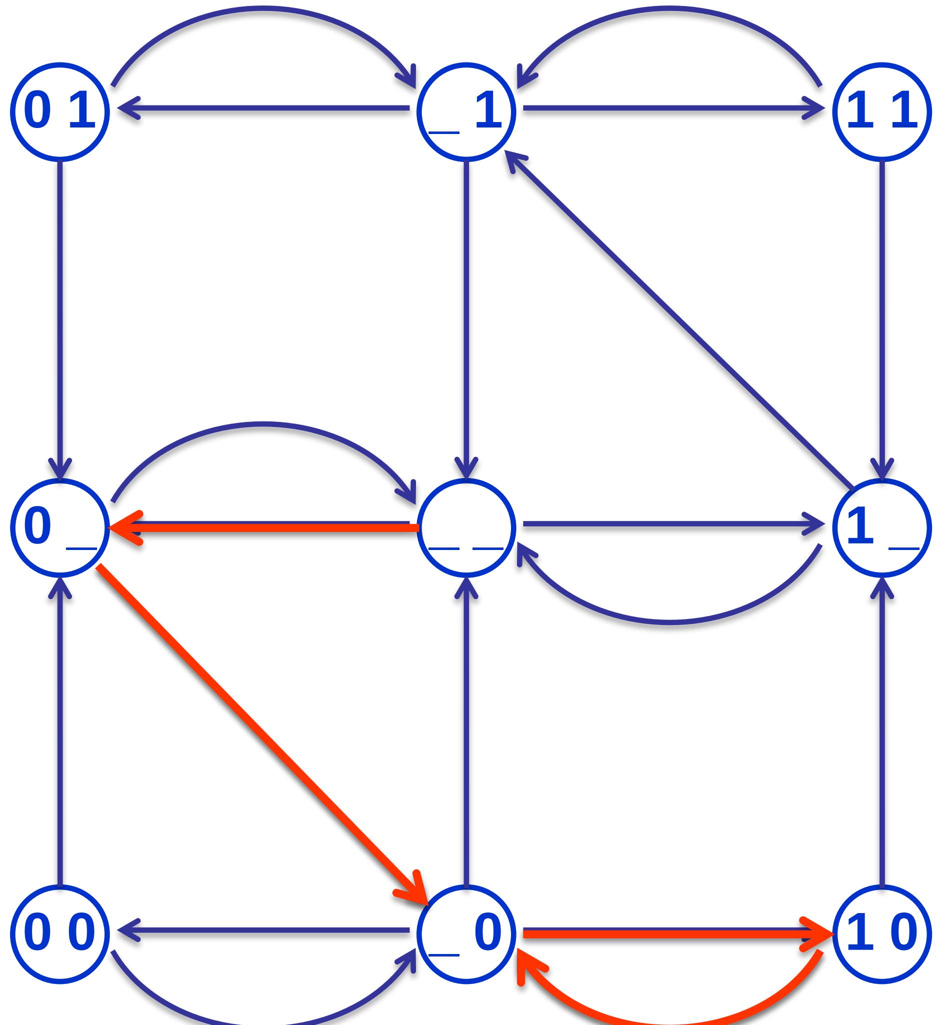
  F ψ = true **U** ψ
  (eventually)

  G ψ = ¬ **F** ¬ψ
  (globally)

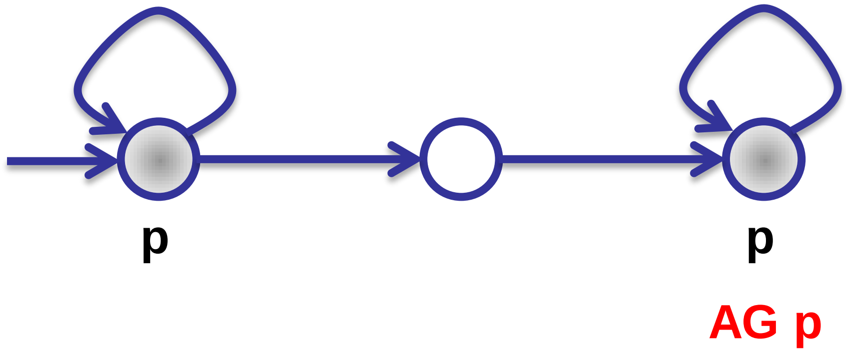
  ψ_1 **R** ψ_2 = ¬ (¬ψ_1 **U** ¬ψ_2)
  (release)

- **GF** (s0_ V s1_ V s_0 V s_1) ***ok***

- **FG** s_ _ ***ko***

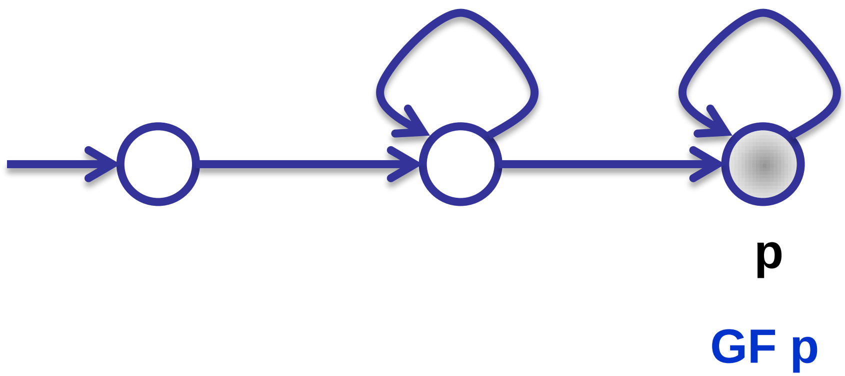


LTL *vs* CTL



$\models$ A (FG p)

$\not\models$ AF AG p

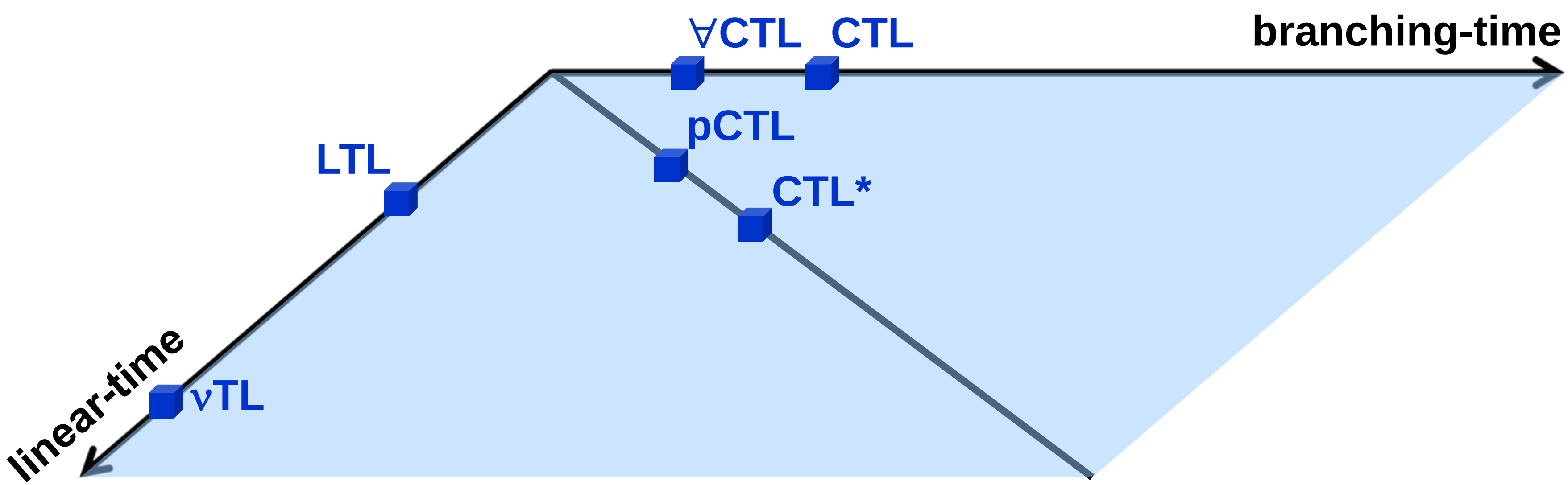


$\not\models$ A (GF p)

$\models$ AG EF p

the two logics are uncomparable

Linear-time *vs* branching-time



Properties on actions

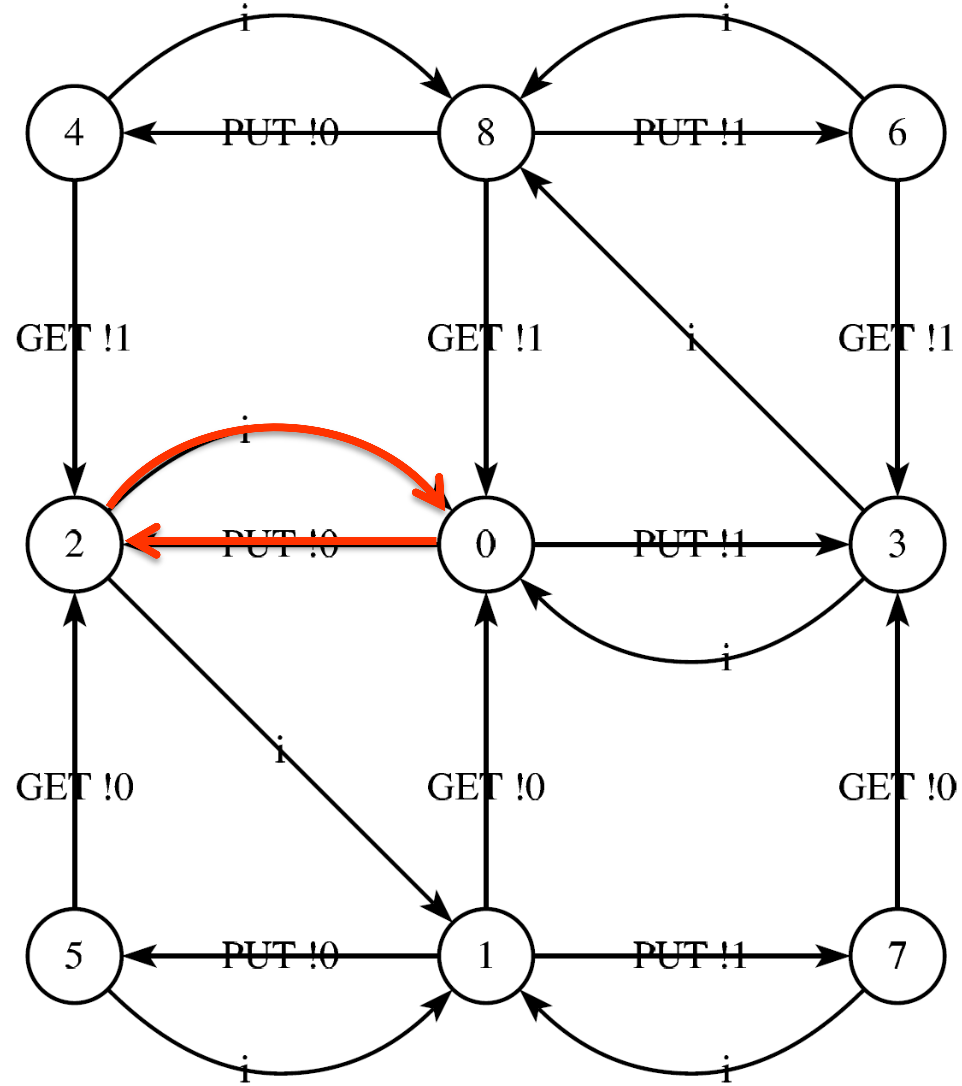
(ACTL – Action-based CTL)

- $\mathbf{AG}_{\text{true}}\ [\text{PUT}_0]$
  $\mathbf{E}\ [\text{true}_{\text{true}}\ \mathbf{U}_{\text{GET0}}\ \text{true}]$

  ok

- $\mathbf{AG}_{\text{true}}\ [\text{PUT}_0]$
  $\mathbf{A}\ [\text{true}_{\text{true}}\ \mathbf{U}_{\text{GET0}}\ \text{true}]$

  ko



Properties on actions

($L\mu$ – modal μ-calculus)

- "Assembly language" for temporal operators
  - Modalities and fixed point operators
  - Hierarchy of fragments $L\mu_k$ with alternation depth k
  - Captures virtually all existing TL operators

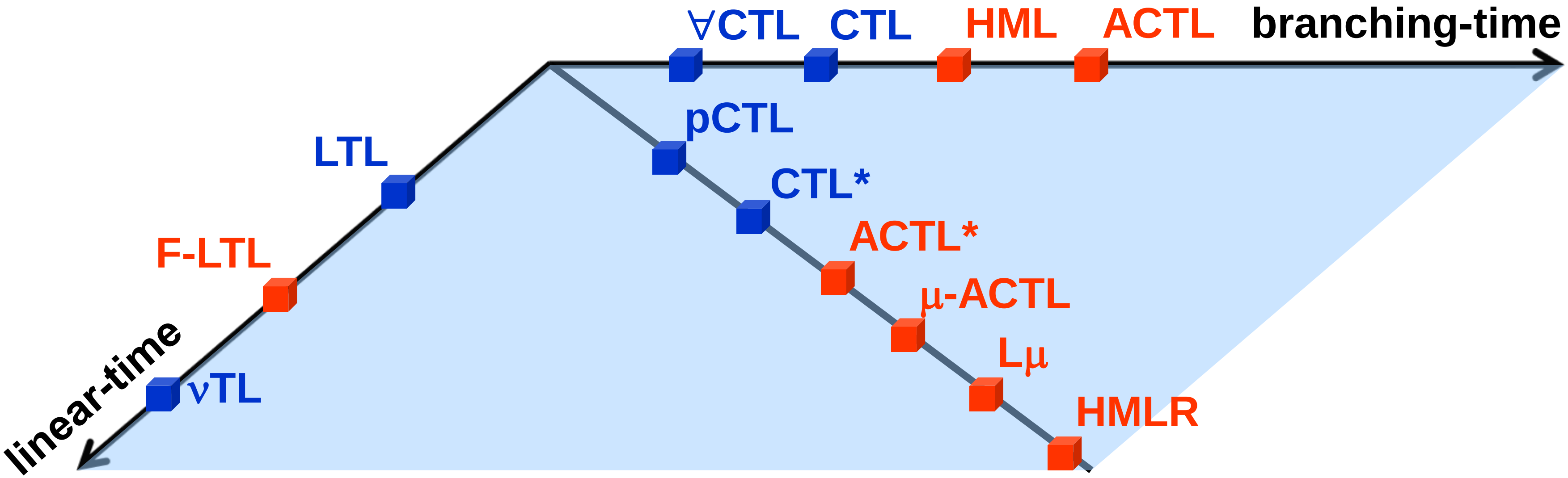
$\mathbf{E}\,[\varphi_1\ \mathbf{U}\ \varphi_2] = \mu X\ .\ \varphi_2 \vee (\varphi_1 \wedge <\mathrm{true}> X)$ **$L\mu_1$**

(CTL)

$\mathbf{AFG}\ \varphi = \neg \nu X\ .\ \mu Y\ .\ (\neg\varphi \wedge X) \vee <\mathrm{true}> Y$ **$L\mu_2$**

(LTL)

State-based *vs* action-based



Extensions with regular features

- Regular expressions / automata
  - Natural description of regular paths

Safety: FIFO buffer policy

$[\text{true}^*.\text{PUT}_0.(\neg\text{GET})^*.\text{PUT}_1.(\neg\text{PUT})^*.\text{GET}_1.(\neg\text{PUT})^*.\text{GET}_0]\,\textbf{false}$

(PDL)

$\nu X\,.\,([\text{PUT}_0]\;\nu Y\,.\,(([\text{PUT}_1]\;\nu Z\,.\,(([\text{GET}_1]\;\nu W\,.\,([\text{GET}_0]\;\text{false} \wedge [\neg\text{PUT}]\;W) \wedge [\neg\text{PUT}]\;Z) \wedge [\neg\text{GET}_0]\;Y) \wedge [\text{true}]\;X)$

(Lμ_1)

Extensions with data

- Handling of data values present in states/actions

  Safety: capacity of (reliable) 2-buffer

  [true*. (PUT . (¬GET)*) {3} **] false**

  regexp with counter

- Parametric formulas (stable w.r.t. model)

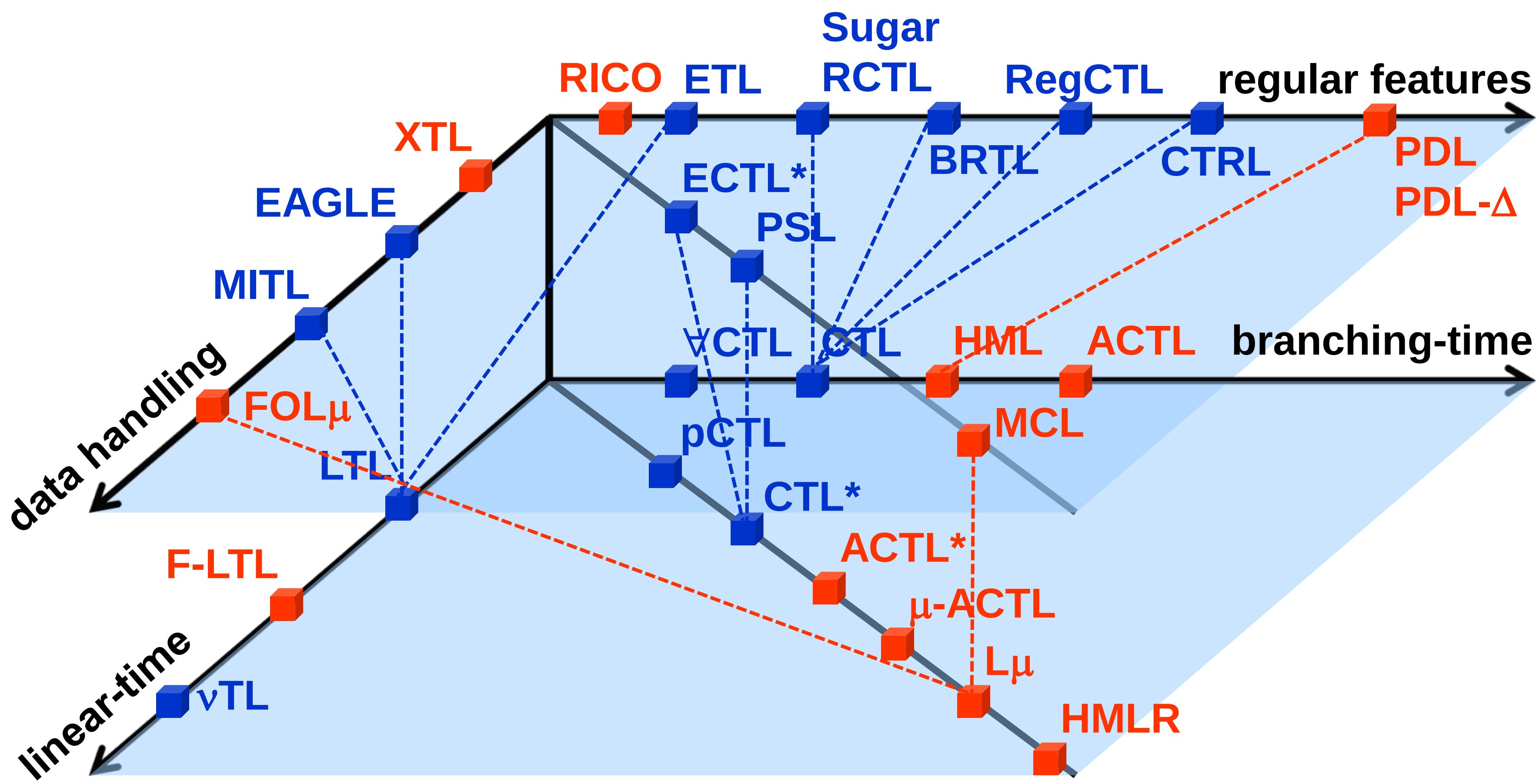
  Response: fair reachability of message delivery

  [true*. {PUT ?m:nat} **] <** true*. {GET !m} **> true**

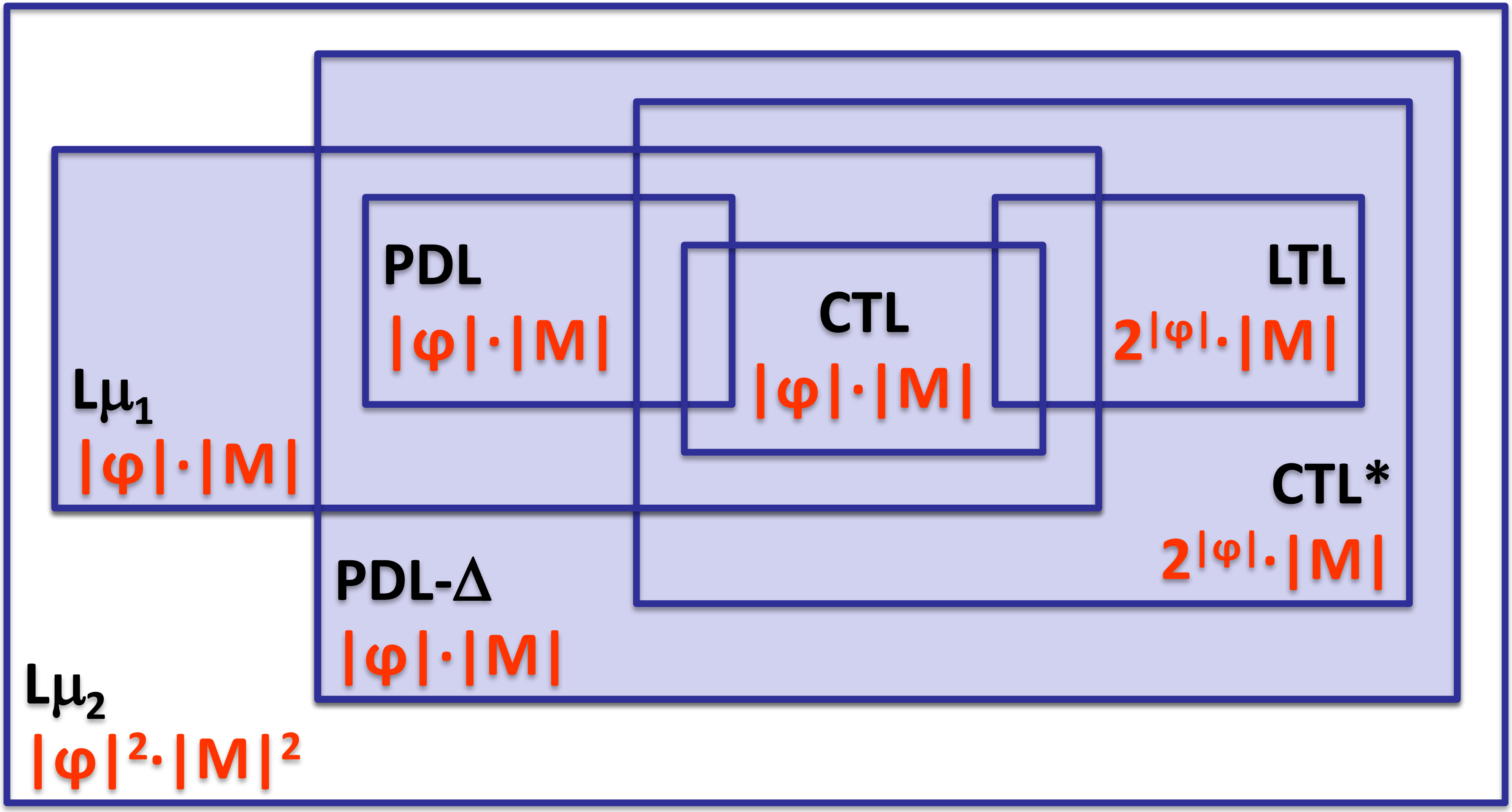
  variable propagation

Ergonomic extensions

(regular constructs and data handling)



Expressiveness and complexity



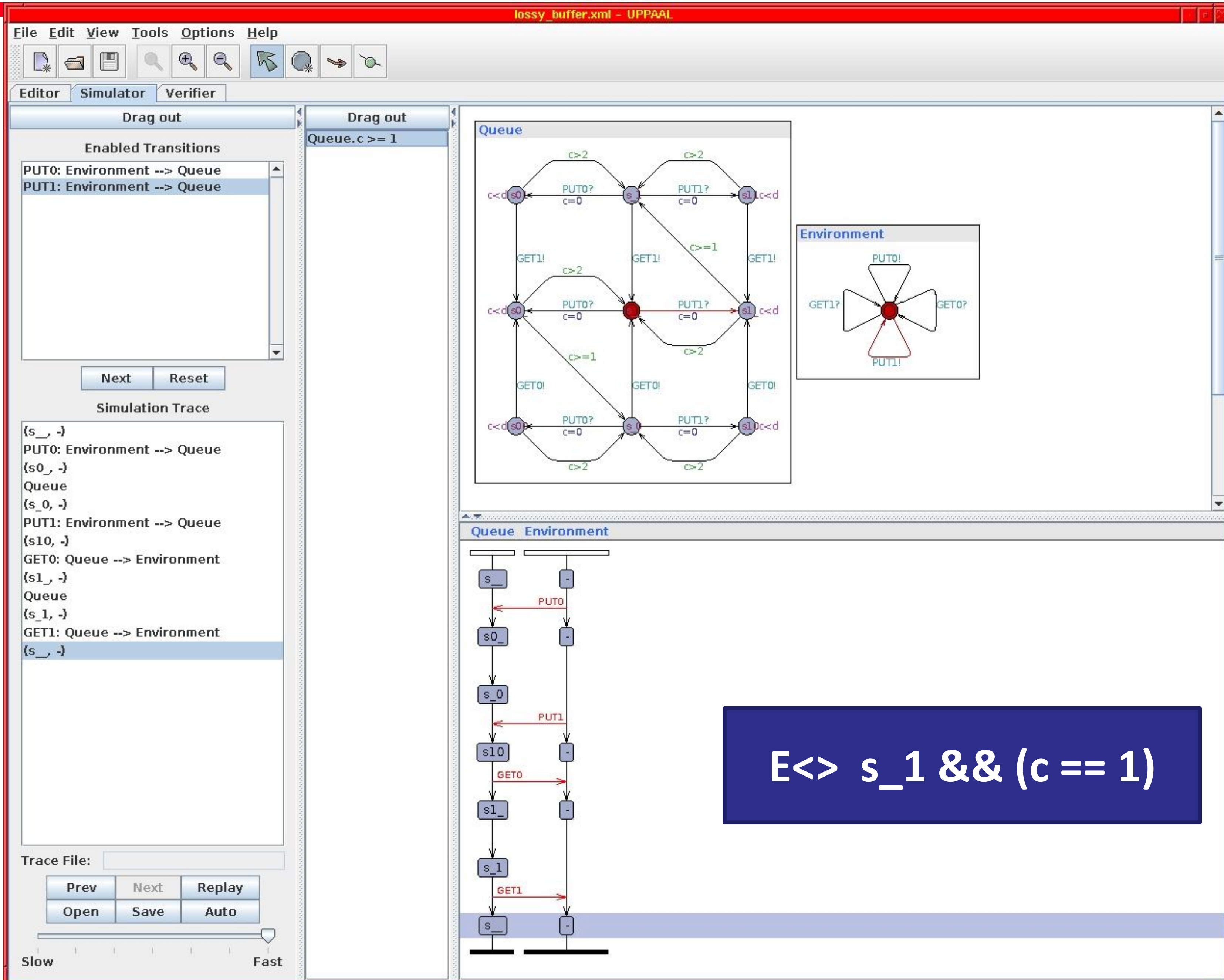
Quantitative properties

Time
(TA, TPN)

Rates
(CTMC, MDP)

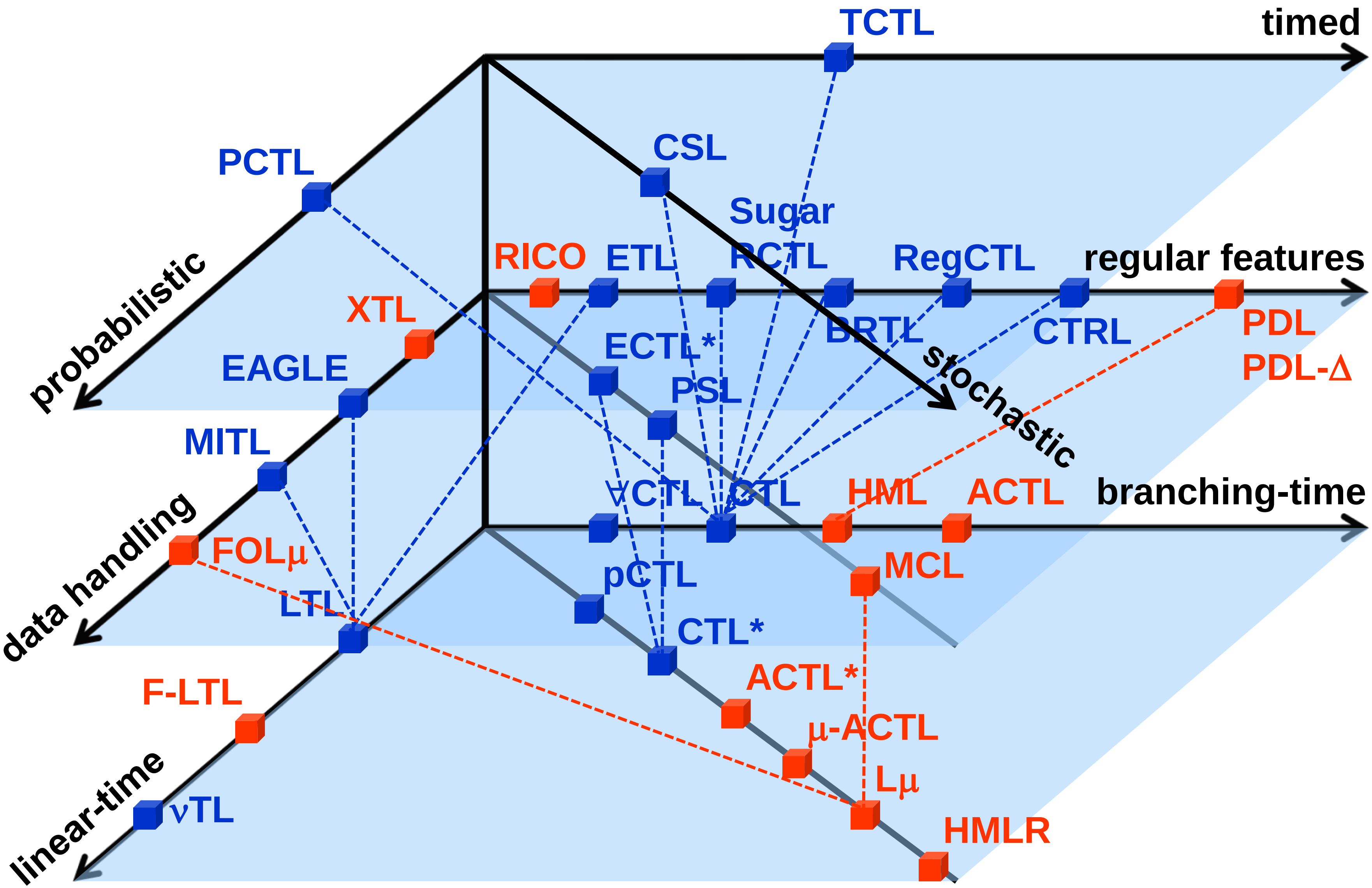
Probabilities
(DTMC)

(courtesy of Wendelin Serwe)



E<> s_1 && (c == 1)

Temporal logic zoo



How to choose the right TL?

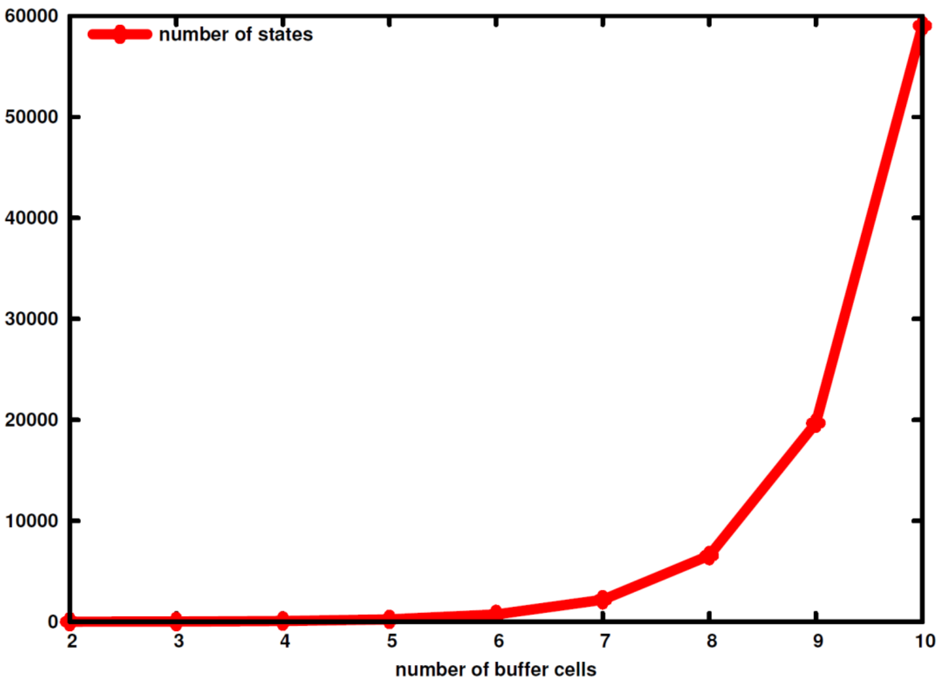
- Nature of the system and its properties:

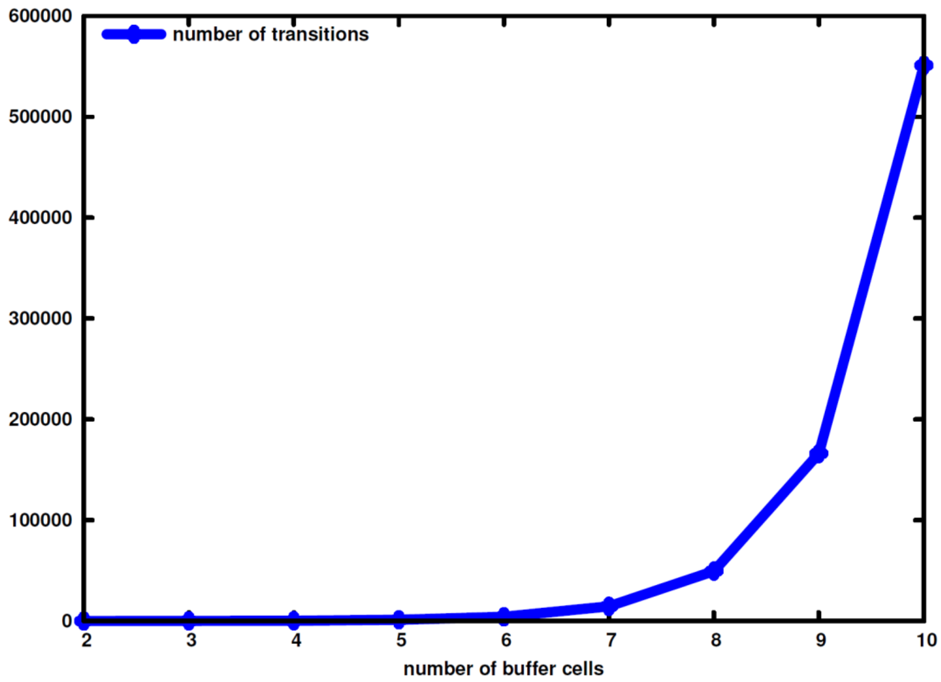
linear / branching	state / action
functional / quantitative	discrete / continuous

- Expressiveness vs model checking complexity
  - Tradeoff is often made in the available tools
- User-friendliness
  - Built-in ergonomic extensions (regexps, data)
  - Tools often provide libraries of derived operators
  - Use of property pattern libraries [Dwyer-et-al-99]

State space explosion

- Exponential growth of the state space with the number of parallel processes

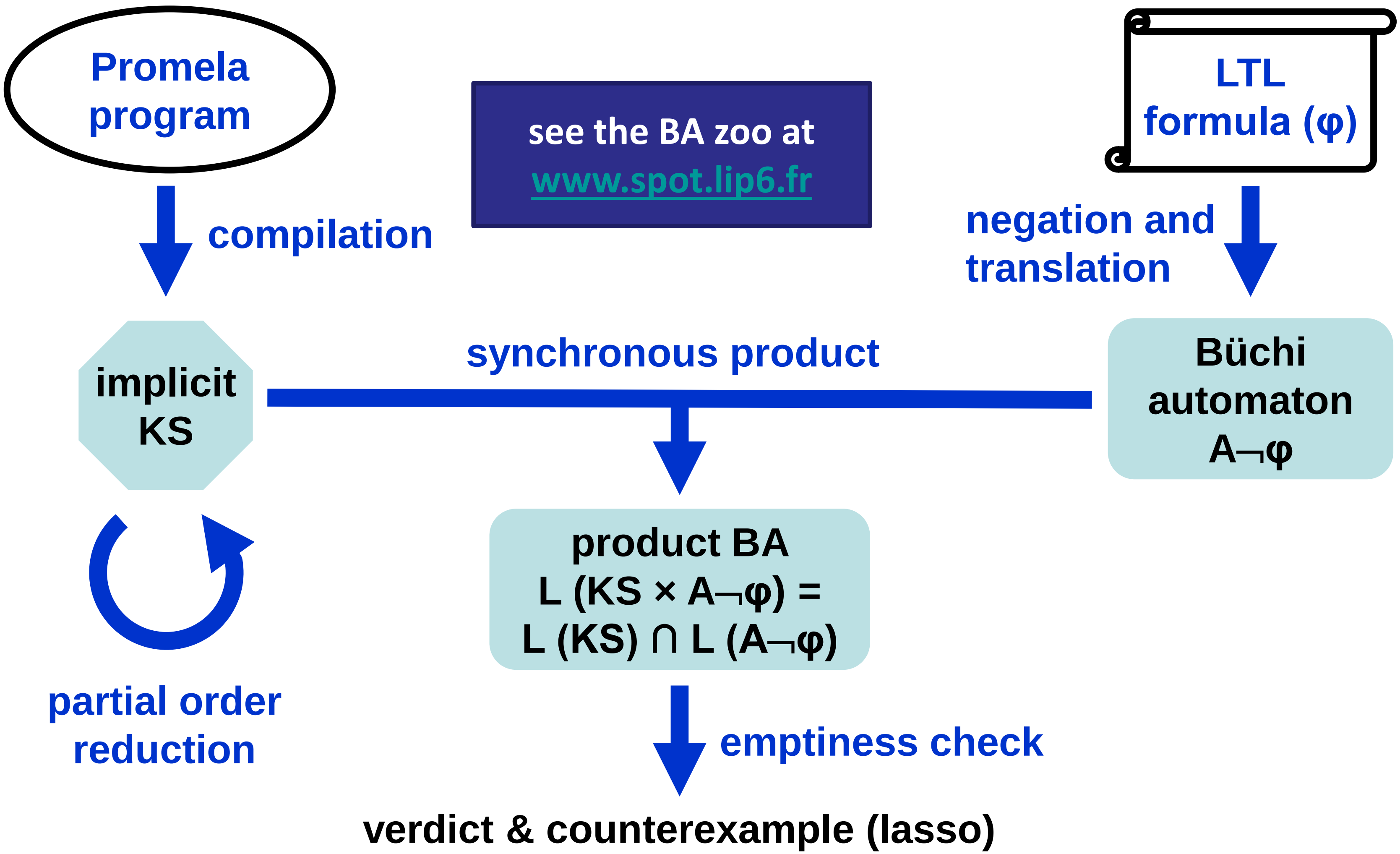




Model checking holy grail:
(endless?) fight against state space explosion

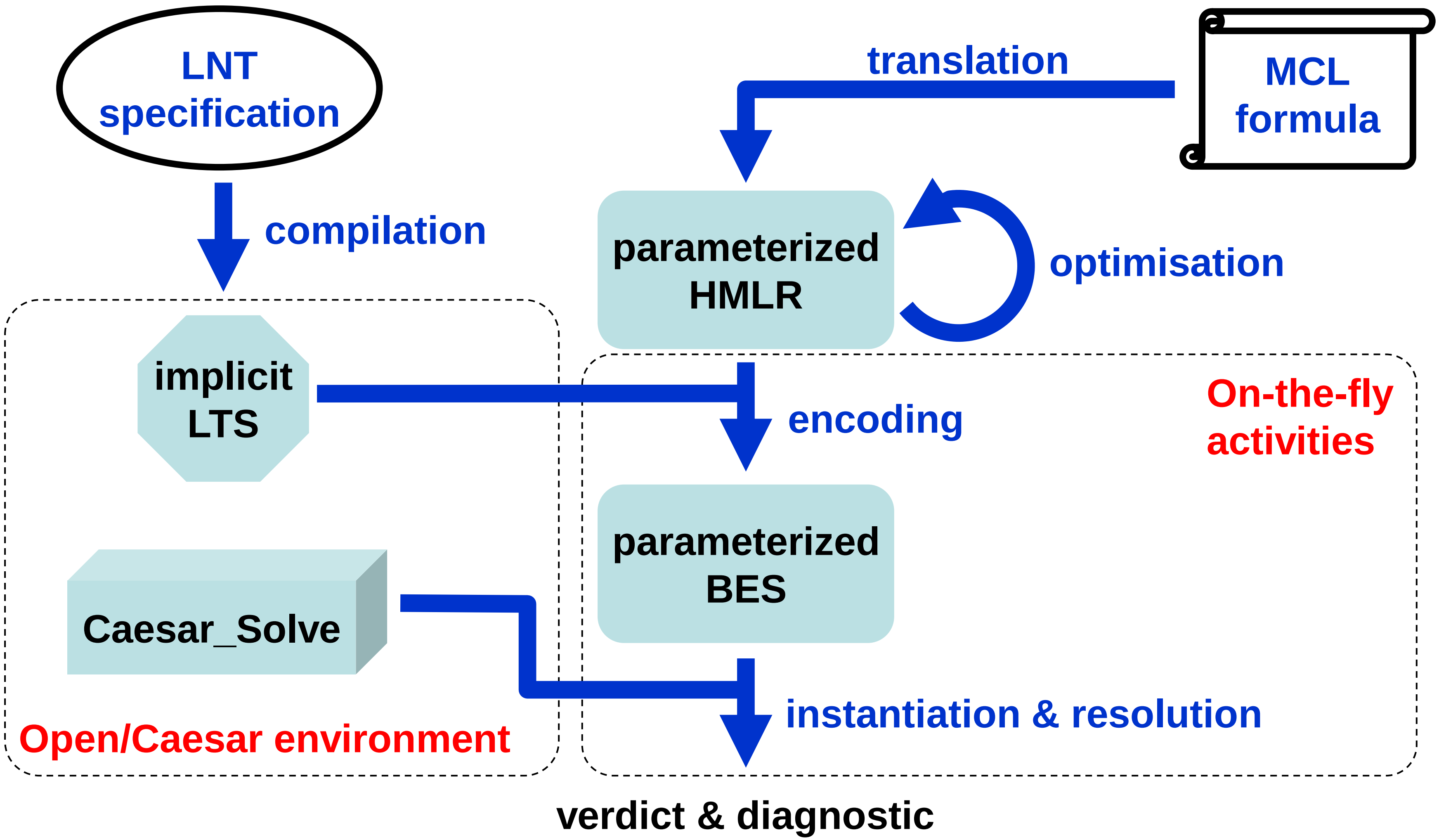
On-the-fly model checking

(linear-time, state-based – LTL/SPIN)



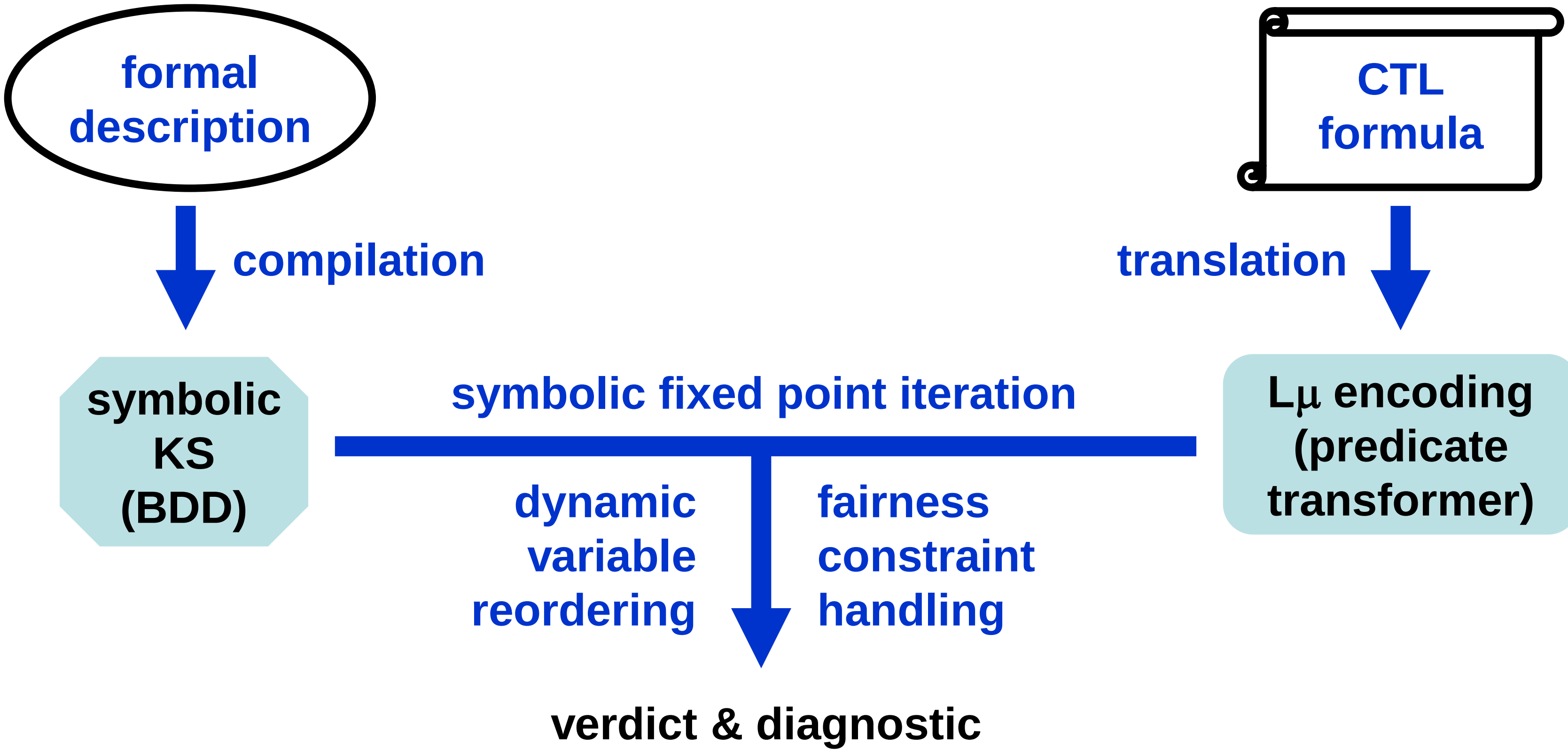
On-the-fly model checking

(branching-time, action-based – MCL/CADP/Evaluator)



Symbolic model checking

(branching-time, state-based logics – CTL/nuSMV)



Other ways to fight state explosion

- Bounded model checking
  - Symbolic partial exploration, use of SAT/SMT solvers
- Parallel and distributed model checking
  - Explicit / symbolic, linear / branching
- Compositional verification
  - Assume-guarantee / partial model checking
- Runtime verification
  - TL formulas → monitors → check execution traces
- Statistical model checking

Model checkers landscape

(partial view)

- SPIN (explicit/parallel)
- SPOT (explicit/symbolic)
- DIVINE (explicit/distributed)
- LTSmin (explicit/distributed)

LTL

- UPPAAL (symbolic) — **Timed CTL**
- TINA (symbolic) — **Timed LTL**
- nuSMV (symbolic) — **CTL^F**
- TLA+ (symbolic) — **TLA**

- PRISM (explicit/symbolic)
- MRMC (explicit/symbolic)
- MODEST (explicit/symbolic)

PCTL
CSL

- LTSA (explicit) — **F-LTL**
- CADP (explicit/distributed) — **MCL**
- JACK (explicit/symbolic) — **μ-ACTL**

Model checking in the design process

- Choose the right modeling language and TL
- Model the *essential* aspects of the system
- Start with on-the-fly (parallel) verification:
  - Fast detection of errors
  - Debug based on counterexamples
- When no more errors found / no memory left:
  - Use symbolic / compositional / distributed verification
  - Use abstraction whenever possible

What to do next?

- Regular increase of model checking capabilities
  - Bounded model checking, SAT/SMT techniques
- Several stable tools (and many others!)
  - Industrial success stories for each method / tool
- Model checking interoperates with other techniques (static analysis, theorem proving, …)
- Ideally, one should be able to apply smoothly several verification techniques on the same system description

  → need for languages / models / tools interoperability

Some references

- [Schnoebelen-et-al-99] *Vérification de logiciels*
- [Clarke-Grumberg-Peled-00] *Model Checking*
- [Baier-Katoen-08] *Principles of Model Checking*

+ many articles on the various model checkers

Thank you