Symbolic Computation
Algebraic Biology I

Bud Mishra

Courant Inst, NYU

NYU SoM, TIFR, MSSM

Systems Biology

- Introduction to Biology
- Regulatory & Metabolic Processes
- Algebraic Models in Biology

Symbolic Computation
Algebraic Biology II

Bud Mishra

Courant Inst, NYU

NYU SoM, TIFR, MSSM

Model Checking

- Temporal Logic
- Kripke Models
- Model Checking
- Biologically Faithful Models

A Simple Model of Biochemical Processes

- Such a model can be obtained through spatio-temporal discretization.
- It can be further compressed through a "bisimulation-equivalence" relation.
- Technicalities to be elaborated further.

XSSys is a modular system

Canonical Form:

$$\begin{cases} \dot{X}_i = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}} \quad i = 1..n \\ C_l(X_1(t),\ldots,X_{n+m}(t)) = \sum(\gamma_l \prod_{j=1}^{n+m} X_j^{f_{lj}}) = 0 \end{cases}$$

Characteristics:

- **Predefined Modular Structure**
- **Automated Translation from Graphical to Mathematical Model**

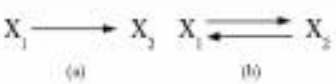


Figure 1: Representation of an unmodified and of a reversible reaction.

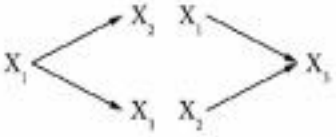


Figure 2: Representation of a divergence and of a convergence branch point (the two processes in each reaction are independent of each other).

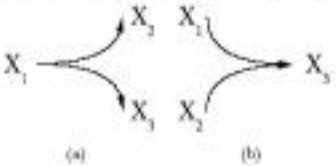


Figure 3: Representation of a single splitting reaction generating two products, X_2 and X_3, in stoichiometric proportions and of a single synthetic reaction involving two source components, X_1 and X_2 always in stoichiometric proportions.

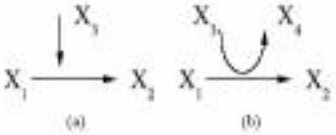
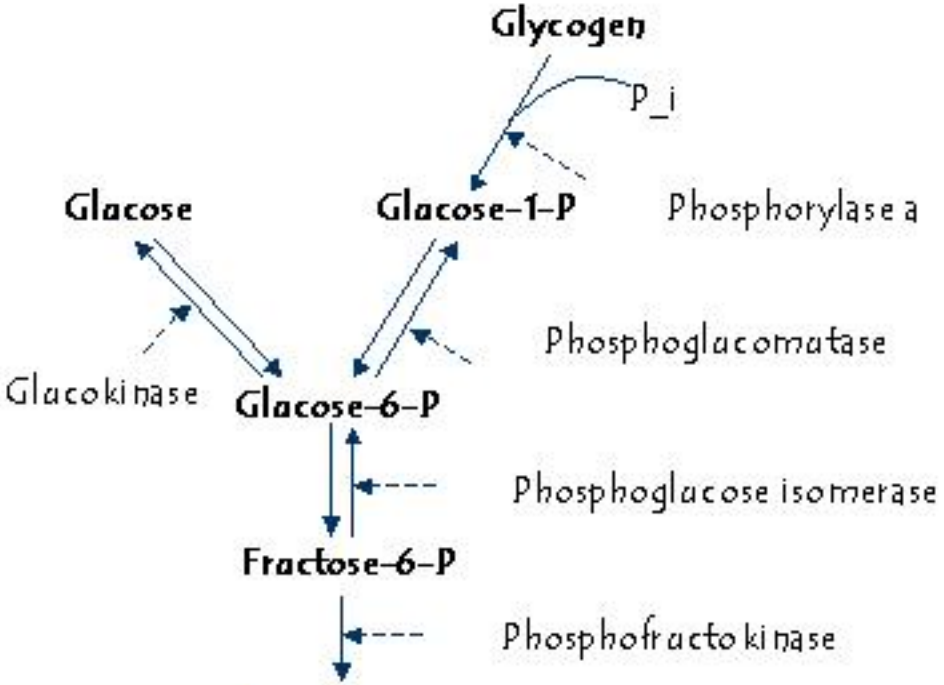


Figure 4: The conversion of X_1 into X_2 is modulated (stimulation or inhibition is represented by the sign of the arrow) by X_3. The reaction between X_1 and X_2 requires coenzyme X_3, which in the process is converted into X_4.

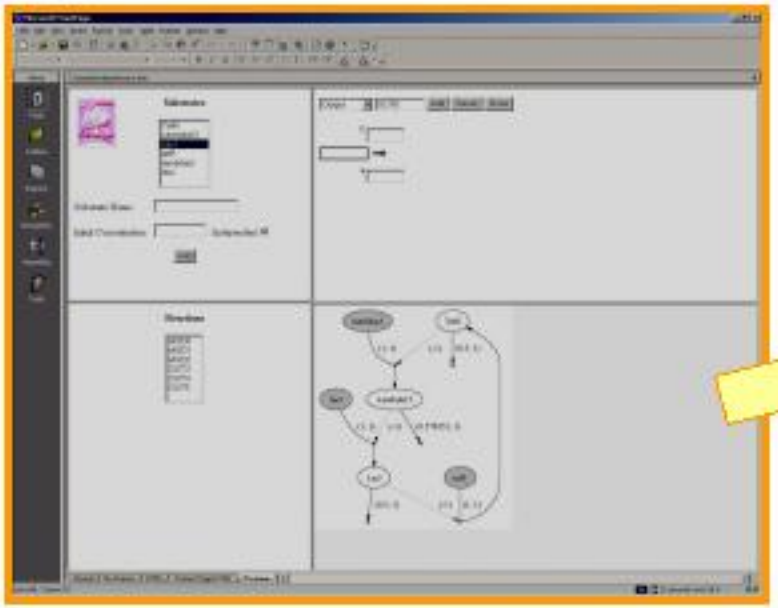
Example

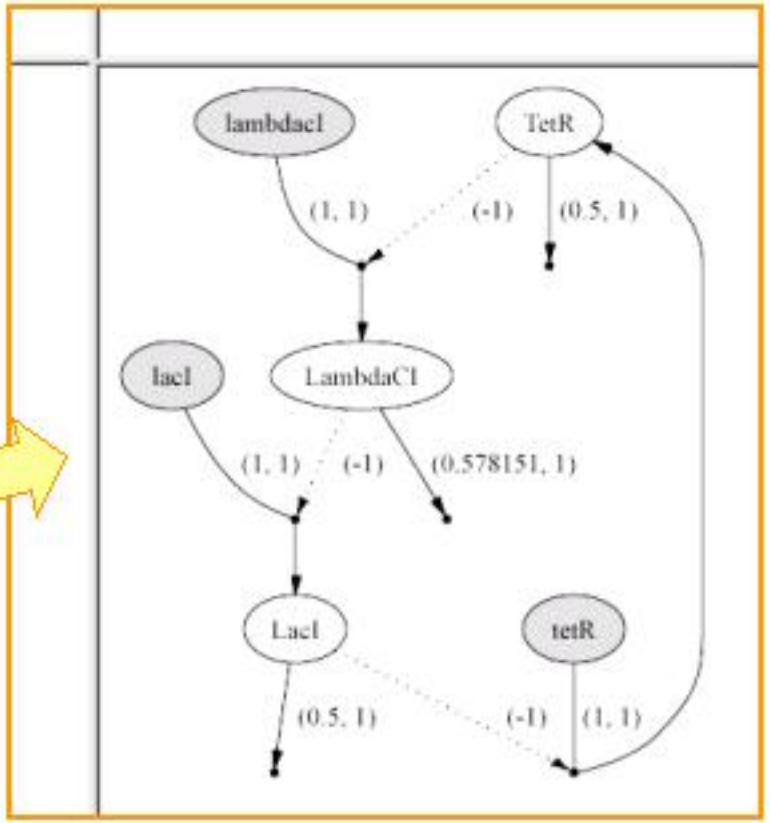


Glycolysis

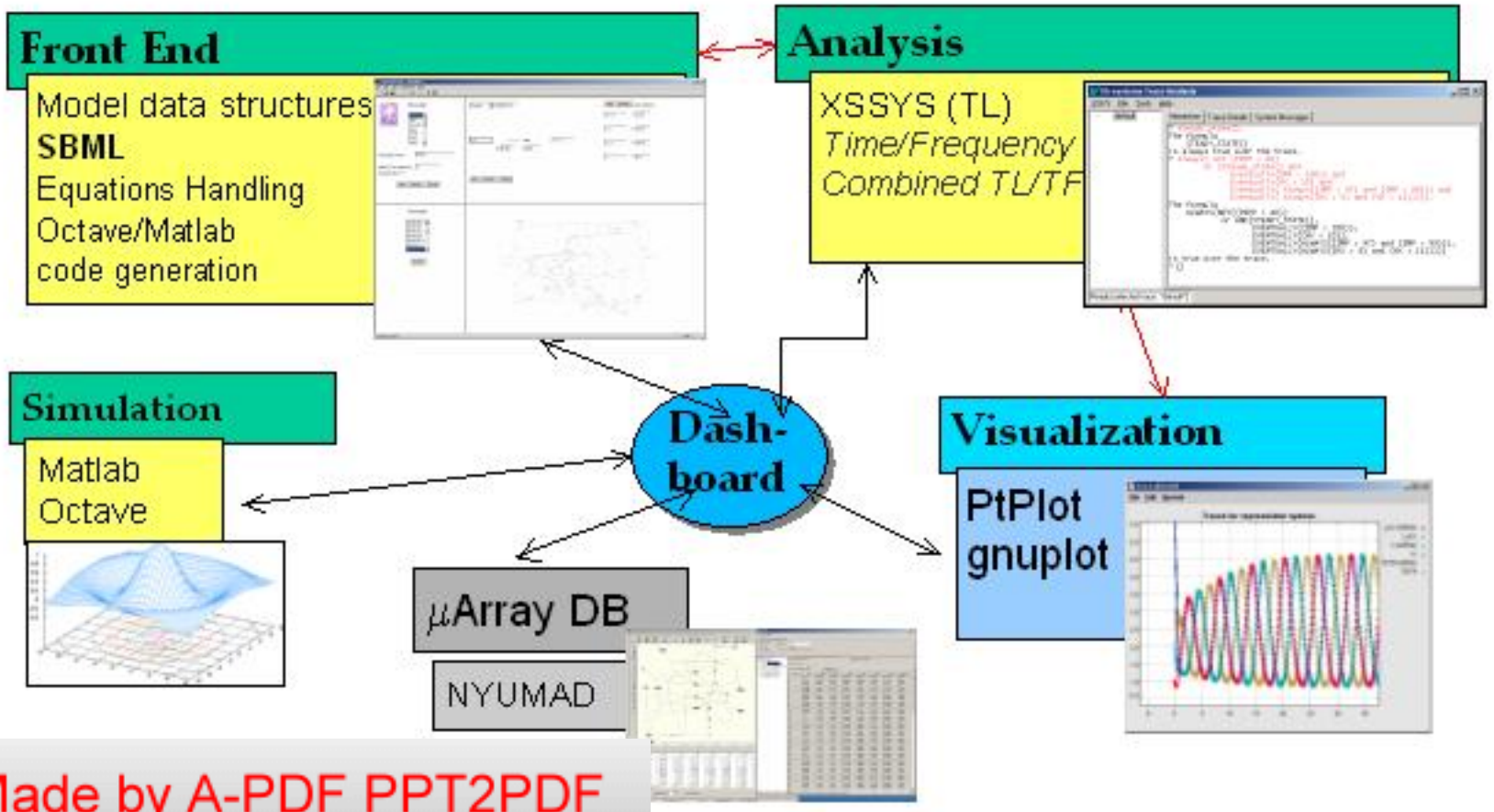
SIMPATHICA

SimPathica System





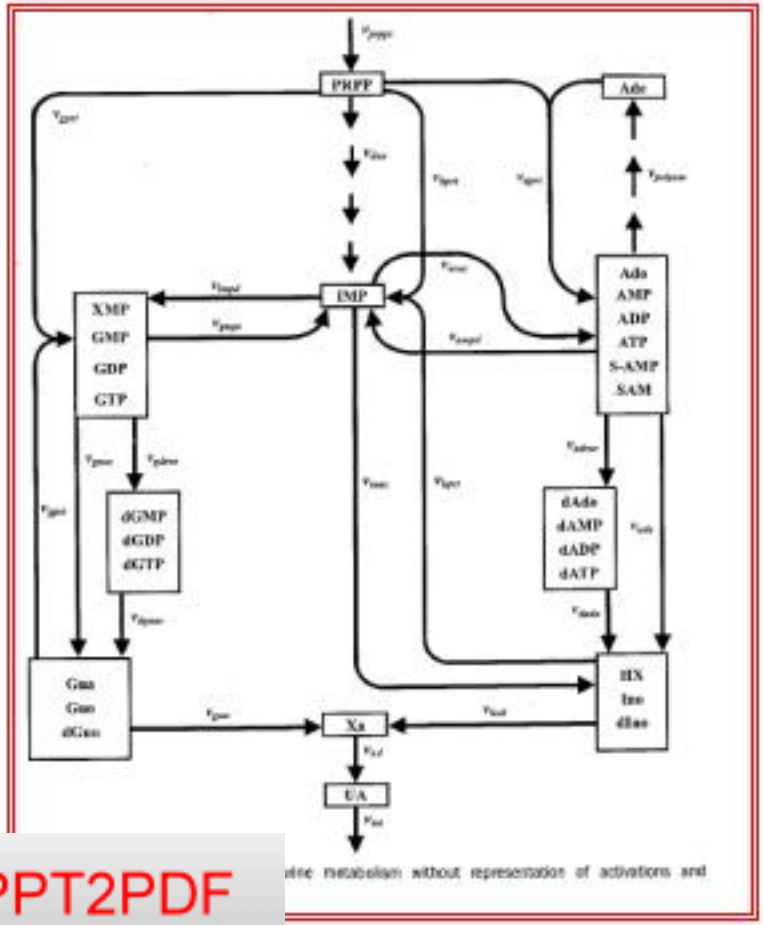
Simpathica is a multi-functional system



Purine Metabolism

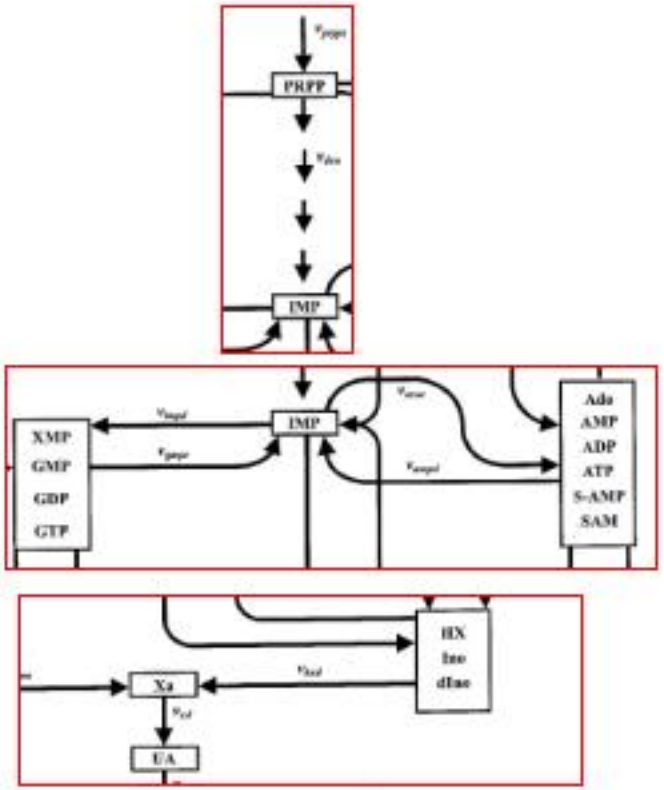
- **Purine Metabolism**
  - Provides the organism with building blocks for the synthesis of DNA and RNA.
  - The consequences of a malfunctioning purine metabolism pathway are severe and can lead to death.
- **The entire pathway is almost closed but also quite complex. It contains**
  - several feedback loops,
  - cross-activations and
  - reversible reactions
- **Thus is an ideal candidate for reasoning with computational tools.**

Simple Model



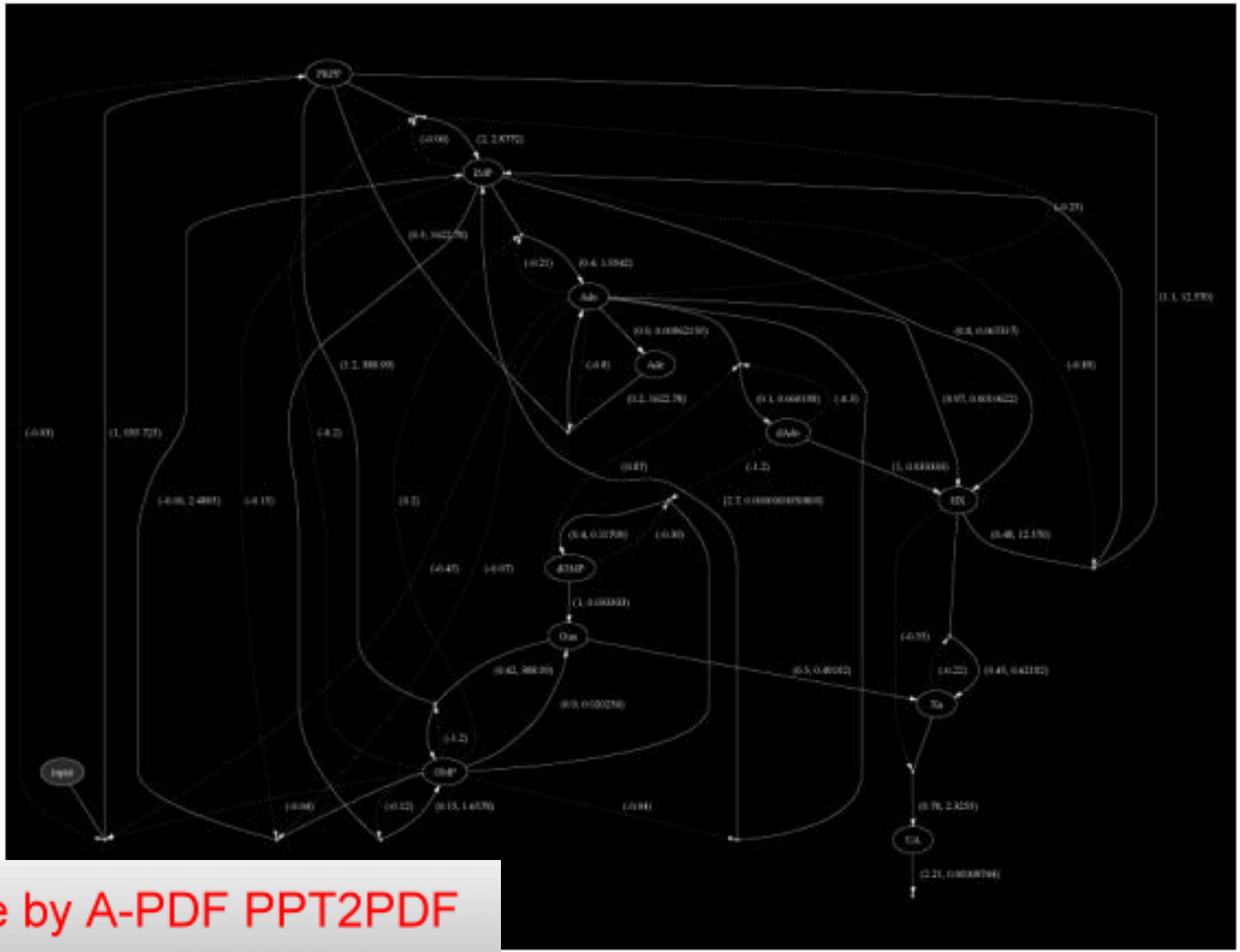
...ine metabolism without representation of activations and

Biochemistry of Purine Metabolism



- The main metabolite in purine biosynthesis is *5-phosphoribosyl-a-1-pyrophosphate* (*PRPP*).
  - A linear cascade of reactions converts PRPP into *inosine monophosphate* (*IMP*). IMP is the central branch point of the purine metabolism pathway.
  - IMP is transformed into AMP and GMP.
  - Guanosine, adenosine and their derivatives are recycled (unless used elsewhere) into *hypoxanthine* (*HX*) and *xanthine* (*XA*).
  - XA is finally oxidized into *uric acid* (*UA*).

Purine Metabolism



Queries

- Variation of the initial concentration of PRPP does not change the steady state. **(PRPP = 10 * PRPP1) implies steady_state()**
- This query will be true when evaluated against the modified simulation run (i.e. the one where the initial concentration of PRPP is 10 times the initial concentration in the first run – PRPP1).

TRUE

- Persistent increase in the initial concentration of PRPP does cause unwanted changes in the steady state values of some metabolites.
- If the increase in the level of PRPP is in the order of 70% then the system does reach a steady state, and we expect to see increases in the levels of IMP and of the hypoxanthine pool in a "comparable" order of magnitude.

Always (PRPP = 1.7*PRPP1) implies steady_state()

TRUE

Queries

- Consider the following statement:
- **Eventually**

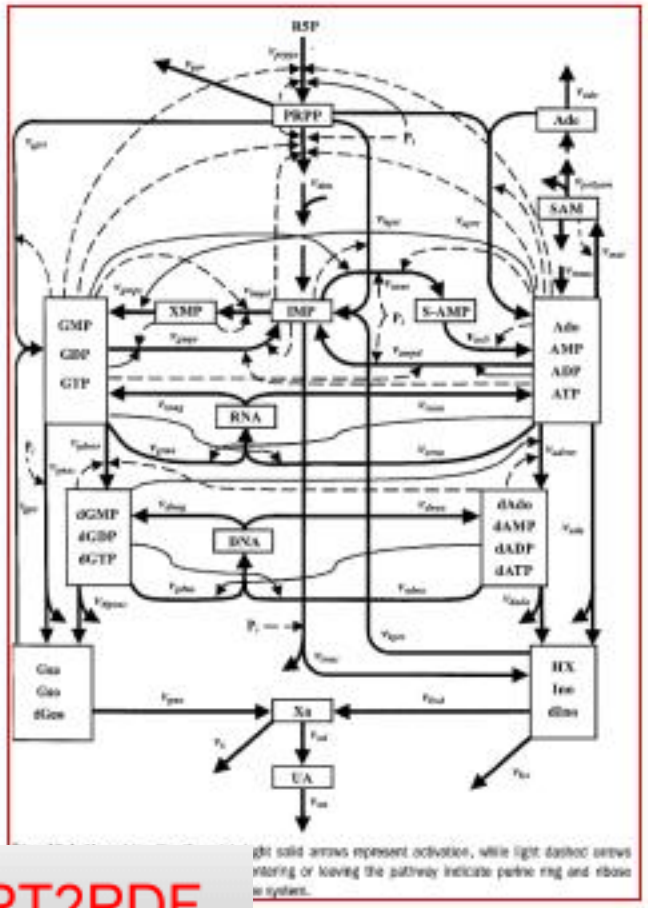
(Always (PRPP = 1.7 * PRPP1)
implies
steady_state()
and Eventually
Always(IMP < 2* IMP1))
and Eventually (Always
(hx_pool < 10*hx_pool1)))

- where IMP1 and hx_pool1 are the values observed in the unmodified trace. The above statement turns out to be false over the modified experiment trace..

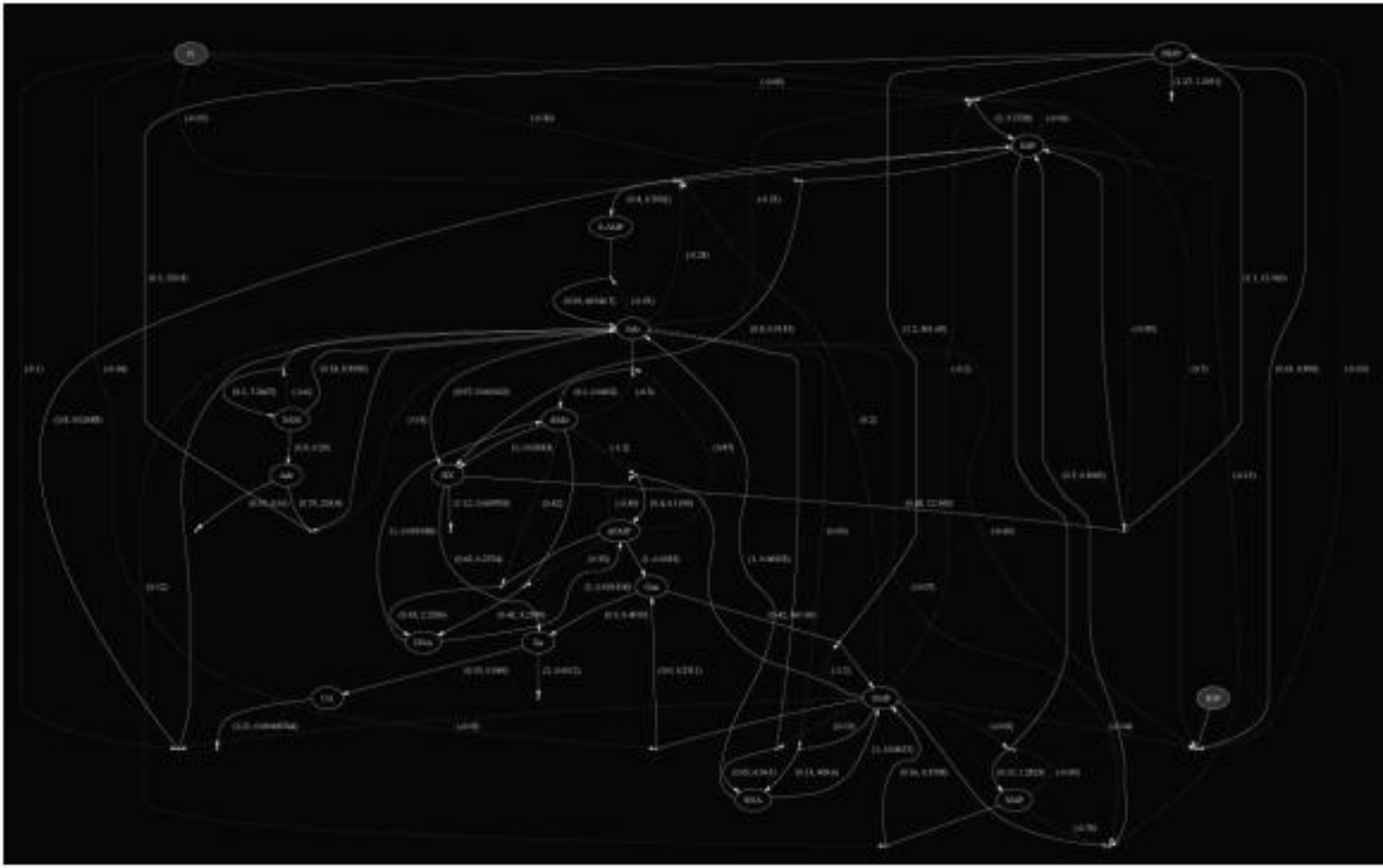
- In fact, the increase in IMP is about 6.5 fold while the hypoxanthine pool increase is about 60 fold.
- Since the above queries turn out to be false over the modified trace, we conclude that the model "over-predicts" the increases in some of its products and that it should therefore be amended

False

Final Model



Purine Metabolism



Framework & Outline

- Language:
  - Ontology
  - Controlled Vocabulary
- Model Building
  - Kripke Structure
  - Model Building; Hidden Kripke Models (HKM)
  - Information Bottleneck
- Invariants & Redescriptions
  - Labeling with Propositions
  - Statistical Significance
- Logic & Models
  - Temporal Logic
  - Model Checking
- Examples
  - Yeast Cell Cycle
  - Host-Pathogen Interaction
  - Life Cycle of a Parasite
  - Cancer Initiation and Progression
- Implementation

Model Checking & Kripke Structure

- Definition: **Kripke Structure**
  - ...captures the intuition about behavior of a reactive system...
  - ...consists of a set of states, a set of transitions between states, and ***a function that labels each state with a state of properties—true in that state.***

Formal Definition

- Let AP be a set of atomic propositions. A **Kripke Structure** M over AP is a four tuple M = (S, S_0, R, L) where
  - S is a finite set of states.
  - $S_0 \subseteq S$ is a set of initial states.
  - $R \subseteq S \times S$ is a transition relation that must be total, that is, for every state $s \in S$ there is a state $s_0 \in S$ such that $R(s, s_0)$.
  - $L : S \rightarrow 2^{AP}$ is a function that labels each state with the state of atomic propositions that are true in that state.

First Order Representation

- **Logical Connectives**: and $\wedge$, or $\vee$ not $\neg$, implies $\Rightarrow$, so on.
- **Quantifiers**: universal quantifiers $\forall$, existential quantifiers $\exists$, so on.
- A logic is **propositional**, if it consists of atomic propositions and formulas created with the logical connectives... but no quantifiers.
- A logic is **first order** if the atomic propositions take values in a domain (not necessarily finite) and in addition the formulas are quantified over the domain.

Example

$$\forall \text{ initial cell mass}_{c, k_1 < c < k_2}$$
$$[\text{CellIn}(S, 0) \Rightarrow \forall_{t>0} \text{CellIn}(S, t)]$$
$$\wedge \forall \text{ initial cell mass}_{c, c > k_2}$$
$$[\text{CellIn}(S, 0) \Rightarrow \exists_{t>0} \text{CellIn}(M, t)]$$

Temporal Logic

- **Temporal Logic** is a formalism for describing a sequence of transitions between states in a **reactive system**...
- In this logic, time is never mentioned explicitly with a metric... But only in a "topological" manner...

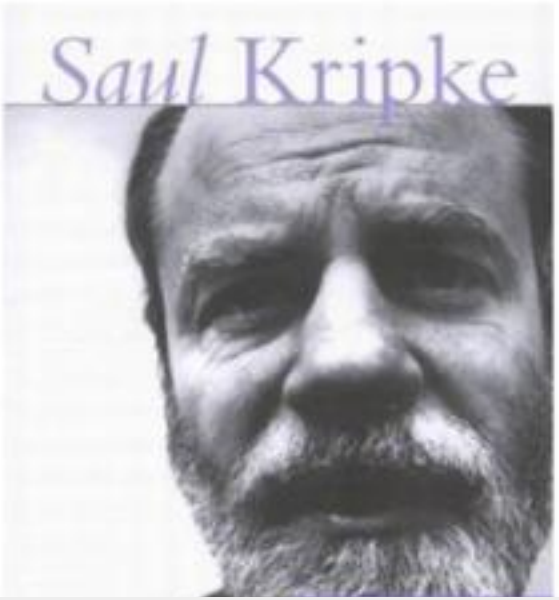
Modes

- "Eventually something happens" ...
  "Always something is true" ...
  "Something is never true" ...
  "Something else holds almost always" ...
  "This is true infinitely often" ...
- Main modes or temporal operators are **X**, **F**, **G**, **U** and **R**.
- Main path quantifiers are **A** and **E**.

Computation Tree Logic CTL

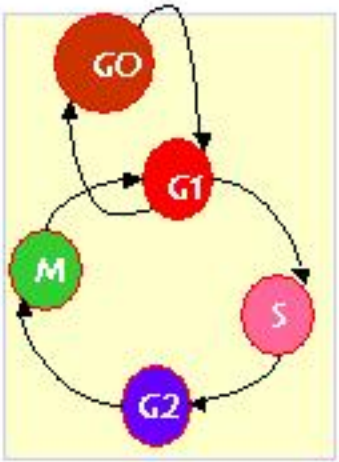
- CTL formulas describe properties of computation trees. ***The tree is formed by designating a state in a Kripke structure as the initial state and then unwinding the structure into an infinite tree with the designated state at the root***...
- The tree shows all possible execution paths in the tree...
- CTL formulas are composed of path quantifiers and temporal operators.

Kripke Structure



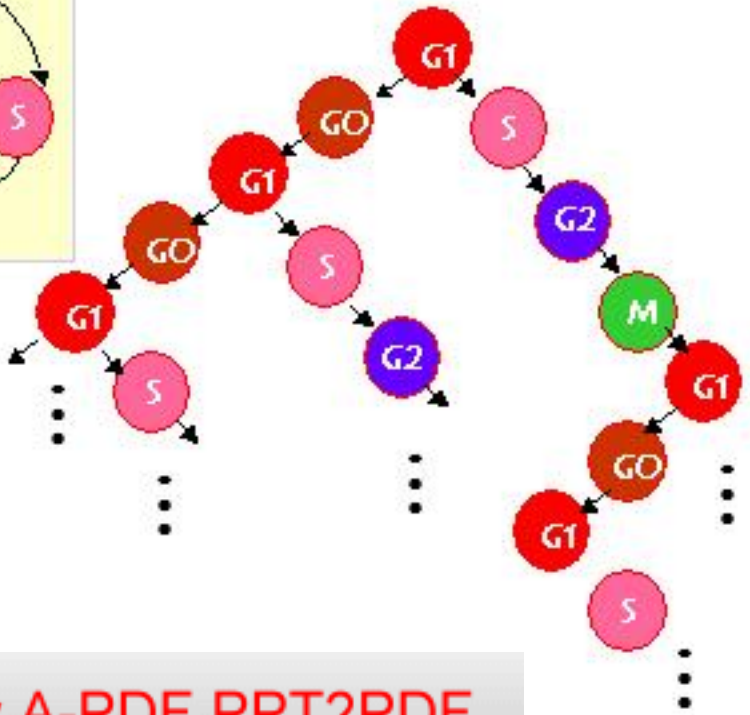
- Thus KS provides formal encoding of a Biological Dynamical System:
- Simple and intuitive pictorial representation of the behavior of a complex system
  - A **Graph** with **nodes** representing **system states** labeled with information true at that state
  - The **edges** represent **system transitions** as the result of some action

Computation Tree



[a]

[b]



- Finite set of states; Some are initial states
- **Total** transition relation: every state has at least one next state i.e. infinite paths
- There is a set of basic environmental variables or features ("atomic propositions")
- In each state, some atomic propositions are true

Computation Tree Logic CTL

- **Next Time:** X P... property P holds in the second state of the path.
- **Eventually:** F P ... property P will hold at some state on the path (in the future)
- **Always:** G P ... property P holds at every state on the path (globally)
- **Until:** P U Q ... property Q holds at some state on the path and property P holds at all preceding states
- **Release:** P R Q ... property Q holds along the path up to and including the first state at which P holds (if it does at all)

Computation Tree Logic CTL

- There are two types of formulas in CTL: **state formulas** and **path formulas**...
  - ... state formulas are true in a specific state...
  - ...path formulas are true along a specific path...

Syntax

- Let AP be the set of atomic proposition names.
- The syntax of state formulas:
  - If $p \in AP$, then p is a state formula.
  - If f and g are state formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are state formulas.
  - If f is a path formula then E f and A f are state formulas.

The syntax of path formulas:

- If f is a state formula, then f is also a path formula.
- If f and g are path formulas, then $\neg$ f, f $\vee$ g, f $\wedge$ g, X f, F f, G f, f U g and f R g are path formulas.

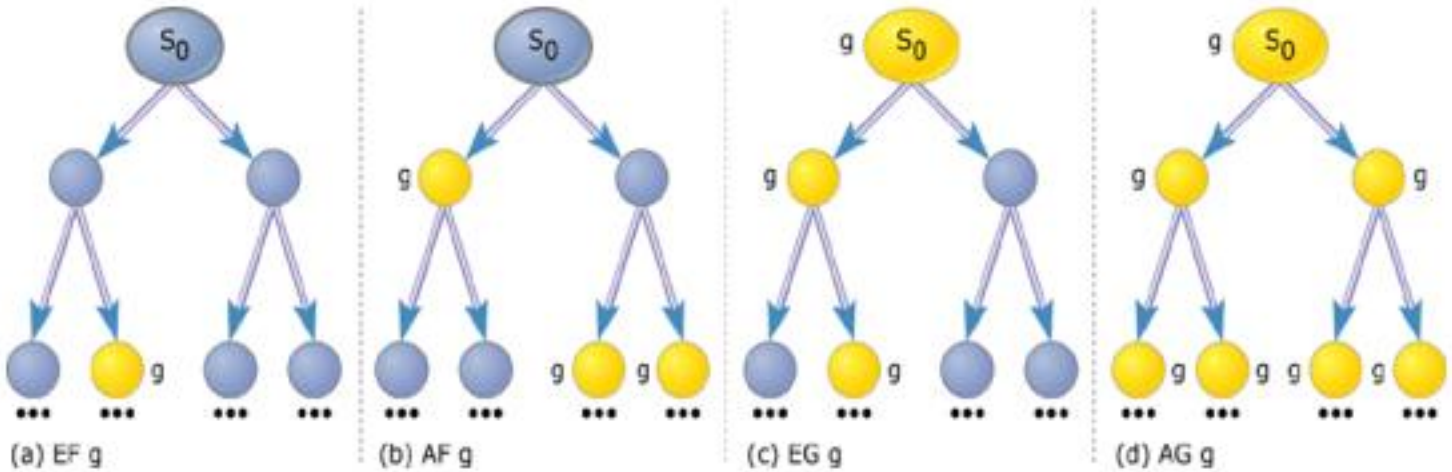
Semantics of CTL with Kripke Structure

- $M = \langle S,R,L \rangle$ = a Kripke Structure. S = the set of states, $R \subseteq S \times S$ = transition relation (total) and $L: S \rightarrow 2^{AP}$ = the labeling function... labels each state with a set of atomic propositions true in that state
- A path in M is an infinite sequence of states $\pi = s_0, s_1, \ldots$ such that $\forall_{I \geq 0} (s_i, s_{i+1}) \in R$

Semantics for CTL

- For $p \in AP$:
  $s \models p \Leftrightarrow p \in L(s)$ $\quad s \models \neg p \Leftrightarrow p \notin L(s)$
- $s \models f \wedge g \Leftrightarrow s \models f$ and $s \models g$
- $s \models f \vee g \Leftrightarrow s \models f$ or $s \models g$
- $s \models EX\, f \Leftrightarrow \exists\, \pi = \langle s_0 s_1 \ldots \rangle$ from s $s_1 \models f$
- $s \models E(f\, U\, g) \Leftrightarrow \exists\, \pi = \langle s_0 s_1 \ldots \rangle$ from s
  $\exists j \geq 0 \,[\, s_j \models g$ and $\forall i : 0 \leq i < j \,[s_i \models f\,]\,]$
- $s \models EG\, f \Leftrightarrow \exists\, \pi = \langle s_0 s_1 \ldots \rangle$ from s $\forall i \geq 0$: $s_i \models f$

Some CTL Operators



EF g AF g EG g AG g

Least Fixed Point Characterization

◊ It suffices to define all path formulas in terms of: P, $\neg f$, $f_1 \wedge f_2$, $\mathcal{A}\mathcal{X}f_1$, $\mathcal{E}\mathcal{X}f_1$, $\mathcal{A}[f_1 \mathcal{U} f_2]$ and $\mathcal{E}[f_1 \mathcal{U} f_2]$

◊ $P \equiv \mu z.P$

◊ $\neg f_1 \equiv \mu z.\neg f_1$

◊ $f_1 \wedge f_2 \equiv \mu z.f_1 \wedge f_2$

◊ $\mathcal{A}\mathcal{X}f_1 \equiv \mu z.\mathcal{A}\mathcal{X}f_1$

◊ $\mathcal{E}\mathcal{X}f_1 \equiv \mu z.\mathcal{E}\mathcal{X}f_1$

◊ $\mathcal{A}[f_1 \mathcal{U} f_2] \equiv \mu z.f_2 \vee (f_1 \wedge \mathcal{A}\mathcal{X}z)$

◊ $\mathcal{E}[f_1 \mathcal{U} f_2] \equiv \mu z.f_2 \vee (f_1 \wedge \mathcal{E}\mathcal{X}z)$

Algorithm

Label-Graph(f, M)
begin case:

- $f = P$:
  while $\exists s \in S$ s.t. $[f \notin Lbl(s)$ and $P \in L(s)]$
  Add f to $Lbl(s)$

- $f = \neg f_1$:
  Label-Graph(f_1, M);
  while $\exists s \in S$ s.t. $[f \notin Lbl(s)$ and $f_1 \notin Lbl(s)]$
  Add f to $Lbl(s)$

- $f = f_1 \wedge f_2$:
  Label-Graph(f_1, M);
  Label-Graph(f_2, M);
  while $\exists s \in S$ s.t. $[f \notin Lbl(s)$ and $f_1 \in Lbl(s)$ and $f_2 \in Lbl(s)]$
  Add f to $Lbl(s)$

- $f = AXf_1$:

while $\exists s \in S$ s.t. $[f \notin Lbl(s)$ and $\forall t \in succ(s), f_1 \in Lbl(t)]$
Add f to $Lbl(s)$

◊ $f = \mathcal{EX} f_1$:
Label-Graph(f_1, M);
while $\exists s \in S$ s.t. $[f \notin Lbl(s)$ and $\exists t \in succ(s), f_1 \in Lbl(t)]$
Add f to $Lbl(s)$

◊ $f = \mathcal{A}[f_1 \mathcal{U} f_2]$:
Label-Graph(f_1, M);
Label-Graph(f_2, M);
while $\exists s \in S$ s.t. $[f \notin Lbl(s)$ and $(f_2 \in Lbl(s)$
or $(f_1 \in Lbl(s)$ and $\forall t \in succ(s), f \in Lbl(t)))]$
Add f to $Lbl(s)$

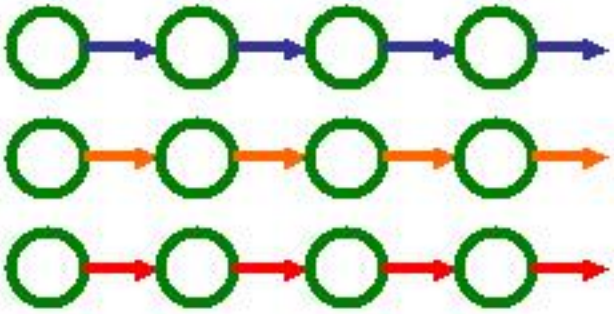
◊ $f = \mathcal{A}[f_1 \mathcal{U} f_2]$:
Label-Graph(f_1, M);
Label-Graph(f_2, M);
while $\exists s \in S$ s.t. $[f \notin Lbl(s)$ and $(f_2 \in Lbl(s)$
or $(f_1 \in Lbl(s)$ and $\exists t \in succ(s), f \in Lbl(t)))]$
Add f to $Lbl(s)$

In Summary

- **First Order Logic:** Time is an explicitly quantified variable
- **Propositional Modal logic:** was invented to formalize modal notions and suppress the quantified variables – with operators "possibly P" and "necessarily P" (similar to "eventually" and "henceforth")
- **Temporal Logic:** Short hand for describing the way properties of the system change with time; Time is implicit.

Branching versus Linear Time

- **Linear-time:** Only one possible future in a moment
  - Look at individual computations
- **Branching-time:** It may be possible to split to different courses depending on possible futures
  - Look at the tree of computations

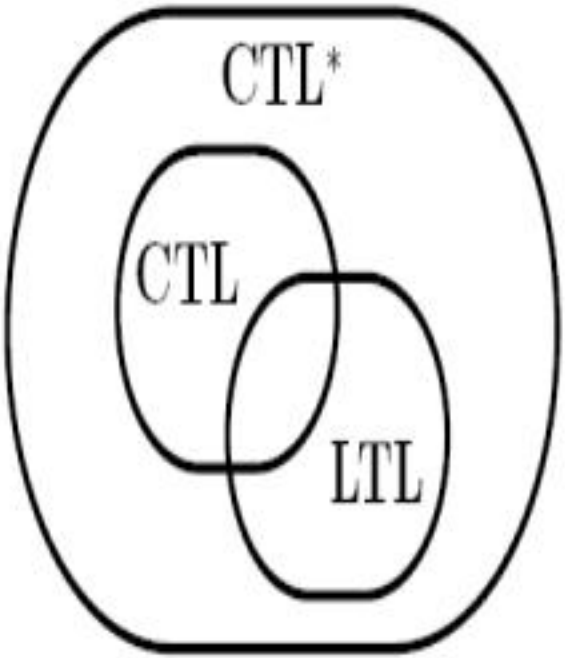


L

Time is Branching: CTL

CTL*

- A path quantifier can be followed by an arbitrary number of temporal operators
- There are properties expressible in CTL but not in LTL and vice-versa
- LTL, CTL are contained in CTL*



Complexity Comparison

Size of transition system: **n**

Size of temporal logic formula: **m**

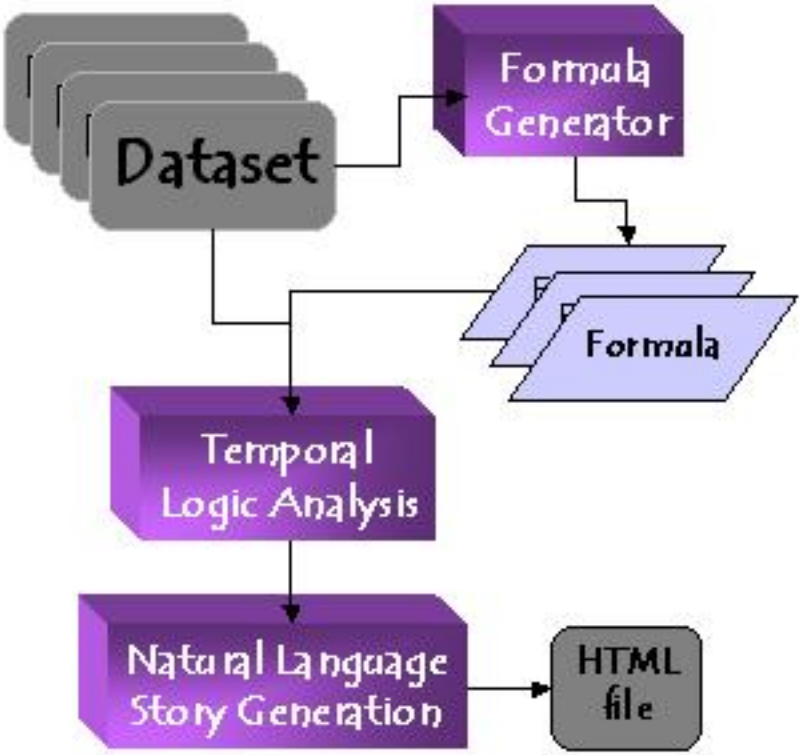
- Worst Case Comparison:
  - CTL: linear - O(nm)
  - LTL: exponential – n $2^{O(m)}$
- For an LTL formula that can also be expressed in vCTL, **LTL model-checking can be done in a time linear in the size of the formula**
- LTL is PSPACE complete: Hamiltonian Path problem can be reduced to an LTL Model Checking problem:

$$\mathbf{F}p_1 \wedge \mathbf{F}p_2 \wedge \mathbf{F}p_3 \wedge ..$$
$$\mathbf{G}\,(p_1 \rightarrow \mathbf{XG}\,\neg p_1) \wedge \mathbf{G}(p_2 \rightarrow \mathbf{XG}\,\neg p_2) \wedge \ldots.$$

Other Model Checking Algorithms

- LTL Model Checking: Tableu-based…
- CTL* Model Checking: Combine CTL and LTL Model Checkers…
- Symbolic Model Checking
  - Binary Decision Diagram
  - OBDD-based model-checking for CTL
  - Fixed-point Representation
  - Automata-based LTL Model-Checking
- SAT-based Model Checking
- Algorithmic Algebraic Model Checking
- Hierarchical Model Checking

Story generation



- Temporal Logic formulae can be rendered in English.
- Temporal Logic formulae can be generated automatically (with care).
- Each formula can be tested against a set of datasets; differences can then be noted.

Cell Cycle Story Generation Results

Report on "Test Experiment: WT, 1 Mutant, 2 Mutants."

The results refer to the following datasets:

- **The first dataset is named "Experiment/Yeast Dataset WT".**
- **The second dataset is named "Experiment/Yeast Dataset Mut1".**
- **The third dataset is named "Experiment/Yeast Dataset mut2".**

...

84. CDH1 less than or equal to 1.0071783 will always hold until CDH1 activates CYCB, is true in the first dataset, is true in the second dataset, and is false in the third dataset.
85. CDH1 represses CYCB implies CYCB is greater than or equal to 0.65, is false in the first dataset, is true in the second dataset, and is true in the third dataset.
86. eventually, CDH1 is less than or equal to CYCB, is false in the first dataset, is true in the second dataset, and is true in the third dataset.

...

Circadian Clock

Temporal Logic cannot reason about cases where time must be "metric."

Circadian Oscillations

- "A model for circadian oscillations in the Drosophila period protein (PER)," Albert Goldbeter,
  - *Proc. R. Soc. Lond. B* (1995), **261**:319-324.
- A theoretical model:
  - Takes into account contemporary experimental observations
  - Model for circadian clock is based on
    1. multiple phosphorylation of PER protein
    2. the negative feedback exerted by PER on the transcription of the period (*per*) gene.

Model

- This minimal biochemical model provides a molecular basis for circadian oscillation of the limit cycle type.
- During oscillations, the peak in per mRNA precedes by several hours the peak in total PER protein.
  - Accepted view: Multiple PER phosphorylation induces time delays which strengthen the capability of negative feedback to produce oscillation.

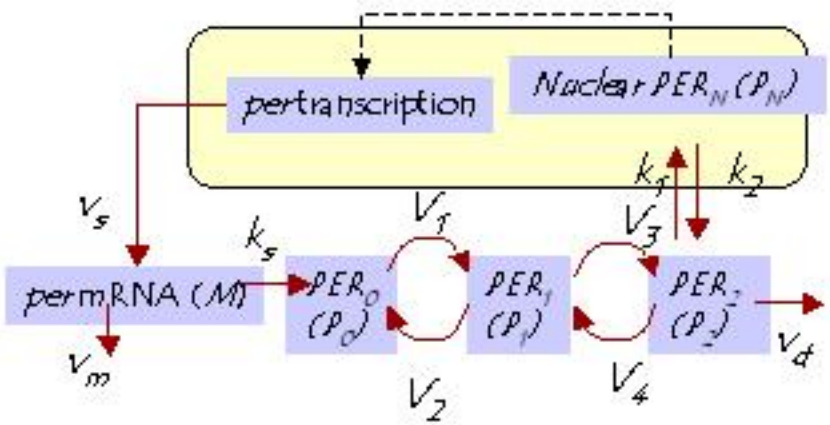
Many Unresolved Issues:

- Isomorphism to **Van der Pol** system,
- Robustness.

Two Competing Biological Models

- Edery et al. (1994) Model:
  - Based upon multiple phosphorylation of PER and on repression of *per* transcription by a phosphorylated form of the PER protein.
- Abbott et al. (1995) Model:
  - Based upon the effect of a larger number of phosphorylated residues and their effect upon delaying the entry of the protein into nucleus and the resulting negative feedback effect on *per* transcription.

Circadian Oscillation of PER and *per* mRNA: Assumptions



1. *per* mRNA is synthesized in the nucleus and transferred ti cytosol, where it is degraded.
   M = Cytosolic concentration of *per*
2. Rate of synthesis of PER (by translation of *per* mRNA) is proportional to M.
3. PER is multiply phosphorylated:
   $P_0 \mapsto P_1 \mapsto P_2$
4. Phosphorylated PER is transported into the nucleus: P_N
5. P_N acts directly as a repressor and reduces the *per* transcription rate.

Phosphorylation of PER

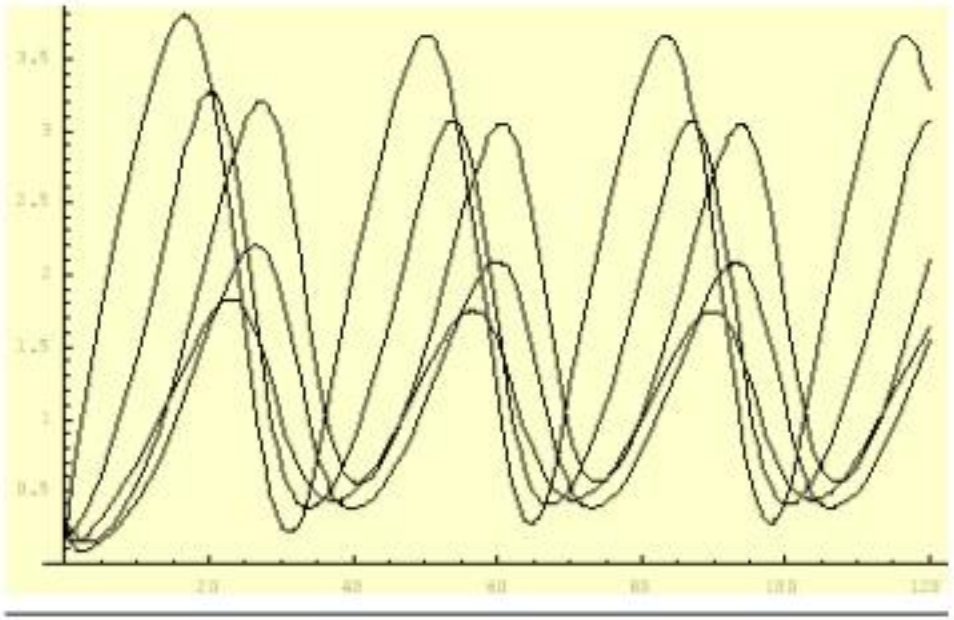
- PER is multiply phosphorylated:
  - To keep the model simple, only three states of the PER protein is considered:

    P_0 = Unphosphorylated, P_1 = Monophosphorylated and P_2 = Biphosphorylated
  - The precise number of phosphorylated residues is still unknown. The role of PER phosphorylation is still unclear.
- Phosphorylation may control nuclear localization and/or degradation of PER.
  - Assume that the fully phosphorylated form P_2 is marked both for degradation and reversible transport into the nucleus.
- The effect of the nuclear form of PER (P_N) on the per transcription (M) is described by an equation of Hill type with a Hill (cooperativity) coefficient of n = 4.

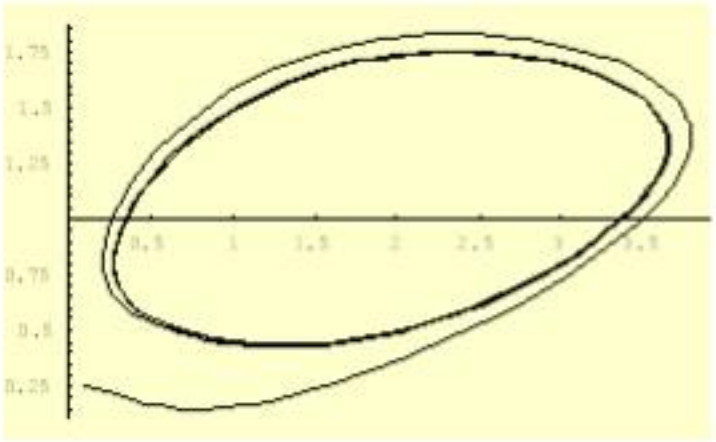
Differential Equations

- $dM/dt = v_s K_1^n/(K_1^n + P_N^n) - v_m M/(K_m+M)$
- $dP_0/dt = k_s M - V_1 P_0/(K_1+P_0) + V_2 P_1/(K_2 + P_1)$
- $dP_1/dt = V_1 P_0/(K_1+P_0) - V_2 P_1/(K_2+P_1) - V_3 P_1/(K_3+P_1) + V_4 P_2/(K_4+P_2)$
- $dP_2/dt = V_3 P_1/(K_3+P_1) - V_4 P_2/(K_4+P_2) - k_1 P_2 + K_2 P_N - v_d P_2/(k_d+P_2)$
- $dP_N/dt = k_1 P_2 - k_2 P_N$
- $P_t = P_0 + P_1 + P_2 + P_N$

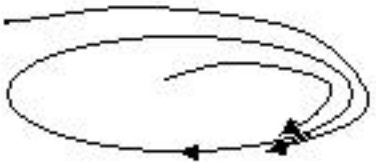
Simulation



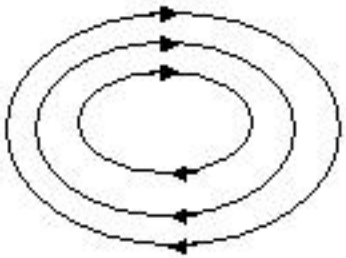
Phase Plane



Periodic Orbits and Limit Cycles



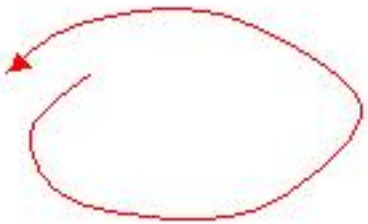
Stable Limit Cycle



Not a Limit Cycle

- (Stable) Limit Cycle $\triangleq$ A periodic trajectory hich attracts other solutions to it.
- A member of a family of "parallel" periodic solutions (with linear centers) is not a limit cycle.
- Limit cycles are robust in two ways:

Robustness of Limit Cycles



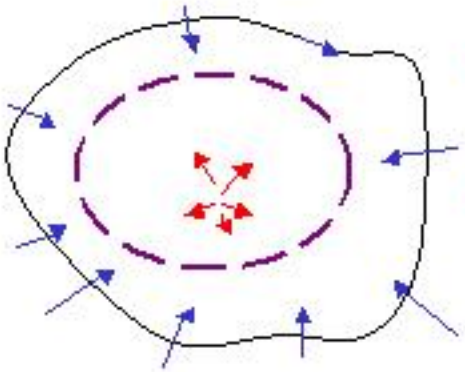
- If perturbation moves state to different initial state away from the cycle, then the system will return to cycle…
  - e.g. Circadian rhythm: Phase adjusts after jet lag…
  - For a linear oscillator, this is not true; it will simply start oscillating along a different orbit and will never return to the original orbit.
  - If dynamics changes a little a limit cycle will still exist (can be proved using Poincare-Bendixon theorem.)
- Think of a linear oscilator:

$$dx/dt = y,\ dy/dt = -x + \varepsilon\, y$$

$$(\Rightarrow d^2x/dt^2 - \varepsilon\, dx/dt + x = 0)$$

  - Changes to a spiral orbit (whether stable or unstable…)

Poincare-Bendixon Theorem



- For systems of **two** equations

  $dx/dt = F(x,y)$ & $dy/dt = G(x,y)$,

- The following criterion determines the existence of a limit cycle:
- **Suppose** a bounded region D in the plane is so that **no trajectory can exit D** (on boundary, the vector field (F,G) points inside or tangentially) and either there are **no steady states inside** or there is **a single steady state that is repelling** then there is a periodic orbit inside D.
- If the periodic orbit is unique then it is a limit cycle.

Bendixon's Criterion

- Given region D simply-connected (no holes)
- **if** the divergence of the vector field is **always positive or is always negative inside** D, then there **cannot be a periodic orbit inside** D:

$F(x,y) = [f(x,y)\ g(x,y)]^T$ & $\operatorname{div} F = \partial f/\partial x + \partial f/\partial y$

- By Gauss divergence theorem:

$$\int\int_D \operatorname{div} F\, dx\, dy = \int_C n \cdot F \neq 0.$$

- Thus F is not tangential to any closed path…No periodic orbit inside D!

Van der Pol Equation

- Consider a system involving two variables: e.g., an mRNA and a protein: x and y.
- For instance, consider the equations:

$$dx/dt = y - x^3 + x$$

$$dy/dt = -x$$

Van der Pol Equation

- In other words:

$$d^2x/dt^2 = dy/dt + (1–3\,x^2)\,dx/dt = (1–3\,x^2)\,dx/dt – x \text{ or}$$

$$d^2x/dt^2 + (3x^2\text{-}1)\,dx/dt + x = 0$$

- This system has a stable limit cycle!
- These equations were originally introduced to model a "self exciting" electric circuit.

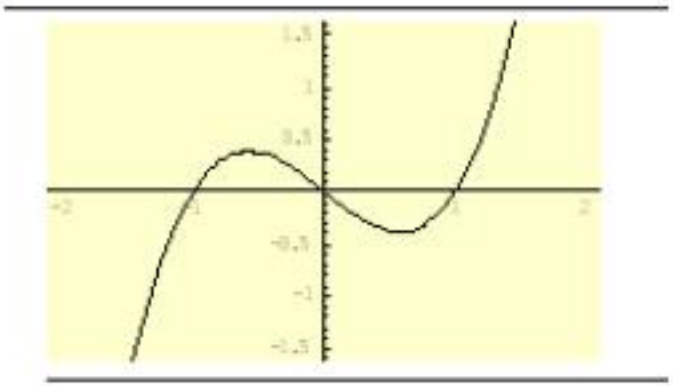
Lienard Equations

- Generalization of Van der Pol system:

$$d^2x/dt^2 + g(x)\, dx/dt + x = 0$$

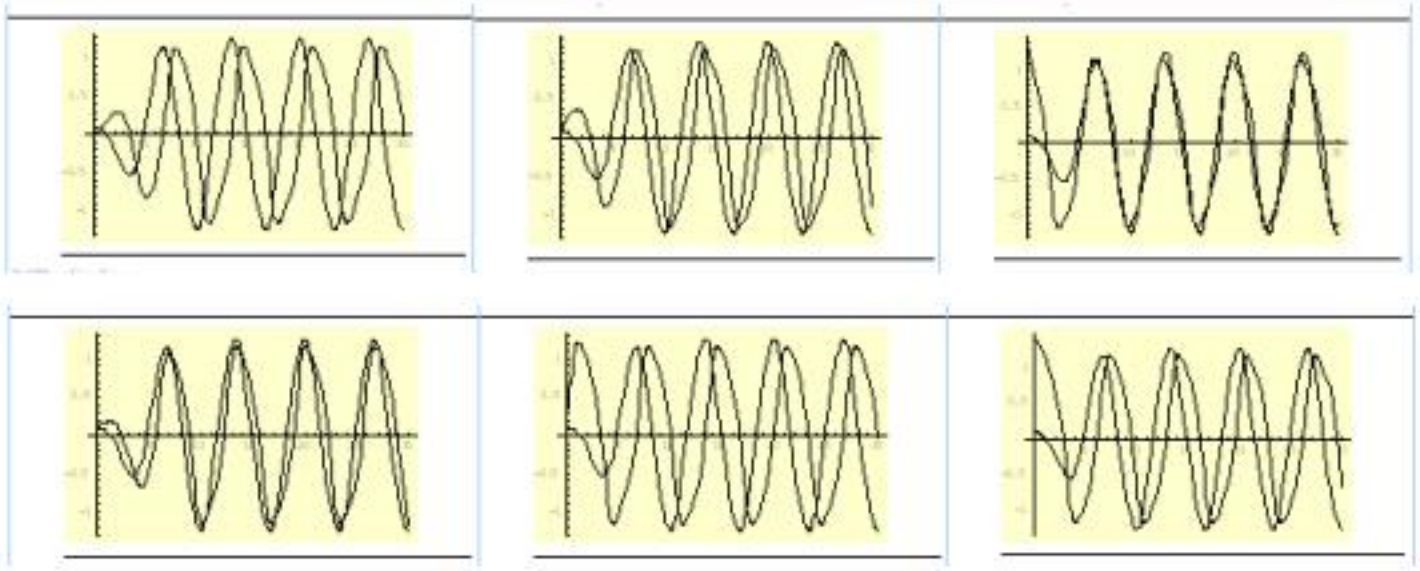
  - If g(x) is zero, this is the linear oscillator.
  - The term involving dx/dt is a "frictional" term, where the friction depends on the position x.
  - For small x we are going to take g(x) negative so that it is an "anti-frictional" term
  - For large x we are going to take g(x) positive so that it is a "frictional" term
- This is sufficient to guarantee the existence of a robust limit cycle...

When $f(x) = -x + x^3$:

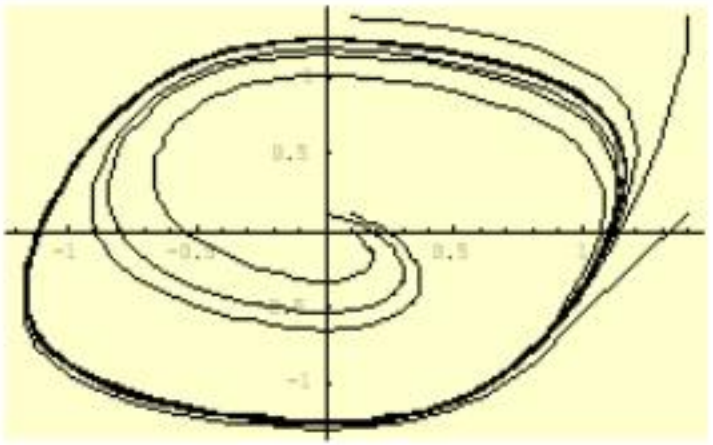


- The graph of $f(x) = -x + x^3$.
- Lienard's Equation: $dx/dt = y - f(x)$; $dy/dt = -x$.
  - f is an odd function of x
  - $f(x) < 0$ in $(0,1)$ and $f(x) > 0$ in $(0, \infty)$
  - f is a strictly monotone increasing function of x (for $x > 1$)
  - f goes to infinity as x goes to ∞
- Sufficient to ensure a limit cycle.

Van der Pol Equation



Van der Pol Equation



- Phase Portrait of the Lienard Equation for $f(x) = -x + x^3$ with $-1.4 \leq x \leq 1.4$ & $1.4 \leq y \leq 1.4$...
- Stability of the limit cycle follows from Poincare-Bendixon.

History

Abbreviated from "**The Birth of Model Checking**" **by** Edmund M. Clarke, Department of Computer Science, Carnegie Mellon University. FLOC 06 Talk.

A Nice Quote from Ed Clarke

- "When the time is ripe for certain things, these things appear in different places in the manner of violets coming to light in early spring. "

(Wolfgang Bolyai to his son Johann in urging him to claim the invention of non-Euclidean geometry without delay.)

Quote from Clarke & Emerson 81

"The task of ***proof construction is in general quite tedious*** *and a good deal of ingenuity may be required to organize the proof in a manageable fashion.*

We argue that ***proof construction is unnecessary in the case of finite state concurrent systems*** *and can be replaced by a* ***model-theoretic approach which***
will mechanically determine if the system meets a specification expressed in propositional temporal logic.

The global state graph of the concurrent systems can be viewed as a finite Kripke structure and ***an efficient algorithm*** *can be given to determine* ***whether a structure is a model of a particular formula*** *(i.e. to* [illegible] *its specification)".*

The Model Checking Problem

The Model Checking Problem (CE81):

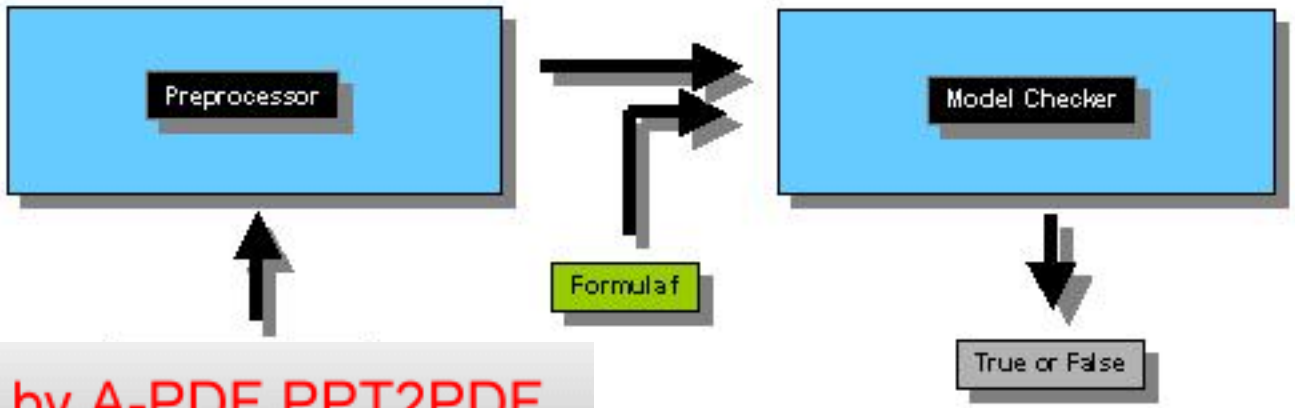
Let M be a **Kripke structure**

(i.e., state-transition graph).

Let f be a **formula of temporal logic**

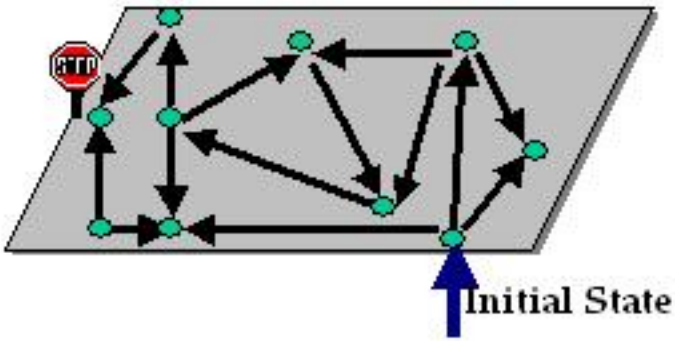
(i.e., the specification).

Find all states s of M such that $M, s \models f$.



Advantages of Model Checking

- No proofs!!
- Fast (compared to other rigorous methods such)
- Diagnostic counterexamples
- No problem with partial specifications
- Logics can easily express main concurrency properties



Safety Property:
bad state unreachable

Counterexample

Main Disadvantages

- Proving a program helps you understand it.
  Bogus!
- Temporal logic specifications are ugly.
  Depends on who is writing them.
- Writing specifications is hard.
  True, but perhaps partially a matter of education.
- State explosion is a major problem.
  Absolutely true, but we are making progress!

Parallel Developments

Tadeo Murata:

- Petrinet tools, late 70s

Kurt Jensen:

- Petrinet tools, late 70s

Gregor Bochmann:

- Protocol Verification, 1978

Holzmann:

- Protocol Verification, 1978-79

Mu-calculus

- **A. Tarski**, A Lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics 5, 285-309, 1955

- **S. C. Kleene**, Introduction to Meta-Mathematics, 1964. (First Recursion Theorem)
- **J. W. de Bakker and D. Scott**, A Theory of programs; unpublished notes, IBM Vienna, 1969.
- **D.M.R. Park**, Fixpoint induction and proofs of program properties, in Machine Intelligence 5, 1970
- **D.M.R. Park**, Finiteness is Mu-Ineffable, University of Warwick Theory of Computation Report, July 1974.
- **E. A. Emerson and E.M. Clarke**, Characterizing correctness properties of Parallel programs using fixpoints in LNCS 85, Automata,Languages, and Programming, pp 169-181, Springer 1980.
- **D. Kozen**, Results on the propositional mu-calculus, Theoretical Computer Science, 27:333-354, 1983.

Dataflow Analysis

- **G. Killdall:** Lattice theoretic approach to iterative data flow Analysis (73)
- **Ullmann and students:** Monotone data-flow analysis frameworks (76)
- **L. Fosdick and L. Osterweil:** Data-flow analysis for static error detection (76)
- **P. Cousot:** Abstract Interpretation and Widening (77, 79)
- **D. Schmidt:** "Data-flow Analysis is Model Checking of Abstract Interpretations" (98)

Temporal Logic

Temporal logics describe the **ordering of events in time** without introducing time explicitly.

They were **developed by philosophers** for investigating how time is used in natural language arguments.

Most have an operator like **G** f that is true in the present if f **is always true in the future.**

To assert that two events e_1 and e_2 never occur at the same time, one writes **G** $(\neg e_1 \vee \neg e_2)$.

The meaning of a temporal logic formula is determined with respect to a labeled state-transition graph or ***Kripke structure***.

Temporal Logic and Program Verification

- **Burstall 74**, **Kroeger 77**, and **Pnueli 77**, all proposed using temporal logic for reasoning about computer programs.

- **Pnueli 77** was the first to use **temporal logic for reasoning** about **concurrency**.
- He **proved program properties** from a set of axioms that described the behavior of the individual statements.
- The method was **extended to sequential circuits** by Bochmann 82 and Owicki and Malachi 81.

- Since **proofs were constructed by hand**, the technique was often difficult to use in practice.

Pnueli 77 and Model Checking

- **Did Pnueli invent Model Checking in 1977 ???**
- **Section on Finite State Systems** most relevant for this meeting.
- **Theorem 4:** *The validity of an arbitrary eventuality G(A → F B) is decidable for for any finite state system.*
- Proof of this theorem uses strongly connected components and is very similar to the technique used for EG(P) in CES 83 / 86.

Expressive Power of Temporal Logic

- **Lamport** was the first to investigate the expressive power of various temporal logics for verification.
- His **1980 POPL paper** discussed two logics: a simple linear-time logic and a simple branching-time logic.
- **Branching-time logic** could not express certain natural fairness properties that can easily expressed in the linear-time logic.
- **Linear-time logic** could not express the possibility of an event occurring at sometime in the future along some computation path.
- **Technical difficulties** with method that Lamport used for his results (somewhat like comparing "apples and oranges").
- **Emerson and Halpern** fixed these problems in an 83 POPL paper.

Clarke and Emerson 81

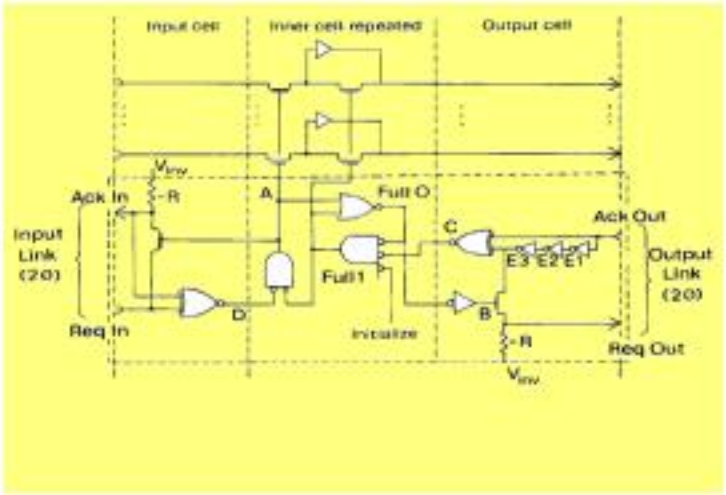
- ***Edmund M. Clarke and E. Allen Emerson***, Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. Logics of Programs Workshop, Yorktown Heights, New York, May 1981, LNCS 131. Also in Emerson's Thesis (81).
- The temporal logic model checking algorithms of Clarke and Emerson 1980's allowed this type of reasoning to be automated.
- Checking that a single model satisfies a formula is much easier than proving the validity of a formula for all models.
- **The algorithm of Clarke and Emerson for CTL was polynomial in |M | and in |f |.**
- They also showed how fairness could be handled without changing the complexity of the algorithm.

The EMC Model Checker

- **Clarke, Emerson, and Sistla** (83 / 86) devised an improved algorithm that was linear in the product of the |M | and |f |.
- The algorithm was implemented in the EMC model checker and used to check a number of network protocols and sequential circuits.
- Could check state transition graphs with between 10^4 and 10^5 states at a rate of about 100 states per second for typical formulas.
- In spite of these limitations, EMC was used successfully to find previously unknown errors in several published circuit designs.
  - EMC tool
  - Fairness Constraints
  - Emptiness of Non-deterministic Buchi Automata

Hardware Verification

- **B. Mishra and E. M. Clarke**, Automatic and Hierarchical Verification of Asynchronous Circuits using Temporal Logic, CMU Tech Report (CMU-CS-83-155) and Theoretical Computer Science 38, 1985, pages 269-291.
- ***First use of Model Checking for Hardware Verification!!***
- **(found bug in the Seitz's FIFO Queue from Mead and Conway, Introduction to VLSI Systems).**
- **Mishra and Clarke 83; Browne, Clarke, and Dill 86; Dill and Clarke 86**



Complexity of LTL

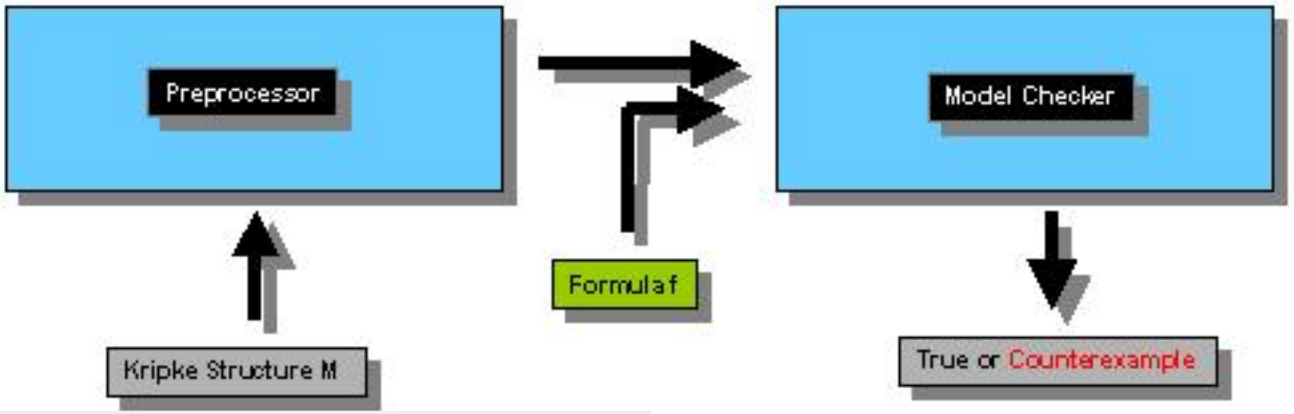
- **Sistla and Clarke (82, 83)** analyzed the **model checking problem for LTL and showed that the problem was PSPACE-complete**.
- **Pnueli and Lichtenstein (85)** an algorithm that is exponential in the length of the formula, but linear in the size of the Model.
  - Based on this observation, they argued that LTL model checking is feasible for short formulas.

CTL* Model Checking

- **CTL*** is a very expressive logic that **combines both branching-time and linear-time operators**.
- **Model checking** for this logic was first considered in CES 83 / 86 where it was shown to be PSPACE-complete.
- Can show that **CTL*** and **LTL** model checking have the same complexity in |M | and |f | (Emerson and Lei 85).
- Thus, for purposes of model checking, there is no practical complexity advantage to restricting oneself to a linear temporal logic.

Witnesses and Counterexamples

- EMC did not give counterexamples for universal CTL properties that were false or witnesses for existential properties that were true.
- **Michael C. Browne** (Mike) added this feature to the **MCB** model checker in 1984.
  - It has been an important feature of Model Checkers ever since.



Automata Theoretic Techniques

- Some verifiers use **automata** as both **specifications** and **implementations**.
- The implementation is checked to see whether its behavior conforms to that of specification.
- Thus, an implementation at one level can be used as a specification for the next level.
- The use of language containment is implicit in the work of Kurshan, which ultimately resulted in the development of the COSPAN verifier.
- Aggarwal, **Kurshan**, and Sabnani 83; **Dill's** Thesis 87; Har'El and Kurshan 90

Two Big Breakthroughs!

- Significant progress was made on the State Explosion Problem around 1990:


- **Symbolic Model Checking**
  Coudert, Berthet, and Madre 89
  Burch, Clarke, McMillan, Dill, and Hwang 90;
  Ken McMillan's thesis 92

- **The Partial Order Reduction**
  Valmari 90
  Godefroid 90
  Peled 94



Dealing with Very Complex Systems

- Special techniques are needed when symbolic methods and the partial order reduction don't work.

- Four basic techniques are

  - *Compositional reasoning,*
  - *Abstraction,*
  - *Symmetry reduction, and*
  - *Induction and parameterized verification*

Big Events in Model Checking since 1990

- Timed and Hybrid Automata
- Model Checking for Security Protocols
- Bounded Model Checking
- Localization Reduction and CEGAR
- Compositional Model Checking and Learning
- Infinite State Systems (e.g., pushdown systems)

Challenges for the Future

- **Software Model Checking**, Model Checking and Static Analysis
- **Model Checking and Theorem Proving** (PVS, STEP, SyMP, Maude)
- Exploiting the **Power of SAT**, Satisfiability Modulo Theories (SMT)
- **Probabilistic Model Checking**
- **Efficient** Model Checking for Timed and **Hybrid Automata**
- Interpreting **Counterexamples**
- **Coverage** (incomplete Model Checking, have I checked enough properties?)
- Scaling up even more!!

..to be continued…

Symbolic Computation
Algebraic Biology III

Bud Mishra

Courant Inst, NYU

NYU SoM, TIFR, MSSM

Semi-Algebraic Geometry

- Real Closed Field
- Tarski Algebra
- Decision Theories
- Hybrid Models
- Algorithmic Algebraic Model

Symbolic Computation
Algebraic Biology IV

Bud Mishra

Courant Inst, NYU

NYU SoM, TIFR, MSSM

Hybrid Systems

- Hybrid Models
- Algorithmic Algebraic Models & Model Checking
- O-minimal Systems & SaCoRe
- IDA
- Open Problems

The End