Méthodes Formelles pour la Biologie des Systèmes

François Fages

Project-Team Lifeware

http://lifeware.inria.fr/

Institut National de Recherche en Informatique et Automatique

Inria Saclay – Ile de France

Modeling Behaviours inTemporal Logic

Qualitative semantics: applies to very large systems without knowing reaction rates

How to query the asynchronous non-deterministic Boolean dynamics ?

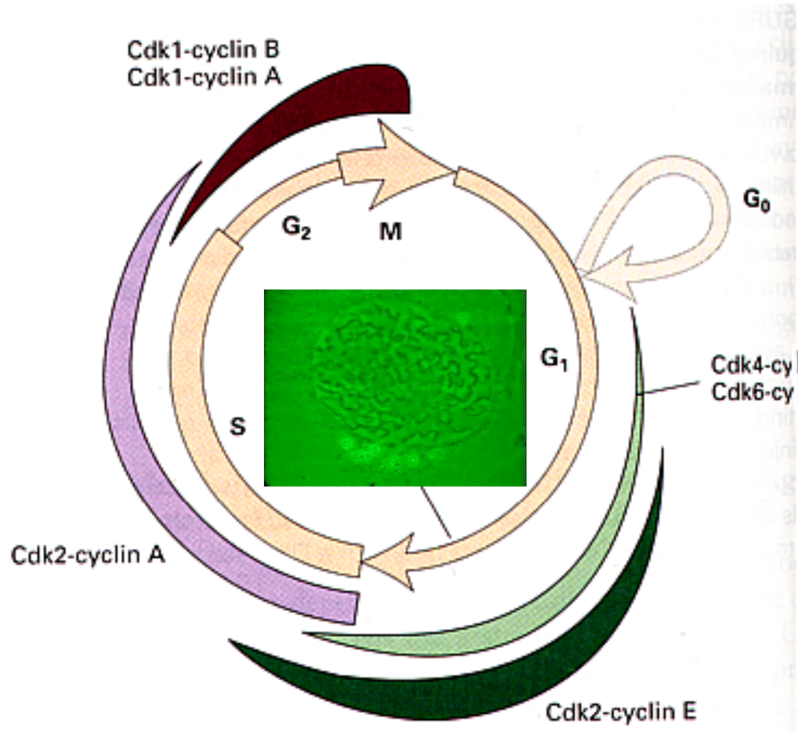
1. Example of Kohn's map of the mammalian cell cycle
2. Computation Tree Logic CTL query language
3. State-based model-checking algorithm
4. Symbolic model-checking algorithm
5. Model reduction preserving CTL properties

Quantitative semantics:

How to generalize to quantitative continuous/stochastic semantics ?

1. First-order FO-LTL(Rlin) on finite traces
2. FO-LTL(Rlin) constraint solving

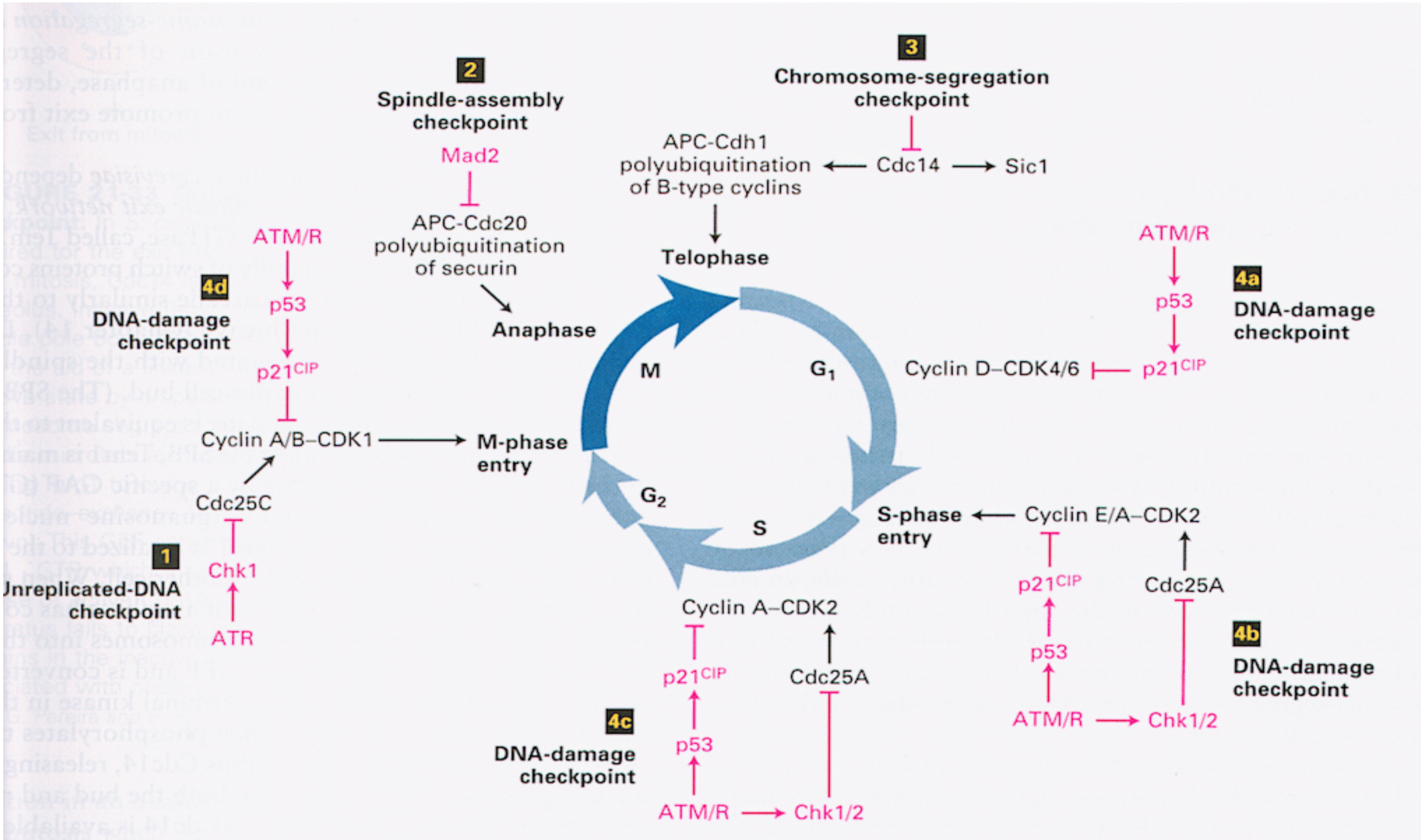
Cell Division Cycle
G0 → G1 → *Synthesis* → G2 → *Mitosis*



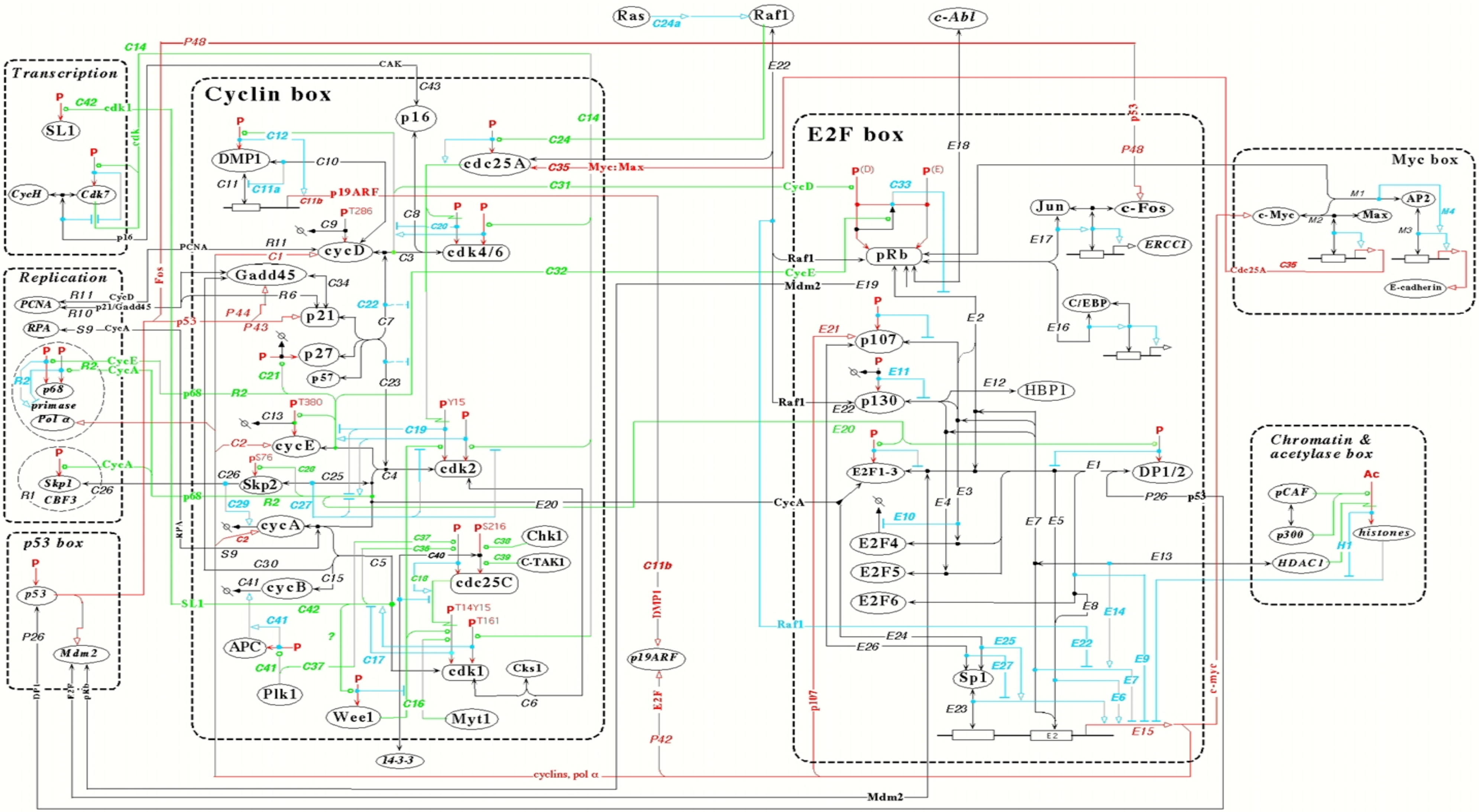
Sir Paul Nurse
Nobel prize 2001
for his work on Cyclins

G1: **CdK4-CycD** S: **Cdk2-CycA** G2,M: **Cdk1-CycA**
Cdk6-CycD **Cdk1-CycB** (MPF)
Cdk2-CycE

Cell Division Cycle Control

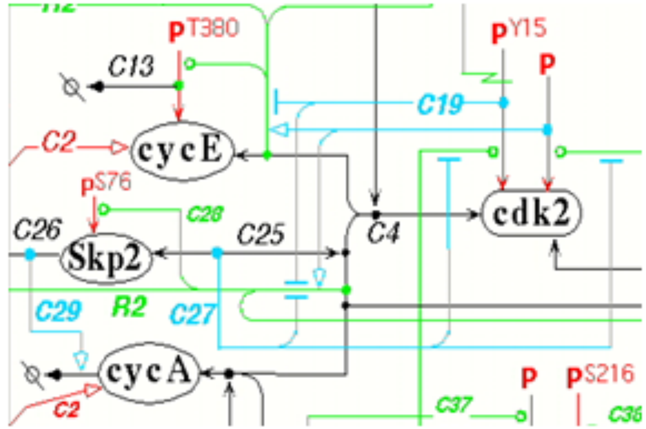


Mammalian Cell Cycle Control Map [Kohn 99]



Kohn's map detail for Cdk2

E.g. complexation of cdk2 with cycA and cycE



Kohn's map:

→ 732 reactions

→ 165 proteins and genes

→ 532 variables

How to analyze a transition system over 2^{532} states ? and $2^{2^{532}}$ sets of states ?

Symbolic model-checking

Represent a set of states by a Boolean constraint:

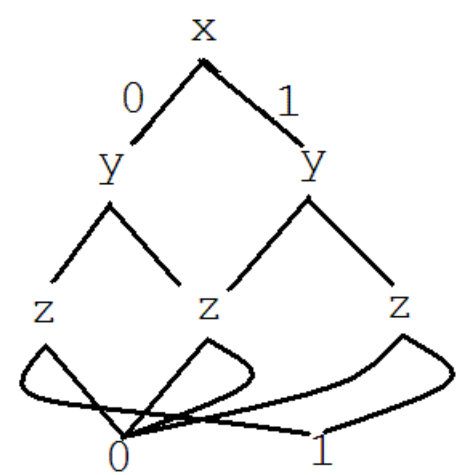
- *True*: full set of 2^{532} states,
- *False*: empty set,
- *M*: set of 2^{531} states where M is present,
- $M \vee \neg N$: set of 3.2^{530} states with M present or N absent
- etc.

Ordered Binary Decision Diagrams OBDD

Ordered Binary Decision Diagrams OBDD [Bryant 85] are decision graphs

- With fixed ordering of variables by levels
- And compressed in binary graphs with maximum sharing of common subtrees

Example: $(x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg x)$ OBDD(x,y,z)



$(x \vee \neg z) \wedge (z \vee \neg y) \wedge (y \vee \neg x)$ has the same OBDD(x,y,z) and is indeed equivalent

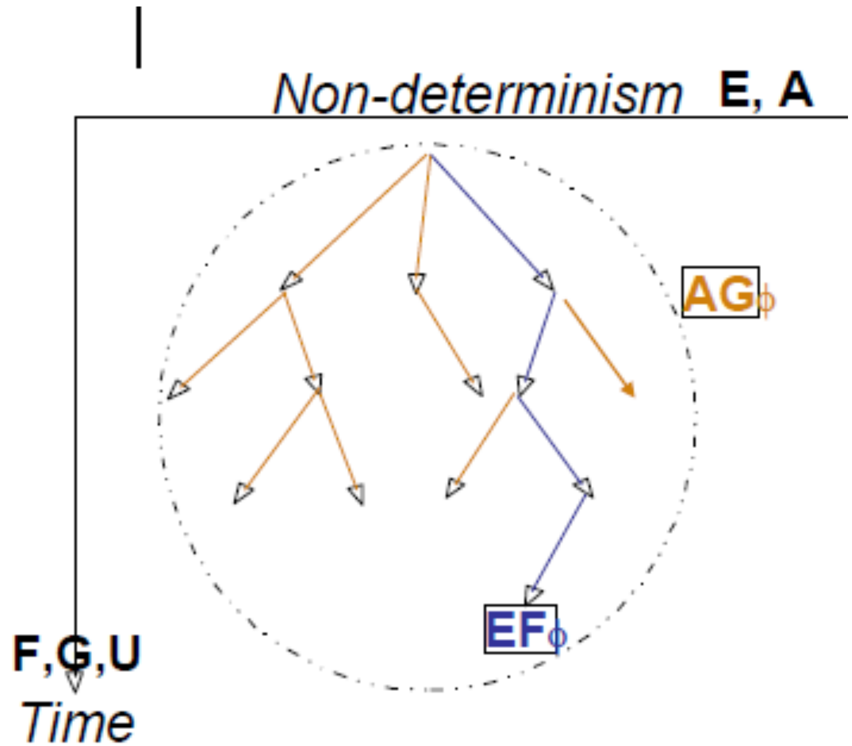
OBDD provide canonical forms for Boolean formulas

OBDD decide not only SAT in NP but also TAUT in co-NP

Computation Tree Logic CTL

Temporal logics extend classical logic with modal operators for time and non-determinism. 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)



Kohn’s Map Model-Checking

BIOCHAM NuSMV symbolic model-checker time in seconds [Chabrier Fages 2003 CMSB]

Initial state G2	Query:	Time:
	compiling	29
Reachability G1	**EF** CycE	2
Reachability G1	**EF** CycD	1.9
Checkpoint for mitosis complex	¬**E** **(**¬Cdc25~{Nterm} **U** Cdk1~{Thr161}-CycB**)**	2.2
Oscillations CycA	**EG ((EF** ¬ CycA**)** ∧ **(EF** CycA**))**	31.8
Osciallations CycB	**EG ((EF** ¬ CycB**)** ∧ **(EF** CycB**))** false in Kohn’s map ! (omission of CycB synthesis)	6

Kripke Semantics of CTL*

A Kripke structure K=(S,R) is a set S of states with a total relation $R \subseteq SxS$

The truth of a formula ϕ in a state s or on a path π of K is defined by:

- $s \models \phi$ if ϕ is a proposition true in s
- $s \models \mathbf{E}\ \phi$ if there is a path π starting from s such that $\pi \models \phi$
- $s \models \mathbf{A}\ \phi$ if for every path π starting from s such that $\pi \models \phi$
- $\pi \models \phi$ for a state formula ϕ if $s \models \phi$ where s is the first state of π
- $\pi \models \mathbf{X}\ \phi$ if $\pi^1 \models \phi$ where π^1 is the suffix of π without its first state
- $\pi \models \mathbf{F}\ \phi$ if $\exists\ k \geq 0$ such that $\pi^k \models \phi$ where π^k is the k^{th} suffix of π
- $\pi \models \mathbf{G}\ \phi$ if $\forall\ k \geq 0$, $\pi^k \models \phi$
- $\pi \models \phi_1\ \mathbf{U}\ \phi_2$ if $\exists\ k \geq 0\ \pi^k \models \phi_2 \wedge \forall\ j < k\ \pi^j \models \phi_1$
- $\pi \models \phi_1\ \mathbf{R}\ \phi_2$ if $\forall\ k \geq 0\ \pi^k \models \phi_2 \vee \exists\ j < k\ \pi^j \models \phi_1$

Duality: $\neg\ \mathbf{E}\phi = \mathbf{A} \neg\ \phi$, $\neg\ \mathbf{F}\phi = \mathbf{G} \neg\ \phi$, $\neg\ \mathbf{X}\phi = \mathbf{X} \neg\ \phi$, $\neg\ (\phi_1\ \mathbf{U}\ \phi_2\ ) = \neg\ \phi_1\ \mathbf{R} \neg\ \phi_2$

Minimal Set of CTL* Operators

- Logical connectives: $\vee$
  $\neg$
- Path quantifier: **E** "exists"
- Temporal operators: **X** "next"
  U "until"

Abbreviations (duality):

$\mathbf{A}\phi = \neg\, \mathbf{E} \neg\, \phi$	"always"
$\mathbf{G}\phi = \neg\, \mathbf{F} \neg\, \phi$	"globally"
$\phi_1\ \mathbf{R}\ \phi_2 = \neg\, (\neg\, \phi_1\ \mathbf{U} \neg\, \phi_2)$	"release"
$\mathbf{F}\phi = \text{true}\ \mathbf{U}\ \phi$	"finally"

CTL Fragment of CTL*

In CTL, each temporal operator must be preceded by a path quantifier

Any CTL formula is thus a state formula
and can be identified to the set of states which satisfy it

$\phi \simeq \{s \in S : s \models \phi\}$ [Emerson 90]

Basis of three operators: **EX, EG, EU**
others defined by duality:

- **EF** ϕ = **E**(true **U** ϕ)
- **AX** $\phi = \neg$ **EX** $\neg\ \phi$
- **AF** $\phi = \neg$ **EG** $\neg\ \phi$
- **AG** $\phi = \neg$ **EF** $\neg\ \phi$

LTL Fragment of CTL*

Linear Time Logic (LTL) formulae are of the form **A**ϕ
where ϕ contains no path quantifier, only temporal operators

Basis of two operators: **X, U**

- The LTL formula **A**(**FG** ϕ) is not expressible in CTL
  AF(AG *ϕ) is stronger, e.g. false on* $p \longrightarrow \neg p \longrightarrow p$

  AF(EG *ϕ) is weaker, e.g. true on* $p \longrightarrow \neg p$

- The CTL formula **EF**(**AG** ϕ) is not expressible in LTL

- LTL and CTL are strict fragments of CTL*

Basic CTL Model-Checking Algorithm

Dynamic programming algorithm for computing, in a *finite* Kripke structure K, the set of states satisfying a CTL formula: $\{s \in K : s \models \phi\}$.

Basic CTL Model-Checking Algorithm

Dynamic programming algorithm for computing, in a *finite* Kripke structure K, the set of states satisfying a CTL formula: $\{s \in K : s \models \phi\}$.

Represent K explicitly as a finite graph and iteratively label the nodes with the *subformulas of ϕ* that are true in that node:

- Add ϕ to the states satisfying ϕ
- Add **EF** ϕ (**EX** ϕ) to

Basic CTL Model-Checking Algorithm

Dynamic programming algorithm for computing, in a *finite* Kripke structure K, the set of states satisfying a CTL formula: $\{s \in K : s \models \phi\}$.

Represent K explicitly as a finite graph and iteratively label the nodes with the *subformulas of* ϕ that 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

Basic CTL Model-Checking Algorithm

Dynamic programming algorithm for computing, in a *finite* Kripke structure K, the set of states satisfying a CTL formula: $\{s \in K : s \models \phi\}$.

Represent K explicitly as a finite graph and iteratively label the nodes with the *subformulas of* ϕ that 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

Basic CTL Model-Checking Algorithm

Dynamic programming algorithm for computing, in a *finite* Kripke structure K, the set of states satisfying a CTL formula: $\{s \in K : s \models \phi\}$.

Represent K explicitly as a finite graph and iteratively label the nodes with the *subformulas of ϕ* that 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 of the subgraph satisfying ϕ which are on a path to a non trivial (i.e. containing at least one edge) strongly connected component.

Space and time in

Basic CTL Model-Checking Algorithm

Dynamic programming algorithm for computing, in a *finite* Kripke structure K, the set of states satisfying a CTL formula: $\{s \in K : s \models \phi\}$.

Represent K explicitly as a finite graph and iteratively label the nodes with the *subformulas of* ϕ that 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 of the subgraph satisfying ϕ which are on a path to a non trivial (i.e. containing at least one edge) strongly connected component.

Space and time in O(IKI*IϕI), CTL model-checking is Ptime-complete

Basic CTL Model-Checking Algorithm

Dynamic programming algorithm for computing, in a *finite* Kripke structure K, the set of states satisfying a CTL formula: $\{s \in K : s \vDash \phi\}$.

Represent K explicitly as a finite graph and iteratively label the nodes with the *subformulas of* ϕ that 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 of the subgraph satisfying ϕ which are on a path to a non trivial (i.e. containing at least one edge) strongly connected component.

Space and time in O(|K|*|ϕ|), CTL model-checking is Ptime-complete

Exercise: apply it to show `EG((EF ¬P) ^ (EF P))` on $P \longrightarrow \neg P$

Symbolic CTL Model-Checking Algorithm

- Represent a set of states by a boolean constraint c(V) over state variables V
  e.g. $p \vee \neg q$ represents the set of all states where p is present and q absent

- Represent the transition relation by a boolean constraint r(V,V')
  e.g. the constraint $p \vee (\neg p \wedge \neg p')$ represents the transition graph $p \to \neg p$

- Represent CTL operators by constraint transformers
  e.g. $[EX(c)] = \exists V'\, r(V,V') \wedge c[V'/V] \triangleq ex(c)$
  constraint of being one immediate predecessor r(V,V') of a state satisfying c(V')

  e.g. $[AX(c)] = \forall V'\, r(V,V') \Rightarrow c[V'/V] \triangleq ax(c)$
  constraint of having all successors r(V,V') satisfying c(V')

Logical Paradigm for Systems Biology

Use of model-checking algorithms [Lincoln et al. 02] [Chabrier Fages 03] [Bernot et al. 04]…

Biological process model = State Transition System K

Biological property = Temporal Logic Formula φ

Model validation = model-checking K, s $\models ?$ φ

Model reduction = model-checking K'?$\subset$K K', s $\models \varphi$

Static experiment design = model-checking K, s? $\models \varphi$

Model functions = true formulae enumeration K, s $\models \varphi$?

Model Inference, dyn. exp. design = constraint solving K?, s? $\models \varphi$

Generalizations to quantitative temporal logics

- FO-LTL(R_{lin}) [Rizk, Batt, F, Soliman 09] MTL [Donze Maler 12] parameter search, robustness
- SAT modulo ODE [Gao Clarke 2012] formal verification on parameter range
- Continuous CRN design K?, s? $\models$ reachable(stable(y $\approx \frac{x^4}{c+x^4}$)

TD8 MAPK Signalling

http://lifeware.inria.fr/biocham4/online/

In [12]:
```
check_ctl(query:checkpoint2(PP_KK,PP_K)).
```

Out[12]:
```
checkpoint2(PP_KK,PP_K) is true
```

In [13]:
```
check_ctl(query:checkpoint2(PP_KK_KKPase,PP_K)).
```

Out[13]:
```
checkpoint2(PP_KK_KKPase,PP_K) is false
```

In [14]:
```
check_ctl(query:checkpoint2(PP_KK_KKPase,PP_K), nusmv_counter_example:yes).
```

Out[14]:
```
Trace:
E1       E1_KKK  E2       E2_P_KKK         K        KK       KKK      KKPase   KKPase_PP_KK     KKPas
e_P_KK   KPase   KPase_PP_K       KPase_P_K        PP_KK    PP_KK_K  PP_KK_P_K         P_K      P_KK
         P_KKK   P_KKK_KK         P_KKK_P_KK       PP_K     PP_KK_KKPase
TRUE     FALSE   TRUE     FALSE   TRUE     TRUE     TRUE     TRUE     FALSE    FALSE    TRUE     FALSE
         FALSE   FALSE    FALSE   FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE

checkpoint2(PP_KK_KKPase,PP_K) is false
```

In [15]:
```
reduce_model.
```

Out[15]:
```
removed [(r1a__d1*'E1_KKK'for'E1_KKK'=>'KKK'+'E1'),(r2a__d2*'E2_P_KKK'for'E2_P_KKK'=>'P_KK
K'+'E2'),(r3a__d3*'P_KKK_KK'for'P_KKK_KK'=>'KK'+'P_KKK'),(r4a__d4*'KKPase_P_KK'for'KKPase_P_K
K'=>'P_KK'+'KKPase'),(r5a__d5*'P_KKK_P_KK'for'P_KKK_P_KK'=>'P_KK'+'P_KKK'),(r6a__d6*'KKPase_P
P_KK'for'KKPase_PP_KK'=>'PP_KK'+'KKPase'),(r7a__d7*'PP_KK_K'for'PP_KK_K'=>'K'+'PP_KK'),(r8a__
d8*'KPase_P_K'for'KPase_P_K'=>'P_K'+'KPase'),(r9a__d9*'PP_KK_P_K'for'PP_KK_P_K'=>'P_K'+'PP_K
K'),(r10a__d10*'KPase_PP_K'for'KPase_PP_K'=>'PP_K'+'KPase')]
```