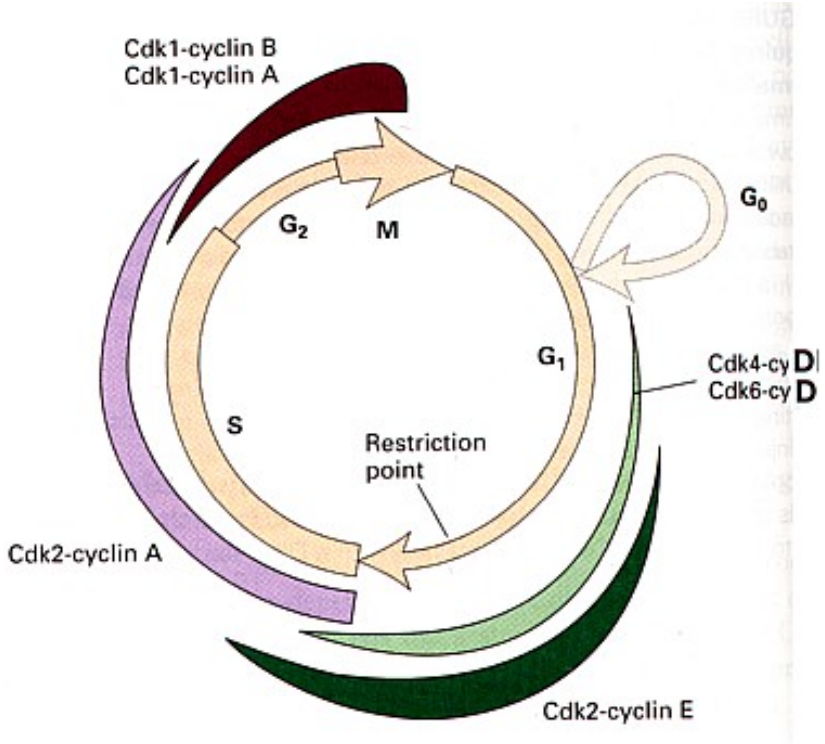
Overview of the Tutorial

1. Introduction
   - Transposing programming concepts to the analysis of living processes
2. Rule-based modeling of biochemical systems
   - Syntax: molecules, reactions, regulations, SBML/SBGN Biocham notations
   - Semantics: Boolean, Differential and Stochastic interpretations of reactions
   - *Static analyses*: consistency, influence graph circuits, protein functions,&
   - Examples in cell signaling, gene expression, virus infection, cell cycle
3. Temporal Logic based formalization of biological properties
   - *Qualitative model-checking* in propositional Computation Tree Logic CTL
   - Quantitative model-checking in Linear Time Logic LTL(R)
   - *Parameter search* in high dimension w.r.t. LTL(R) specifications
   - *Robustness and sensitivity analyses* w.r.t. LTL(R) specifications
4. Conclusion

Cell Cycle Control by Cyclins: G1→S→G2→M

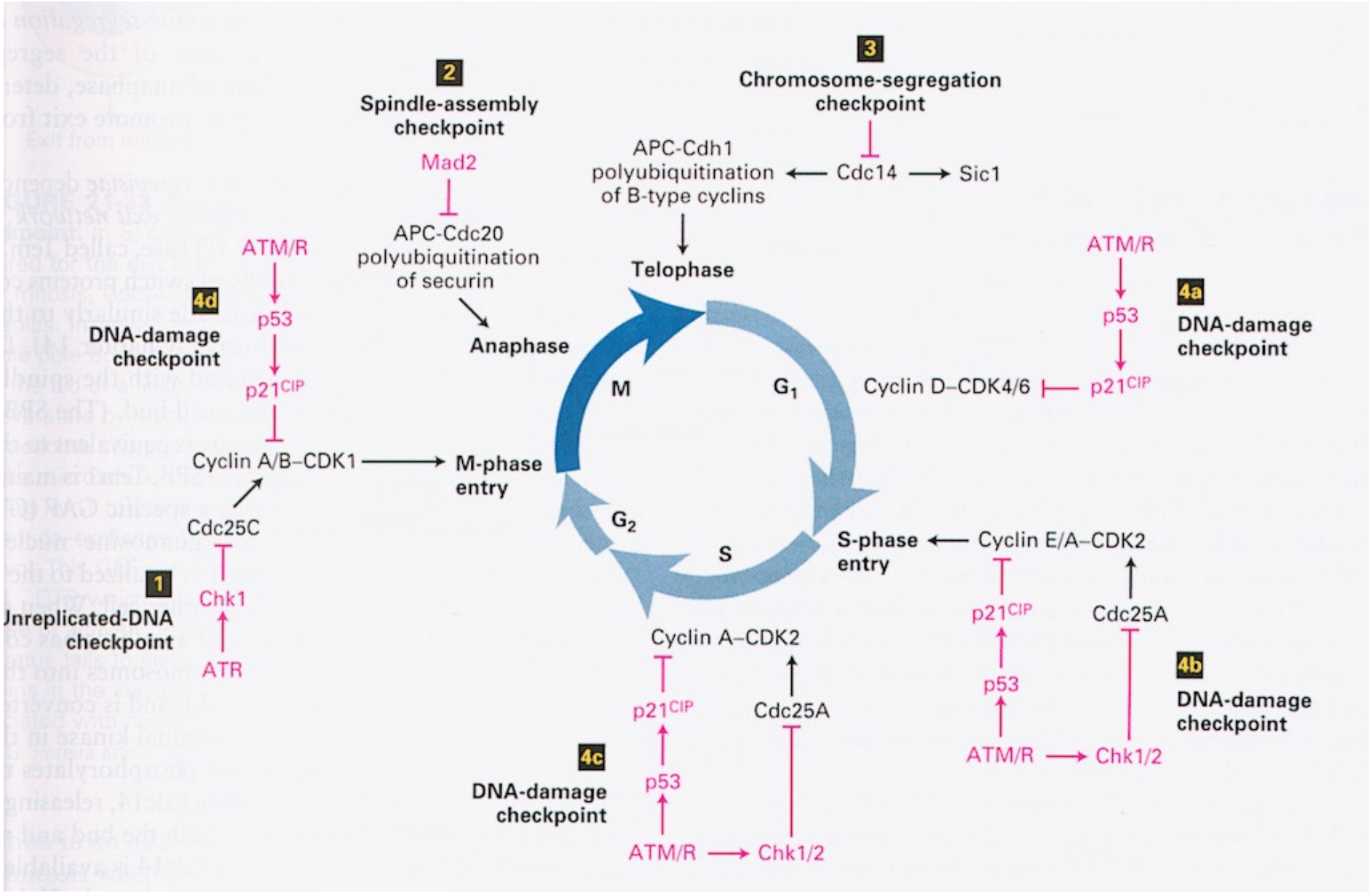
Sir Paul Nurse
Nobel prize 2001



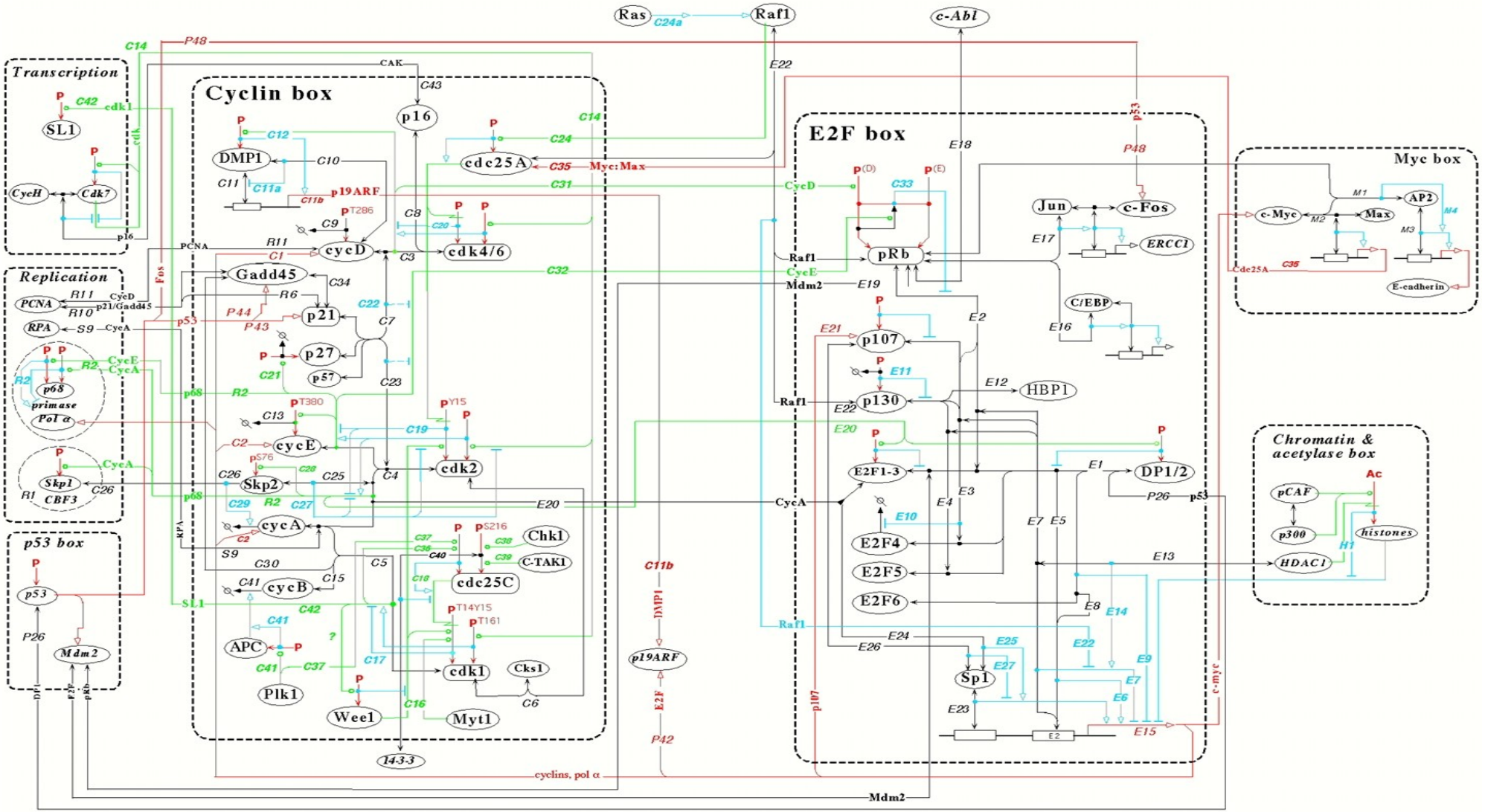
G1: **CdK4-CycD**
Cdk6-CycD
Cdk2-CycE

S: **Cdk2-CycA**

G2,M: **Cdk1-CycA**
Cdk1-CycB (MPF)

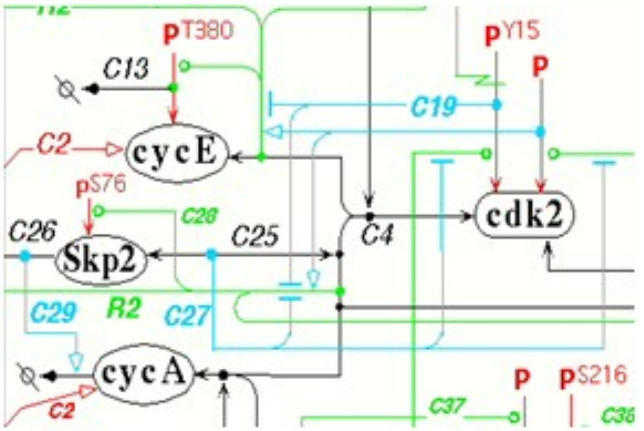

2
Spindle-assembly checkpoint
Mad2
APC-Cdc20 polyubiquitination of securin
Anaphase
3
Chromosome-segregation checkpoint
APC-Cdh1 polyubiquitination of B-type cyclins
Cdc14
Sic1
Telophase
ATM/R
4d
DNA-damage checkpoint
p53
p21CIP
Cyclin A/B–CDK1
M-phase entry
Cdc25C
1
Unreplicated-DNA checkpoint
Chk1
ATR
M
G1
G2
S
ATM/R
4a
DNA-damage checkpoint
p53
p21CIP
Cyclin D–CDK4/6
S-phase entry
Cyclin E/A–CDK2
p21CIP
Cdc25A
4b
DNA-damage checkpoint
p53
ATM/R
Chk1/2
Cyclin A–CDK2
p21CIP
Cdc25A
4c
DNA-damage checkpoint
p53
ATM/R
Chk1/2

Mammalian Cell Cycle Control Map [Kohn 99]



Kohn s map detail for Cdk2

Complexation with CycA and CycE



Biocham Rule Patterns:

```
cdk2~$P + cycA-$C => cdk2~$P-cycA-$C
    where $C in {_,cks1} .
cdk2~$P + cycE~$Q-$C => cdk2~$P-cycE~$Q-$C
    where $C in {_,cks1} .
p57 + cdk2~$P-cycA-$C => p57-cdk2~$P-cycA-$C
    where $C in {_, cks1}.
cycE-$C =[cdk2~{p2}-cycE-$S]=> cycE~{T380}-$C
    where $S in {_, cks1} and $C in {_, cdk2~?, cdk2~?-cks1}
```

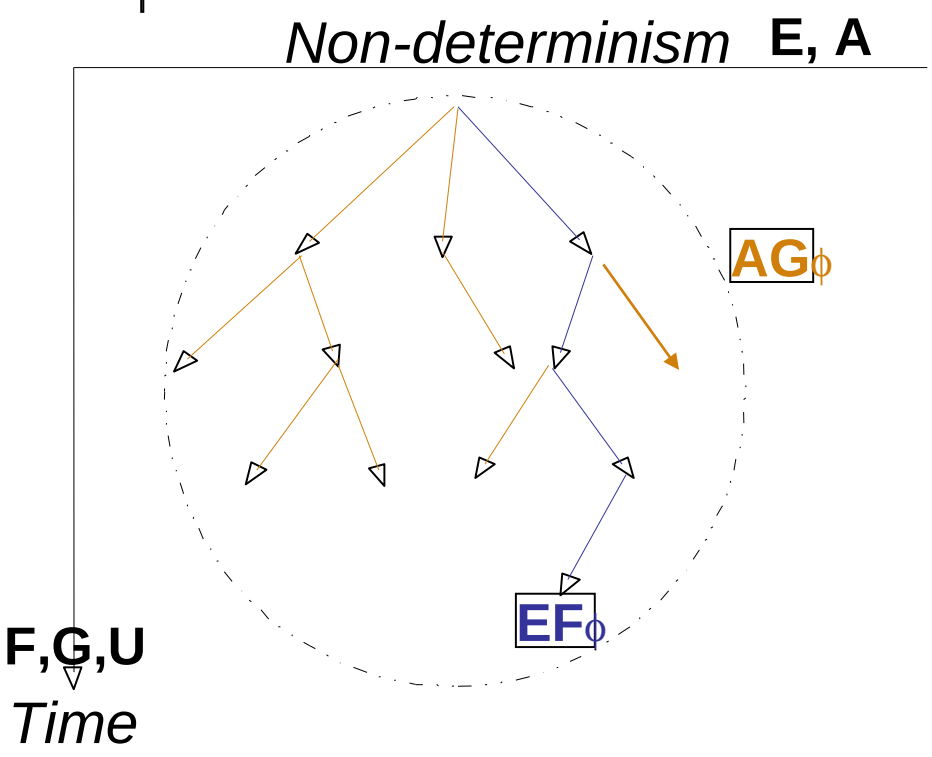
Total: 147 rule patterns 2733 expanded rules [Chiaverini Danos 03]

Computation Tree Logic CTL

Temporal logics extend classical logic with modal operators for time & non-det.

Introduced for program verification by [Pnueli 77]

Non-det. / Time	**E** exists	**A** always
X next time	**EX**(φ)	**AX**(φ)
F finally	**EF**(φ) ***¬AG(¬ φ)***	**AF**(φ) *liveness*
G globally	**EG**(φ) ***¬ AF(¬ φ)***	**AG**() *safety*
U until	**E** (φ1 **U** φ2)	**A** (φ1 **U** φ2)



Biological Properties formalized in CTL (1/3)

About *reachability*:

- Can the cell produce some protein P? `reachable(P)==EF(P)`
- Can the cell produce P, Q and not R?

Biological Properties formalized in CTL (1/3)

About *reachability*:

- Can the cell produce some protein P? **reachable(P)==EF(P)**
- Can the cell produce P, Q and not R? **reachable(P^Q^¬R)**
- Can the cell always produce P?

Biological Properties formalized in CTL (1/3)

About *reachability*:

- Can the cell produce some protein P? **reachable(P)==EF(P)**
- Can the cell produce P, Q and not R? **reachable(P^Q^¬R)**
- Can the cell always produce P? **AG(reachable(P))**

About *pathways*:

- Can the cell reach a set s of (partially described) states while passing by another set of states s_2?

Biological Properties formalized in CTL (1/3)

About *reachability*:

- Can the cell produce some protein P? **reachable(P)==EF(P)**
- Can the cell produce P, Q and not R? **reachable(P^Q^¬R)**
- Can the cell always produce P? **AG(reachable(P))**

About *pathways*:

- Can the cell reach a set S of (partially described) states while passing by another set of states S_2? **EF(S_2^EFS)**
- Is it possible to produce P without Q ?

Biological Properties formalized in CTL (1/3)

About *reachability*:

- Can the cell produce some protein P? **reachable(P)==EF(P)**
- Can the cell produce P, Q and not R? **reachable(P^Q^¬R)**
- Can the cell always produce P? **AG(reachable(P))**

About *pathways*:

- Can the cell reach a set s of (partially described) states while passing by another set of states s_2? **EF(s_2^EFs)**
- Is it possible to produce P without Q ? **E(¬Q U P)**
- Is state s_2 a necessary checkpoint for reaching state s?

checkpoint(s_2,s)== ¬E(¬s_2 U s)

Biological Properties formalized in CTL (1/3)

About *reachability*:

- Can the cell produce some protein P? **reachable(P)==EF(P)**
- Can the cell produce P, Q and not R? **reachable(P^Q^¬R)**
- Can the cell always produce P? **AG(reachable(P))**

About *pathways*:

- Can the cell reach a set s of (partially described) states while passing by another set of states s_2? **EF(s_2^EFs)**
- Is it possible to produce P without Q ? **E(¬Q U P)**
- Is state s_2 a necessary checkpoint for reaching state s?

  checkpoint(s_2,s)== ¬E(¬s_2 U s)
- Is s_2 always a checkpoint for s? **AG(¬s -> checkpoint(s_2,s))**

Biological Properties formalized in CTL (2/3)

About *stability*:

- Is state `s` a stable state? **`stable(s)== AG(s)`**

Biological Properties formalized in CTL (2/3)

About *stability*:

- Is state `s` a stable state? **`stable(s)== AG(s)`**
- Is `s` a steady state (with possibility of escaping) ? **`steady(s)==EG(s)`**

- Can the cell reach a stable state `s`?

Biological Properties formalized in CTL (2/3)

About *stability*:

- Is state `s` a stable state? **`stable(s)== AG(s)`**
- Is `s` a steady state (with possibility of escaping) ? **`steady(s)==EG(s)`**

- Can the cell reach a stable state `s`? **`EF(stable(s))`**
  alternance of path quantifiers **`EFAG`** ϕ,
  not in Linear Time Logic LTL (fragment without path quantifiers)
  `FG` ϕ is *not in LTL*
- Must the cell reach a stable state `s`?

Biological Properties formalized in CTL (2/3)

About *stability*:

- Is state `s` a stable state? **`stable(s)== AG(s)`**
- Is `s` a steady state (with possibility of escaping) ? **`steady(s)==EG(s)`**

- Can the cell reach a stable state `s`? **`EF(stable(s))`**
  alternance of path quantifiers **`EFAG`** ϕ,
  not in Linear Time Logic LTL (fragment without path quantifiers)
  `FG` ϕ is *not in LTL*
- Must the cell reach a stable state `s`? **`AG(stable(s))`**

- What are the stable states?

Biological Properties formalized in CTL (2/3)

About *stability*:

- Is a set of states `s` a stable state? **`stable(s)== AG(s)`**
- Is `s` a steady state (with possibility of escaping) ? **`steady(s)==EG(s)`**

- Can the cell reach a stable state `s`? **`EF(stable(s))`**
  alternance of path quantifiers **`EFAG`** ϕ,
  not in Linear Time Logic LTL (fragment without path quantifiers)
  `FG` ϕ is *not in LTL*
- Must the cell reach a stable state `s`? **`AG(stable(s))`**

- What are the stable states? *Not expressible in CTL.*
  needs to combine CTL with search [Chan 00, Calzone-Chabrier-Fages-Soliman 05, Fages-Rizk 07 09].

Biological Properties formalized in CTL (3/3)

About *durations*:

- How long does it take for a molecule to become activated?
- In a given time, how many Cyclins A can be accumulated?
- What is the duration of a given cell cycle s phase?

Biological Properties formalized in CTL (3/3)

About *durations*:

- How long does it take for a molecule to become activated?
- In a given time, how many Cyclins A can be accumulated?
- What is the duration of a given cell cycle s phase?

CTL operators abstract from durations. Time intervals can be modeled in FOL by adding numerical constraints for start times and durations.

Biological Properties formalized in CTL (3/3)

About *durations*:

- How long does it take for a molecule to become activated?
- In a given time, how many Cyclins A can be accumulated?
- What is the duration of a given cell cycle s phase?

CTL operators abstract from durations. Time intervals can be modeled in FOL by adding numerical constraints for start times and durations.

About *oscillations*:

- Can the system exhibit a cyclic behavior w.r.t. the presence of P ?

  oscil(P**)== EG((F** ¬P**) ^ (F** P**))**

  temporal operators not preceded by a path operator: CTL* formula

  approximation in CTL: **oscil(**P**)== EG((EF** ¬P**) ^ (EF** P**))**

Basic CTL Model-Checking Algorithm

Model Checking is an algorithm for computing, in a given *finite* Kripke structure K the set of states satisfying a CTL formula:

$$\{s \in S : s \models \phi\}.$$

Basic CTL Model-Checking Algorithm

Model Checking is an algorithm for computing, in a given *finite* Kripke structure K the set of states satisfying a CTL formula:

$$\{s \in S : s \models \phi\}.$$

Represent K as a (finite) graph and iteratively label the nodes with the *subformulas of* ϕ which are true in that node.

Basic CTL Model-Checking Algorithm

Model Checking is an algorithm for computing, in a given *finite* Kripke structure K the set of states satisfying a CTL formula:

$$\{s \in S : s \models \phi\}.$$

Represent K as a (finite) graph and iteratively label the nodes with the *subformulas of* ϕ which are true in that node.

" Add ϕ to the states satisfying ϕ

Basic CTL Model-Checking Algorithm

Model Checking is an algorithm for computing, in a given *finite* Kripke structure K the set of states satisfying a CTL formula:

$$\{s \in S : s \models \phi\}.$$

Represent K as a (finite) graph and iteratively label the nodes with the *subformulas of* ϕ which are true in that node.

- Add ϕ to the states satisfying ϕ
- Add **EF** ϕ (**EX** ϕ) to the (immediate) predecessors of states labeled by ϕ

Basic CTL Model-Checking Algorithm

Model Checking is an algorithm for computing, in a given *finite* Kripke structure K the set of states satisfying a CTL formula:

$$\{s \in S : s \models \phi\}.$$

Represent K as a (finite) graph and iteratively label the nodes with the *subformulas of* ϕ which are true in that node.

- Add ϕ to the states satisfying ϕ
- Add **EF** ϕ (**EX** ϕ) to the (immediate) predecessors of states labeled by ϕ
- Add **E**($\phi 1$ **U** $\phi 2$) to the predecessor states of $\phi 2$ while they satisfy $\phi 1$

Basic CTL Model-Checking Algorithm

Model Checking is an algorithm for computing, in a given *finite* Kripke structure K the set of states satisfying a CTL formula:

$$\{s \in S : s \models \phi\}.$$

Represent K as a (finite) graph and iteratively label the nodes with the *subformulas of* ϕ which are true in that node.

- Add ϕ to the states satisfying ϕ
- Add **EF** ϕ (**EX** ϕ) to the (immediate) predecessors of states labeled by ϕ
- Add **E**($\phi 1$ **U** $\phi 2$) to the predecessor states of $\phi 2$ while they satisfy $\phi 1$
- Add **EG** ϕ to the states for which there exists a path leading to a non trivial strongly connected component of the subgraph of states satisfying ϕ.

Basic CTL Model-Checking Algorithm

Model Checking is an algorithm for computing, in a given *finite* Kripke structure K the set of states satisfying a CTL formula:

$$\{s \in S : s \models \phi\}.$$

Represent K as a (finite) graph and iteratively label the nodes with the *subformulas of* ϕ which are true in that node.

- Add ϕ to the states satisfying ϕ
- Add **EF** ϕ (**EX** ϕ) to the (immediate) predecessors of states labeled by ϕ
- Add **E**($\phi 1$ **U** $\phi 2$) to the predecessor states of $\phi 2$ while they satisfy $\phi 1$
- Add **EG** ϕ to the states for which there exists a path leading to a non trivial strongly connected component of the subgraph of states satisfying ϕ.

Model-checking algorithm in O(|K|*|ϕ|).

Complexity: CTL model-checking is Ptime-complete,

Symbolic CTL Model-Checking Algorithm

Represent *finite* Kripke structures using Boolean constraints for

- sets of states as a boolean constraint c(V)
- the transition relation as a boolean constraint r(V,V)

Symbolic CTL Model-Checking Algorithm

Represent *finite* Kripke structures using Boolean constraints for

- sets of states as a boolean constraint c(V)
- the transition relation as a boolean constraint r(V,V′)

Ordered Binary Decision Diagrams OBDD [Bryant 85] provide *canonical forms* for Boolean formulas (decides SAT in NP, and equivalence TAUT in co-NP)

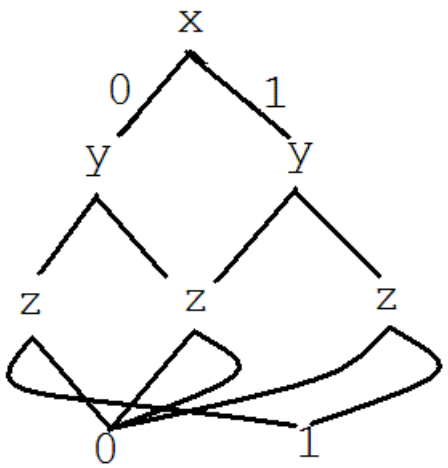
$(x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg x)$

and

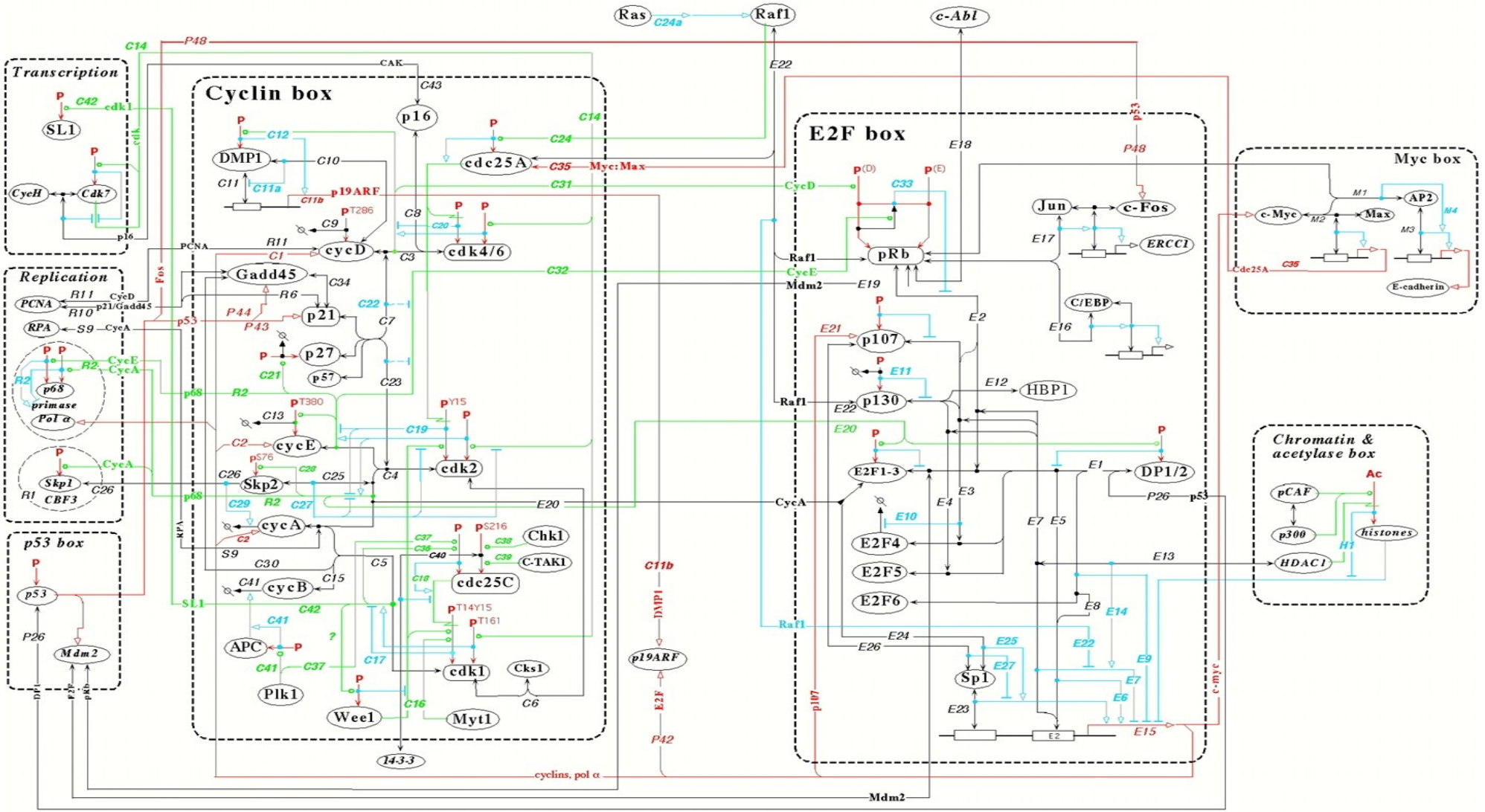
$(x \vee \neg z) \wedge (z \vee \neg y) \wedge (y \vee \neg x)$

are equivalent, they

have the same BDD(x,y,z)



Mammalian Cell Cycle Control Map [Kohn 99]



Mammalian Cell Cycle Control Benchmark

147-2733 rules, 165 proteins and genes, 500 variables, 2^{500} states.

BIOCHAM NuSMV model-checker time in seconds: [Chabrier Fages 03 CMSB]

Initial state G2	Query:	Time:
	compiling	29s
Reachability G1	**EF CycE**	2s
Reachability G1	**EF CycD**	1.9s
Reachability G1	**EF PCNA-CycD**	1.7s
Checkpoint for mitosis complex	**¬ EF (¬ Cdc25~{Nterm} U Cdk1~{Thr161}-CycB)**	2.2s
Oscillations CycA	**EG ((EF ¬ CycA) & (EF CycA))**	31.8s
Osciallations CycB	**EG ((EF ¬ CycB) & (EF CycB))** false !	6s

Linear Time Logic with Constraints LTL(R)

Constraints over *concentrations* and *derivatives* as formulae over the reals:

[M] > 0.2

[M]+[P] > [Q]

d([M])/dt < 0

LTL(R) formulae

minimum threshold value reached: **F**([M]>0.2)

minimum threshold value reached and maintained: **FG**([M]>0.2)

local maximum V: **F** ([M]<V & **F** ([M]=V & **F** ([M]<V))

F ([M]>2 & **F** (d([M])/dt<0 & **F** ([M]<2 & d([M])/dt>0 & **F**(d([M])/dt<0))))

oscil(M,n) defined as at least n alternances of the sign of the derivative

LTL(R) Constraints with Real-time Variable

LTL(R) formulae with real-time variable

Threshold value with a minimum delay

$\mathbf{F}([M]>0.2) \;\&\; \mathbf{G}(Time<5 \Rightarrow [M]<0.2)$

Numerical data time series (for curve fitting)

$\mathbf{F}(Time=1 \;\&\; [M]=0.05 \;\&\; \mathbf{F}(Time=2 \;\&\; [M]=0.12 \;\&\; \mathbf{F}(Time=3 \;\&\; [M]=0.25)))$

Period constraint

$$Period(A,75)= \exists\, t\, \exists v\, \mathbf{F}(Time = t \;\&\; [A] = v \;\&\; d([A])/dt > 0 \;\&\; \mathbf{X}(d([A])/dt < 0) \;\&\; \mathbf{F}(Time = t + 75 \;\&\; [A] = v \;\&\; d([A])/dt > 0 \;\&\; \mathbf{X}(d([A])/dt < 0)))$$

Numerical Integration of ODE Models

$dX/dt = f(X)$.

Initial conditions X_0

Idea: discretize time t_0, $t_1=t_0+\Delta t_0$, $t_2=t_1+\Delta t_1$, &

and compute a numerical trace $(t_0,X_0,dX_0/dt)$, $(t_1,X_1,dX_1/dt)$, & , $(t_n,X_n,dX_n/dt)$

Euler s method $t_{i+1}=t_i+\Delta t$ $X_{i+1}=X_i+f(X_i)*\Delta t$

error estimation $E(X_{i+1})=|f(X_i)-f(X_{i+1})|*\Delta t$

Runge-Kutta s method intermediate computations at $\Delta t/2$

adaptive step method: $\Delta t_{i+1}= \Delta t_i/2$ while E>Emax, otherwise $\Delta t_{i+1}= 2*\Delta t_i$

Rosenbrock s implicit method for stiff systems:

solve $X_{i+1}=X_i+f(X_{i+1})*\Delta t$ by formal differentiation

LTL(R) Trace-based Model Checking Algo

Input: An ODE system *M* given with initial conditions, an LTL(R) formula ϕ

Hypothesis: the formula can be checked over a finite period of time [0,T]

Output: whether ϕ is true in *M*

LTL(R) Trace-based Model Checking Algo

Input: An ODE system *M* given with initial conditions, an LTL(R) formula ϕ

Hypothesis: the formula can be checked over a finite period of time [0,T]

Output: whether ϕ is true in *M*

" Compute a trace by numerical integration from 0 to T

LTL(R) Trace-based Model Checking Algo

Input: An ODE system *M* given with initial conditions, an LTL(R) formula ϕ

Hypothesis: the formula can be checked over a finite period of time [0,T]

Output: whether ϕ is true in *M*

" Compute a trace by numerical integration from 0 to T

Label each state of the trace with the formula sconstraints that are true,

LTL(R) Trace-based Model Checking Algo

Input: An ODE system *M* given with initial conditions, an LTL(R) formula ϕ

Hypothesis: the formula can be checked over a finite period of time [0,T]

Output: whether ϕ is true in *M*

- Compute a trace by numerical integration from 0 to T
  Label each state of the trace with the formula sconstraints that are true,
- Iteratively label the states with the sub-formulae that are true:
  Add **X** ϕ_1 to the immediate predecessors of states labeled by ϕ_1,

LTL(R) Trace-based Model Checking Algo

Input: An ODE system *M* given with initial conditions, an LTL(R) formula ϕ

Hypothesis: the formula can be checked over a finite period of time [0,T]

Output: whether ϕ is true in *M*

- Compute a trace by numerical integration from 0 to T

  Label each state of the trace with the formula sconstraints that are true,
- Iteratively label the states with the sub-formulae that are true:

  Add **X** ϕ_1 to the immediate predecessors of states labeled by ϕ_1,

  Add ϕ_1 **U** ϕ_2 to the predecessors of states labelled by ϕ_2 while they satisfy ϕ_1,

LTL(R) Trace-based Model Checking Algo

Input: An ODE system *M* given with initial conditions, an LTL(R) formula ϕ

Hypothesis: the formula can be checked over a finite period of time [0,T]

Output: whether ϕ is true in *M*

- Compute a trace by numerical integration from 0 to T
  Label each state of the trace with the formula sconstraints that are true,
- Iteratively label the states with the sub-formulae that are true:
  - Add **X** ϕ_1 to the immediate predecessors of states labeled by ϕ_1,
  - Add ϕ_1 **U** ϕ_2 to the predecessors of states labelled by ϕ_2 while they satisfy ϕ_1,
  - Add ϕ_1 **W** ϕ_2 to the states labelled by $\phi_1 \wedge \phi_2$, to the last state if it is labelled by ϕ_1, and to the predecessors of states labelled by ϕ_1 **W** ϕ_2 while they satisfy ϕ_1,

LTL(R) Trace-based Model Checking Algo

Input: An ODE system *M* given with initial conditions, an LTL(R) formula ϕ

Hypothesis: the formula can be checked over a finite period of time [0,T]

Output: whether ϕ is true in *M*

- Compute a trace by numerical integration from 0 to T
  Label each state of the trace with the formula sconstraints that are true,
- Iteratively label the states with the sub-formulae that are true:
  - Add **X** ϕ_1 to the immediate predecessors of states labeled by ϕ_1,
  - Add ϕ_1 **U** ϕ_2 to the predecessors of states labelled by ϕ_2 while they satisfy ϕ_1,
  - Add ϕ_1 **W** ϕ_2 to the states labelled by $\phi_1 \wedge \phi_2$, to the last state if it is labelled by ϕ_1, and to the predecessors of states labelled by ϕ_1 **W** ϕ_2 while they satisfy ϕ_1,
- Return true if the initial state is labelled by ϕ, and false otherwise

Naïve Parameter Search Algorithm

Input: an ODE model M(p) with n parameters p in range [pmin,pmax],
an LTL(R) specification ϕ

Output: parameter values v such that M(v) |= ϕ
or fail if no such values

Naïve Parameter Search Algorithm

Input: an ODE model M(p) with n parameters p in range [pmin,pmax],
an LTL(R) specification ϕ

Output: parameter values v such that M(v) |= ϕ
or fail if no such values

6. Scan the parameter value space [pmin,pmax]^n with a fixed step

Naïve Parameter Search Algorithm

Input: an ODE model M(p) with n parameters p in range [pmin,pmax],
an LTL(R) specification ϕ

Output: parameter values v such that M(v) |= ϕ
or fail if no such values

" Scan the parameter value space [pmin,pmax]^n with a fixed step

" Test whether M(v) |= ϕ by trace-based model checking

Naïve Parameter Search Algorithm

Input: an ODE model M(p) with n parameters p in range [pmin,pmax],
an LTL(R) specification ϕ

Output: parameter values v such that M(v) |= ϕ
or fail if no such values

- Scan the parameter value space [pmin,pmax]^n with a fixed step
- Test whether M(v) |= ϕ by trace-based model checking
- Return the first value set v which satisfies ϕ

Naïve Parameter Search Algorithm

Input: an ODE model M(p) with n parameters p in range [pmin,pmax],
an LTL(R) specification ϕ

Output: parameter values v such that M(v) |= ϕ
or fail if no such values

" Scan the parameter value space [pmin,pmax]^n with a fixed step
" Test whether M(v) |= ϕ by trace-based model checking
" Return the first value set v which satisfies ϕ

Exponential complexity in O(s^n) where s is the maximum number of tried values in the range of n parameters

Gradient-based methods need a satisfaction degree for LTL(R) formulae&

Cell Cycle Control Model [Tyson 91]

k1 for _=>Cyclin.
k2*[Cyclin] for Cyclin=>_.
k3*[Cyclin]*[Cdc2~{p1}] for Cyclin+Cdc2~{p1}=>Cdc2~{p1}-Cyclin~{p1}.
k4p*[Cdc2~{p1}-Cyclin~{p1}] for Cdc2~{p1}-Cyclin~{p1}=>Cdc2-Cyclin~{p1}.
k4*[Cdc2-Cyclin~{p1}]^2*[Cdc2~{p1}-Cyclin~{p1}] for
Cdc2~{p1}-Cyclin~{p1}=[Cdc2-Cyclin~{p1}]=>Cdc2-Cyclin~{p1}.
k5*[Cdc2-Cyclin~{p1}] for Cdc2-Cyclin~{p1}=>Cdc2~{p1}-Cyclin~{p1}.
k6*[Cdc2-Cyclin~{p1}] for Cdc2-Cyclin~{p1}=>Cdc2+Cyclin~{p1}.
k7*[Cyclin~{p1}] for Cyclin~{p1}=>_.
k8*[Cdc2] for Cdc2=>Cdc2~{p1}.
k9*[Cdc2~{p1}] for Cdc2~{p1}=>Cdc2.
parameter(k1,0.015). parameter(k2,0.015). parameter(k3,200).
parameter(k4p,0.018). parameter(k4,180). parameter(k5,0).
parameter(k6,1). parameter(k7,0.6). parameter(k8,100).parameter(k9,100).
present(Cdc2,1).

Learning Parameters from Temporal Properties

```
biocham: learn_parameter([k3,k4],[(0,200),(0,200)],20,
                                    oscil(Cdc2-Cyclin~{p1},3),150).
```

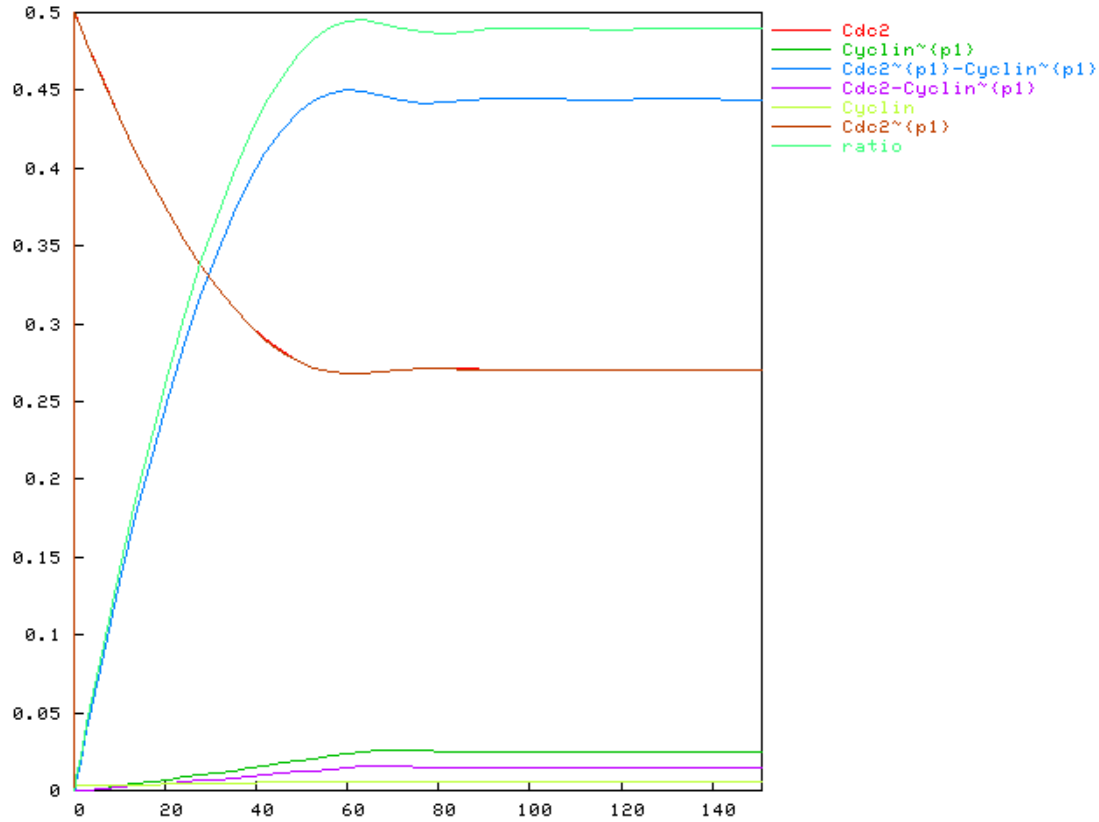
Learning Parameters from Temporal Properties

biocham: learn_parameter([k3,k4],[(0,200),(0,200)],20,
oscil(Cdc2-Cyclin~{p1}.3).150).

First values found :

parameter(k3,10).

parameter(k4,70).



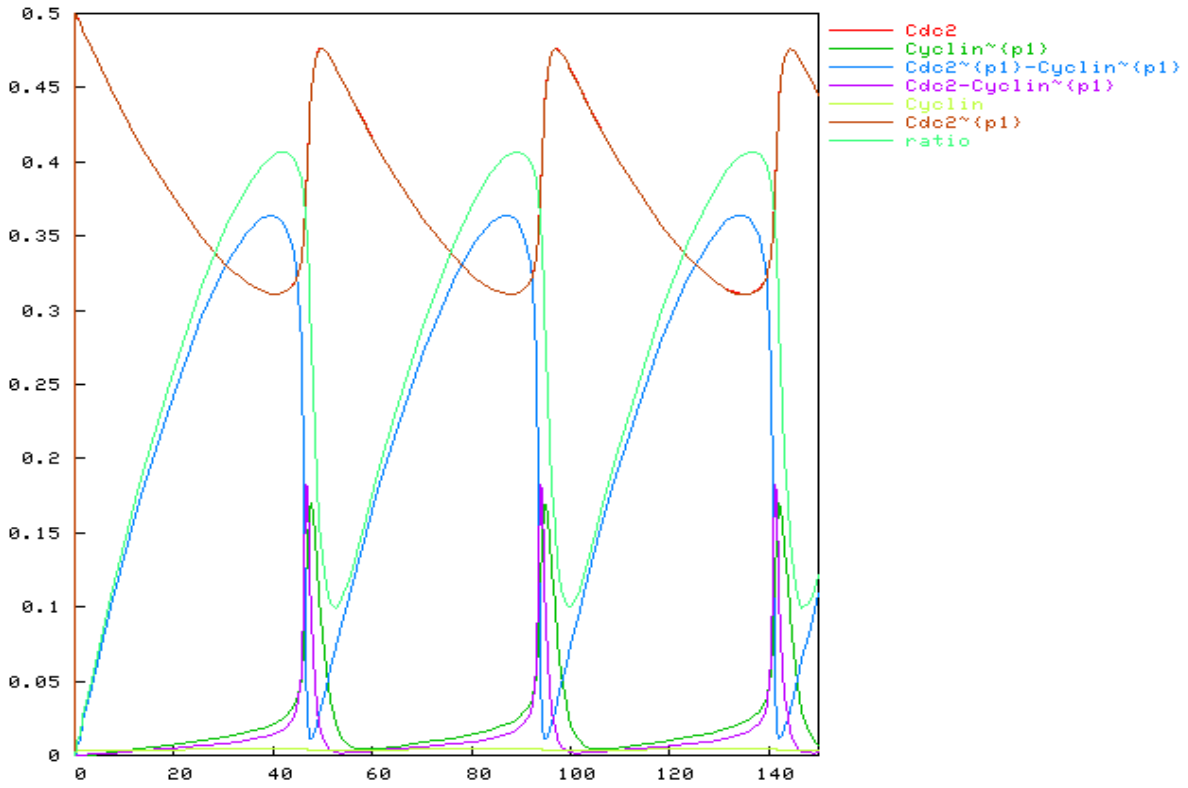
Learning Parameters from Temporal Properties

biocham: learn_parameter([k3,k4],[(0,200),(0,200)],20,
oscil(Cdc2-Cyclin~{p1},3) & F([Cdc2-Cyclin~{p1}]>0.15), 150).

First values found :

parameter(k3,10).

parameter(k4,120).



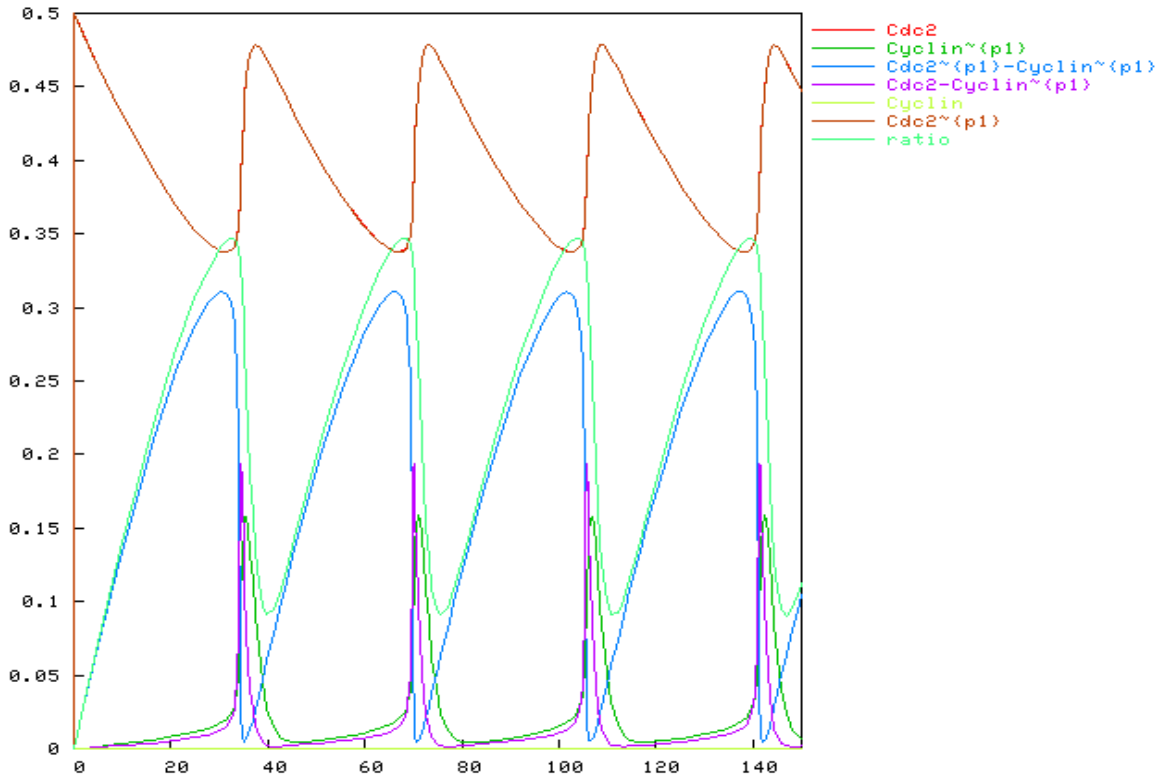
Learning Parameters from Temporal Properties

biocham: learn_parameter([k3,k4],[(0,200),(0,200)],20,
period(Cdc2-Cyclin~{p1},35), 150).

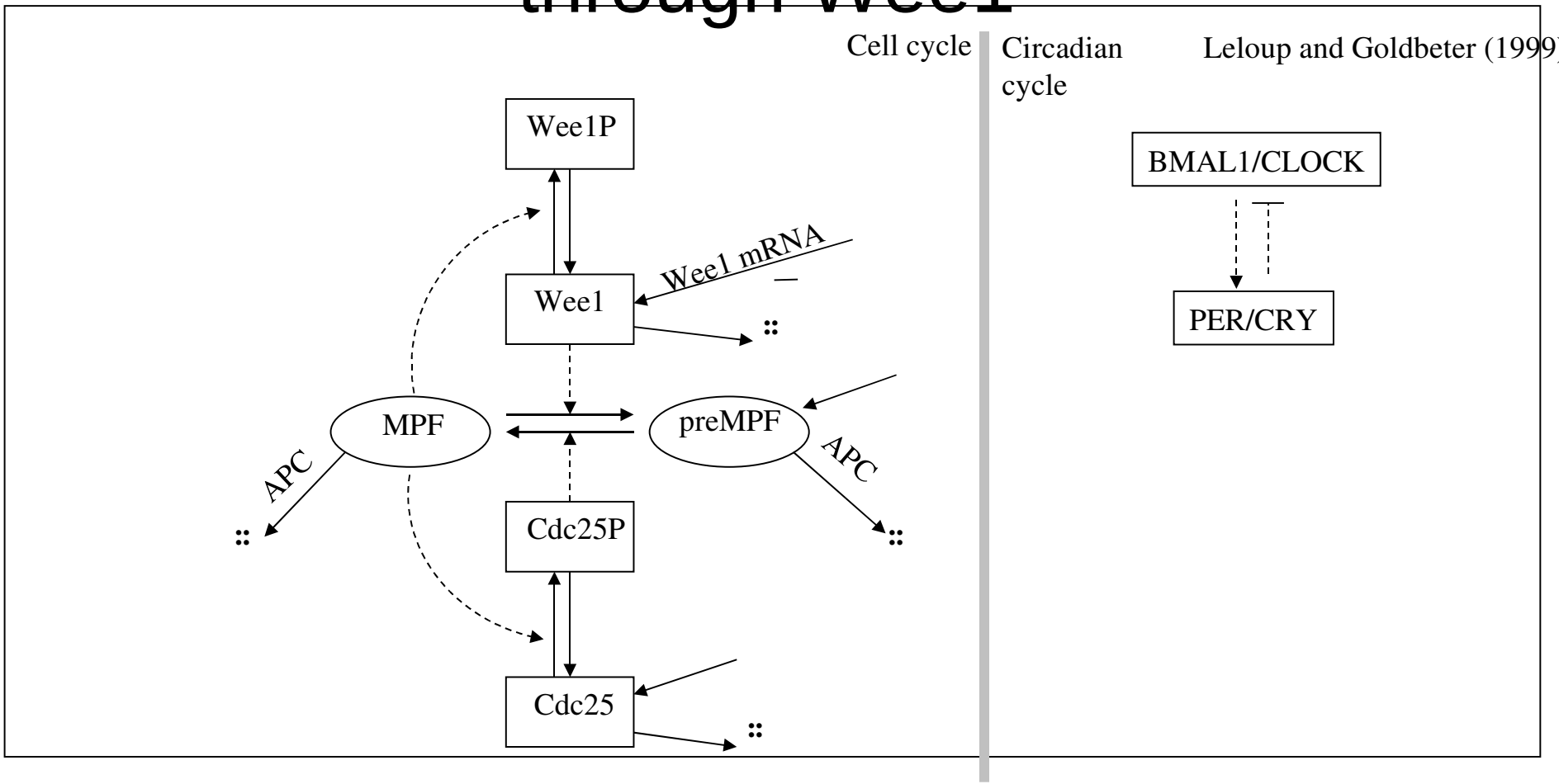
First values found:

parameter(k3,10).

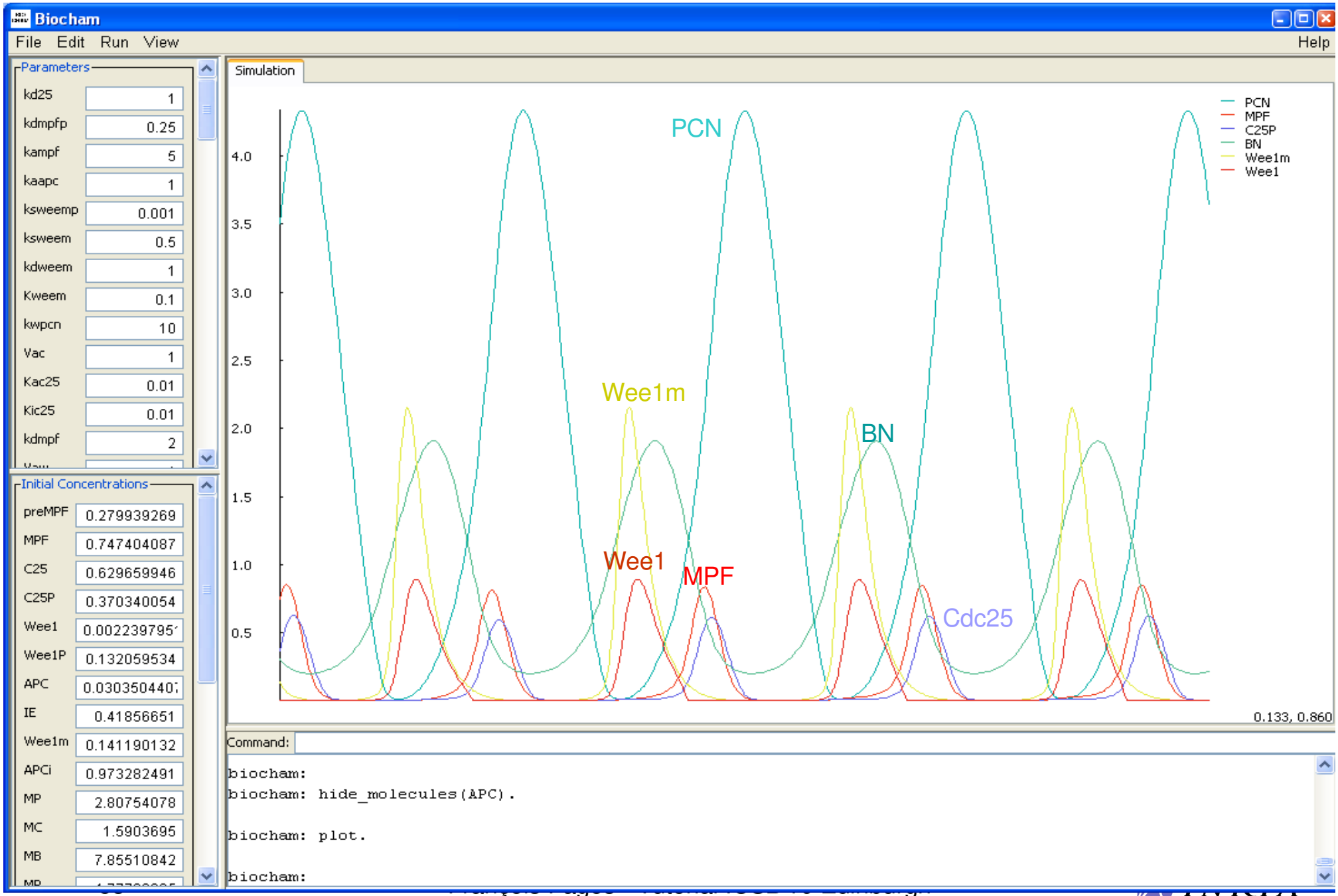
parameter(k4,280).



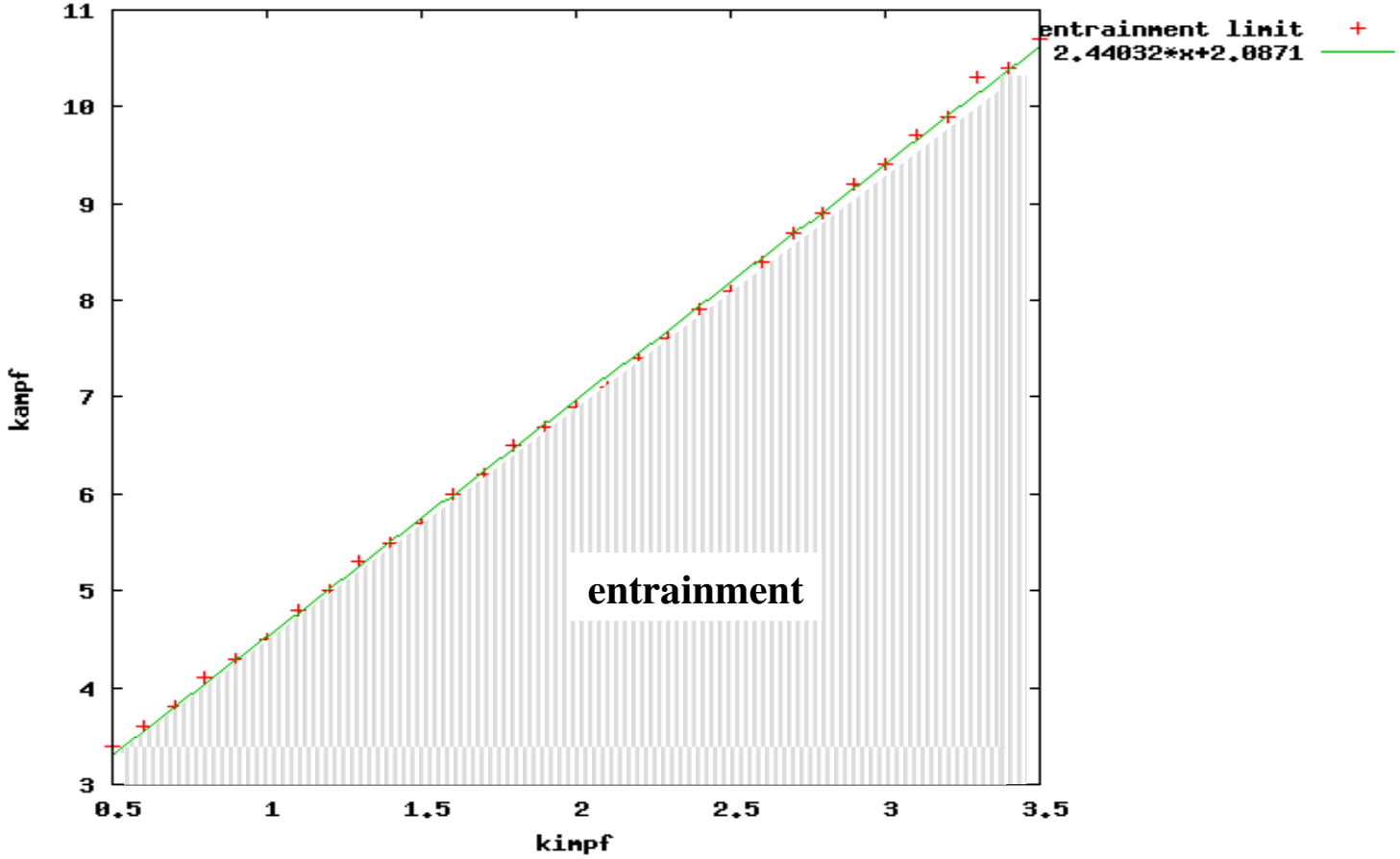
Coupling Cell and Circadian Cycles through Wee1



[L. Calzone, S. Soliman 2006]


Biocham
File Edit Run View
Help
Parameters
kd25 1
kdmpfp 0.25
kampf 5
kaapc 1
ksweemp 0.001
ksweem 0.5
kdweem 1
Kweem 0.1
kwpcn 10
Vac 1
Kac25 0.01
Kic25 0.01
kdmpf 2
Initial Concentrations
preMPF 0.279939269
MPF 0.747404087
C25 0.629659946
C25P 0.370340054
Wee1 0.002239795
Wee1P 0.132059534
APC 0.030350440
IE 0.41856651
Wee1m 0.141190132
APCi 0.973282491
MP 2.80754078
MC 1.5903695
MB 7.85510842
Simulation
PCN
MPF
C25P
BN
Wee1m
Wee1
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
PCN
Wee1m
BN
Wee1
MPF
Cdc25
0.133, 0.860
Command:
biocham:
biocham: hide_molecules(APC).
biocham: plot.
biocham:

Condition of Entrainment in Period on Wee1/Cdc25



Entrainment in period constraint expressed in LTL with the period formula

Overview of the Tutorial

1. Introduction
   - Transposing programming concepts to the analysis of living processes
2. Rule-based modeling of biochemical systems
   - Syntax: molecules, reactions, regulations, SBML/SBGN Biocham notations
   - Semantics: Boolean, Differential and Stochastic interpretations of reactions
   - *Static analyses*: consistency, influence graph circuits, protein functions,&
   - Examples in cell signaling, gene expression, virus infection, cell cycle
3. Temporal Logic based formalization of biological properties
   - *Qualitative model-checking* in propositional Computation Tree Logic CTL
   - Quantitative model-checking in Linear Time Logic LTL(R)
   - *Parameter search* in high dimension w.r.t. LTL(R) specifications
   - *Robustness and sensitivity analyses* w.r.t. LTL(R) specifications
4. Conclusion