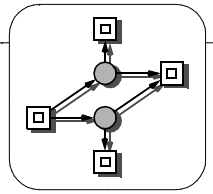
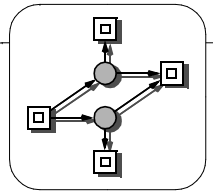


(INFORMAL) INTRODUCTION INTO TEMPORAL LOGICS



QUALITATIVE ANALYSIS METHODS, OVERVIEW

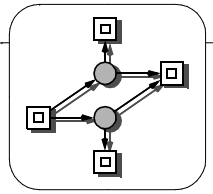
- NET REDUCTION
- STRUCTURAL PROPERTIES
- LINEAR PROGRAMMING
  - place / transition invariants
  - state equation
  - trap equation

static analysis

- REACHABILITY ANALYSIS
  - (complete) reachability graph
  - compressed state spaces
    - BDDs, NDDs, ..., xDDs
    - Kronecker products
  - reduced state spaces
    - coverability graph
    - symmetry
    - stubborn sets
  - branching process

dynamic analysis

(model checking)



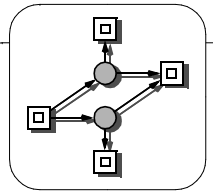
MOTIVATION (1)

- up to now, mainly:
  GENERAL PROPERTIES (REQUIREMENTS)
  -> *must be valid for any system, independently of its special functionality*

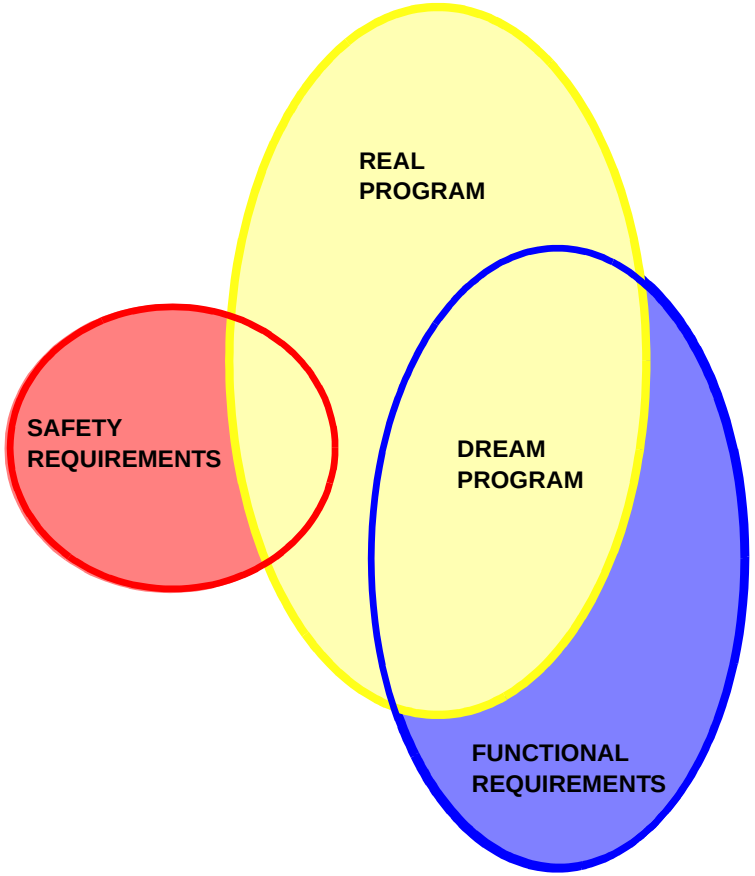
  -> *boundedness*
  -> *liveness*
  -> *reversibility*
  -> *. . .*

- moreover, there are
  SPECIAL PROPERTIES (REQUIREMENTS)
  -> *reflect the special functionality*

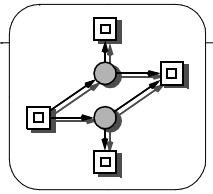
  -> *safety properties* *(s)*
  "something bad never happens"
  -> *progress properties* *(p)*
  "something good will happen finally"
  -> *consistency properties* *(c)*
  to check the model's integrity
  -> *insights into system behaviour* *(i)*
  -> *. . .*



SAFETY & FUNCTIONAL REQUIREMENTS

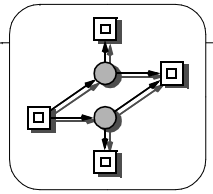


functional requirements =
progress properties + consistency properties



SPECIAL PROPERTIES, EXAMPLES

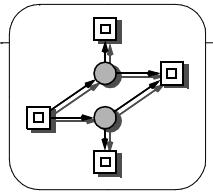
- botanical garden
  - -> *which counter values are printed ?* *(i)*
- mutex algorithms
  - -> *there is never more than one process in its mutex region* *(s)*
- production cell
  - -> *the press will only be closed, when no robot arm is positioned inside it* *(s)*
  - -> *any plate at the input position will finally reach the cell's output position* *(p)*
  - -> *the swivel is always positioned at exactly one angle* *(c)*
  - -> *both robot arms hold a plate simultaneously* *(i)*
- concurrent pushers
  - -> *to avoid machine collisions, two adjacent pushers will never be moved at the same time* *(s)*
  - -> *at any time, a pusher can be driven in one direction only* *(c)*



MOTIVATION (2)

- some properties can be expressed as un-/reachability of a given system state
  - -> *safety property* -> *unreachability*
  - -> *(weak) progress property* -> *reachability*
- BUT, system states tend to be lengthy
  - -> *hard to read*
  - -> *error-prone specification*
- most properties cannot be expressed as un-/reachability of a given system state
- wanted: general query language to specify any special property
  - -> *flexible*
  - -> *readable*

  } ***temporal logics***
- size of the state space calls for automatic evaluation techniques for any query
  - -> ***model checking***

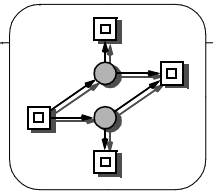


BEHAVIOURAL NET PROPERTIES

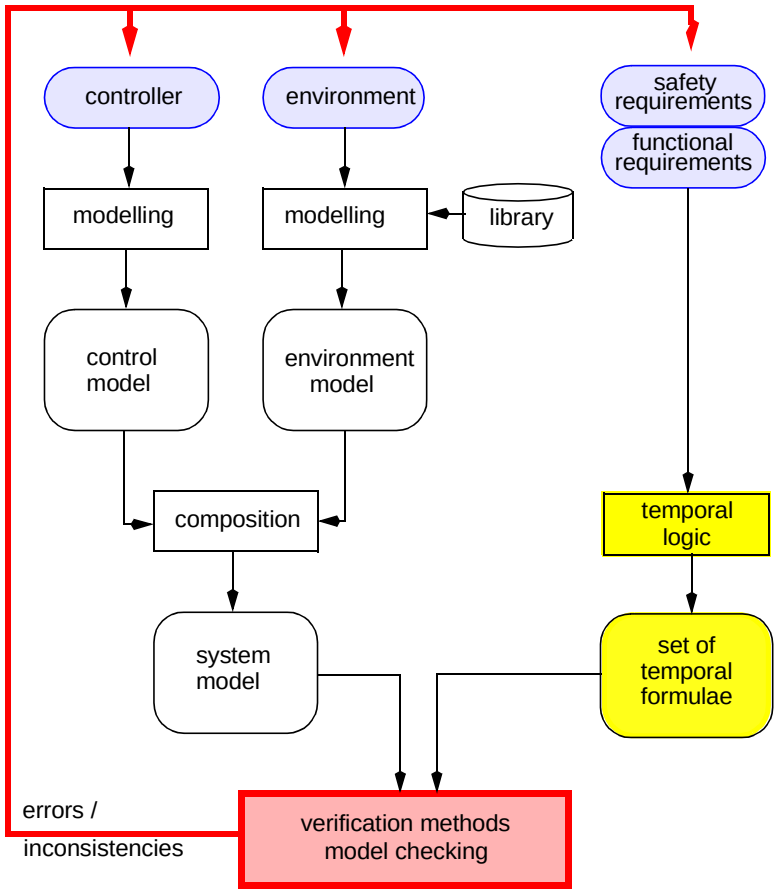
- MARKABILITY of places
  - markable *(place liveness)*
  - k-bounded *(safe)*
- LIVENESS of transitions
  - zero times firing *(m_0-dead)*
  - finite times firing *(dead, non-live)*
  - infinite times (probably) firing *(live)*
  - infinite times (definitely) firing *(livelock free)*
- REACHABILITY of states
  - dead states
  - reproducibility
  - reversibility *(m_0 - home state)*
- bad states *(facts)*
- user-specified states
- NET INVARIANTS
  - transition invariants
  - place invariants
- temporal relationship of logic formulae

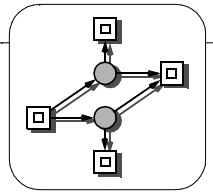
general properties

special properties



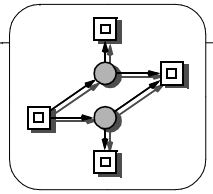
MODEL BASED SYSTEM VALIDATION, PROCESS AND TOOL





MODEL CHECKING, BASIC INGREDIENTS

- **a language to model the system**
  - -> *formal semantics*
  - -> *many options, e.g. Petri nets*
- **a language to specify model properties**
  - -> *temporal Logics,*
  - -> *several options, e.g.*
    Computational Tree Logic (CTL)
- **an analysis approach to check a model against its properties**
  - -> *model checking,*
  - -> *various approaches (algorithms + data structures), e.g.*
    using reachability graph (RG)
    - = *labelled state transition system (STS)*
    - = *Kripke structure*
    - ≈ *Continuous Time Markov Chain (CTMC)*

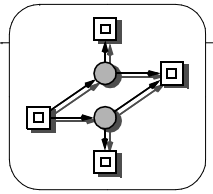


MODEL CHECKING, 2007

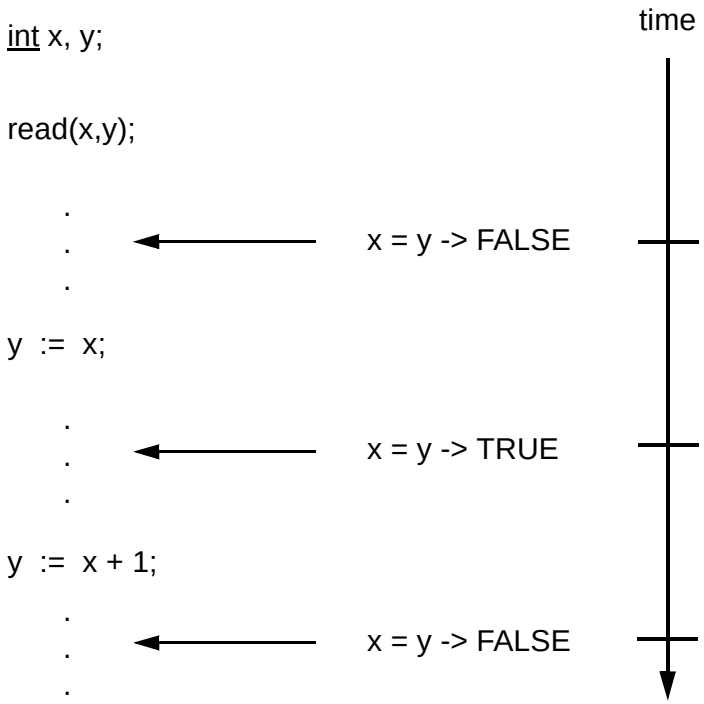
- ACM Turing Award
  = Nobel Prize of computing

  goes to
  Model Checking Pioneers
  Edmund M. Clarke,
  E. Allen Emerson and Joseph Sifakis

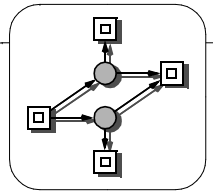
"[for their roles] in developing Model Checking into a highly effective verification technology, widely adopted in the hardware and software industries."



TEMPORAL LOGICS, INTRODUCTION

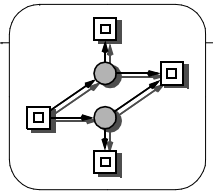


How to specify logical statements with respect to the execution of a system ?

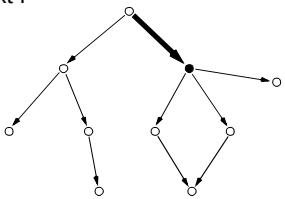
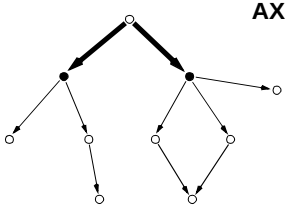
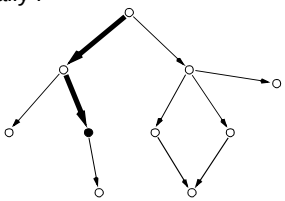
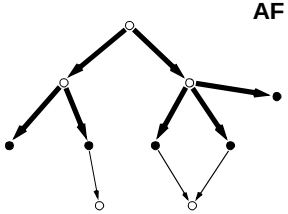
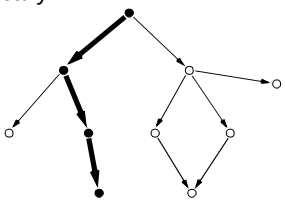
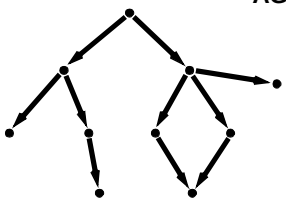
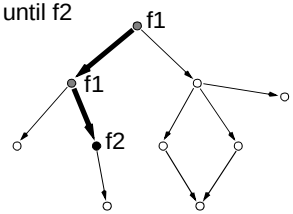
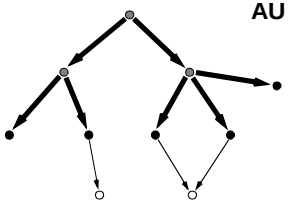


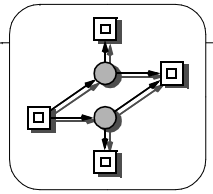
TEMPORAL LOGICS, BASICS

- extension of classical (propositional/predicate) logics by temporal operators
- constants
  - -> *TRUE, FALSE*
- atomic propositions
  - -> *elementary statements, having - in a given state - a well-defined truth value*
  - -> *e.g. mutex , for 1-bounded pn*
  - -> *e.g. buffer = 2, buffer > 2 , else*
- classical Boolean operators
  - -> *negation* !
  - -> *conjunction* *
  - -> *disjunction* +
  - -> *implication* ->
- temporal operators
  - -> *to refer to the sequence of states along the execution of a system*

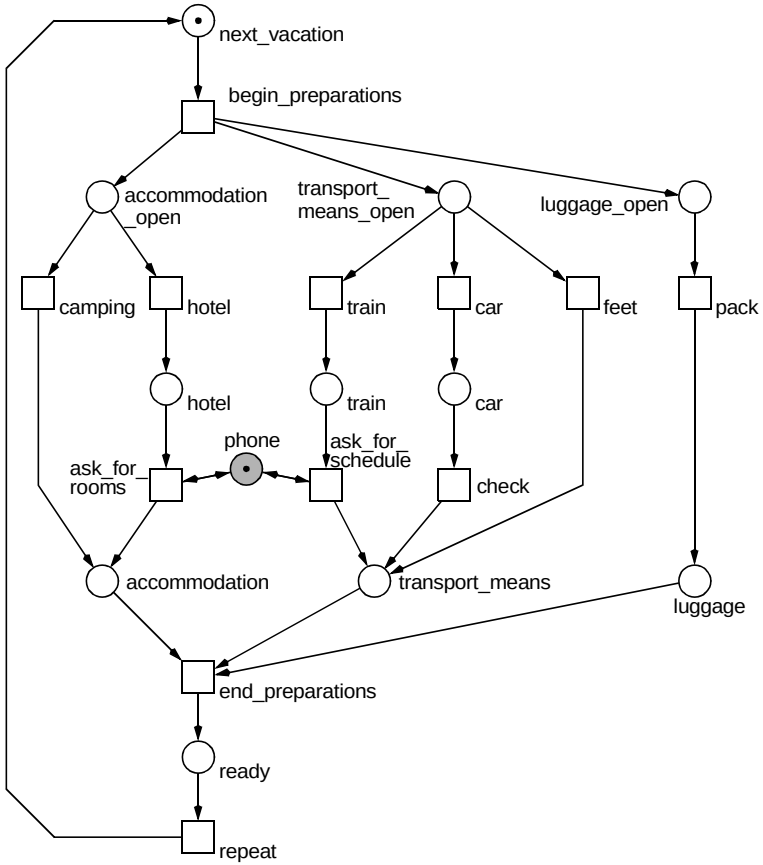


CTL OPERATORS, INTERLEAVING SEMANTICS

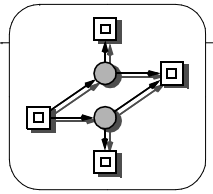
	on some branch	on all branches
next f	**EX** 	**AX** 
finally f	**EF** 	**AF** 
globally f	**EG** 	**AG** 
f1 until f2	**EU** 	**AU** 



EXAMPLE, TRAVEL PREPARATION



HINT: avoid special characters in node names !



SOME SIMPLE PROPERTIES (1)

// (1) this is a possible combination of choices

EF (hotel * car); **-> TRUE**

// (2) this is the only possible combination of choices

AF (hotel * car); **-> FALSE**

// (3) this is an impossible combination of choices

! **EF** (train * car); **-> TRUE**

// (4) any travel planning will be finished finally

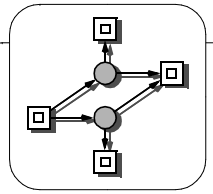
AG (next_vacation -> **AF** (ready)); **-> TRUE**

// (5) atomic use of the phone only,
the phone is never reserved over several steps

AG (phone); **-> TRUE**

// (6) liveness of T end_preparations **-> TRUE**

AG (**EF** (accommodation * transport_means * luggage));



SOME SIMPLE PROPERTIES (2)

// (7) livelock freedom/progress of T end_preparations
-> TRUE

AG (**AF** (accommodation * transport_means * luggage));

// (8) **-> FALSE**

AG (luggage_open -> **AX** (luggage));

// (9) **-> TRUE**

AG (luggage_open -> **EX** (luggage));

// (10) **-> TRUE**

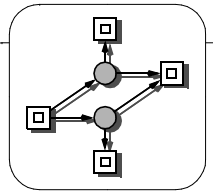
AG (luggage_open -> **AF** (luggage));

// (11) **-> FALSE**

A (luggage_open **U** luggage);

// (12) **-> TRUE**

AG (luggage_open -> **A** (luggage_open **U** luggage));



TEMPORAL LOGICS, TYPICAL QUESTIONS

- **reachability-related**

  EF *(φ)*

  There exists at least one computation path to reach eventually a state, where φ will be true.

- **safety-related**

  AG *(! φ)* *-> equivalent to ! **EF** (φ)*

  For every computation path, φ will never be true.

- **invariant-related**

  AG *(φ)* *-> equivalent to ! **EF** (! φ)*

  For every computation path, φ will be true for ever.

- **liveness-related**

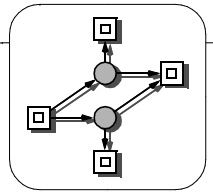
  AG *(**EF** (φ))*

  What ever happens, there exists the chance (at least one path) that φ will be true again.

- **progress-related**

  AG *(**AF** (φ))*

  For every computation path, φ will eventually be true.



(ANOTHER) PROPERTY TAXONOMY

- **fatal errors**

  e.g. safety properties

  If a robot arm is loaded, its magnet is not deactivated until the robot is in its unloading position.

  AG *(φ -> **A** (!arm1_mag_off **U** ψ)), where*

  φ = arm1_mag_on
  ** arm1_pickup_angle*
  ** arm1_pickup_ext*

  *ψ = arm1_release_angle * arm1_release_ext*

- **warnings**

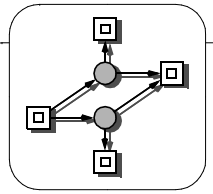
  e.g. liveness

  AG *(**EF** enabled (t))* *for each transition t*

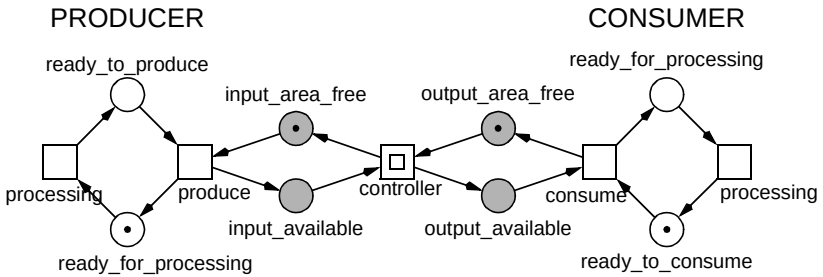
- **insights**

  Is it possible, that both robot arms carry a plate at the same time?

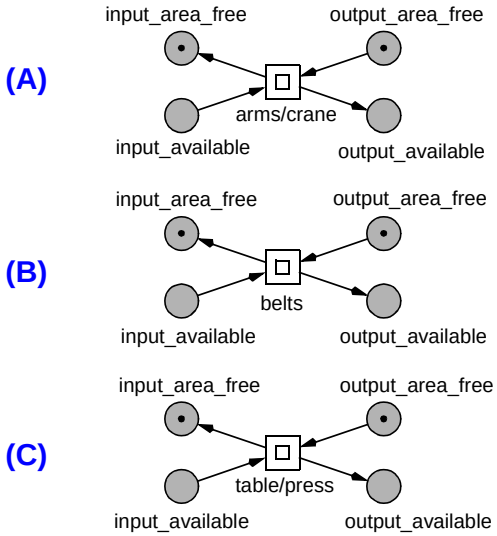
  EF *(arm1_mag_on * arm2_mag_on)*

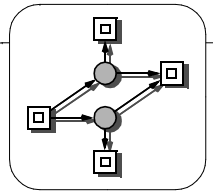


PRODUCTION CELL, PROCON PATTERNS

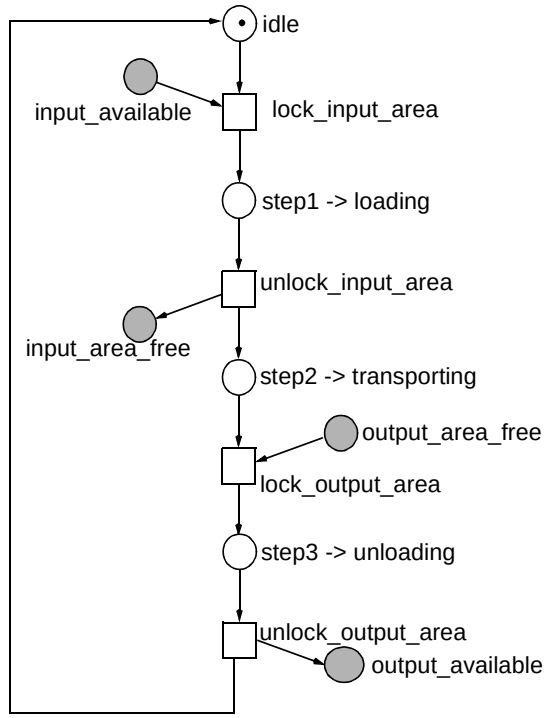


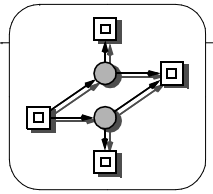
THREE TYPES OF COOPERATION PATTERN





(A) INDEPENDENT INPUT/OUTPUT

- arms/crane:
  step-wise synchronization
  with only one of the adjacent controllers
- pattern property, e.g.
  AG *(step1 -> ! (input_available + input_area_free))*
- 

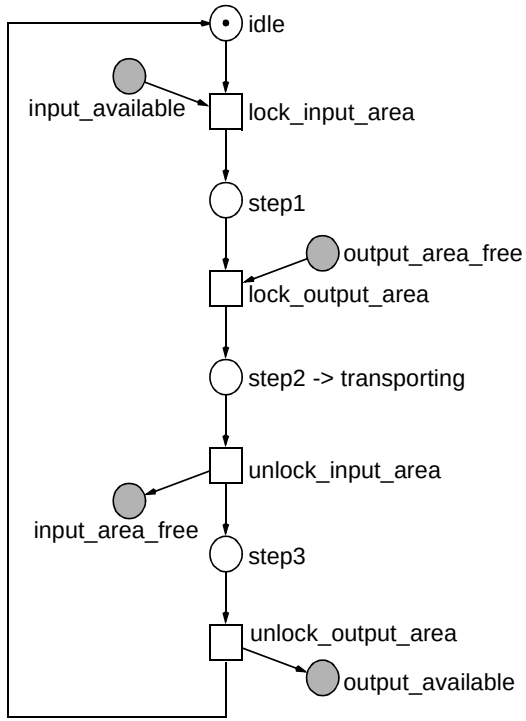


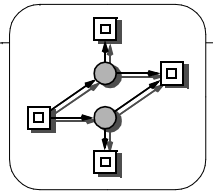
(B) DEPENDENT INPUT/OUTPUT

- belts:
  simultaneous control of input and output region

- pattern property

  AG *(step2 -> ! (input_available + input_area_free + output_area_free + output_available))*

- 

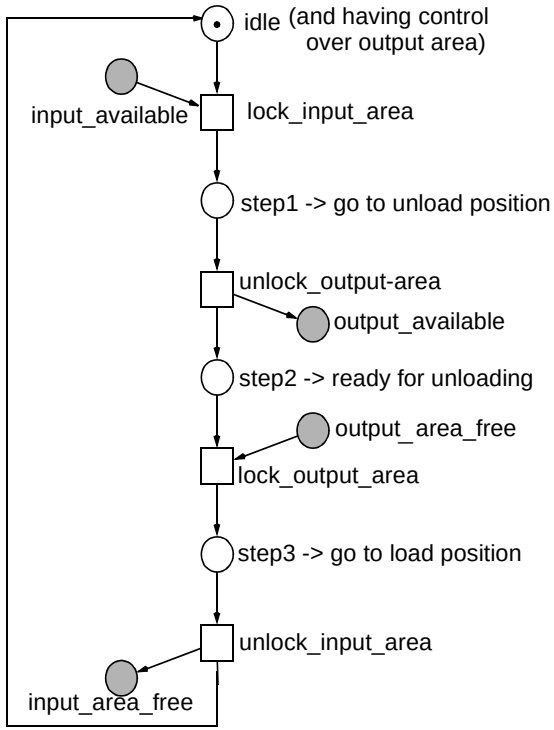


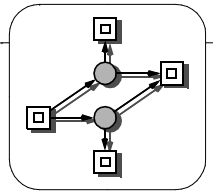
(C) MUTUALLY EXCLUSIVE INPUT/OUTPUT

- table/press:
  the controller must always hold a lock on one of its cooperation regions;

- pattern property

  AG *(! (input_available + input_area_free) + ! (output_available + output_area_free))*

- 



CONCURRENT PUSHERS, TYPICAL SAFETY PROPERTIES

- At any time, a pusher can be driven in one direction only.

$$\mathbf{AG}\ (\neg(Pi\_R1\_on \wedge Pi\_R2\_on)),\ \forall i$$

- To avoid collisions, it is not allowed to move adjacent pushers at the same time.

$$\left(\sum_{i=1}^{2} Pj\_Ri\_on + \sum_{i=1}^{2} Pk\_Ri\_on \right) \leq 1$$

$$\forall j, k : j + 1 = k$$

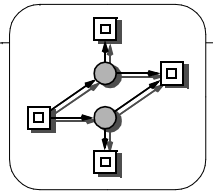
- While moving a pusher, a new work piece must not arrive in its input position.

$$\mathbf{AG}\ (posi\_full \rightarrow Pi\_basic),\ \forall i$$

- No pusher must be driven too far / near.

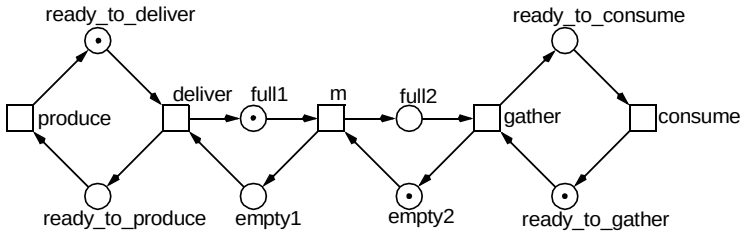
$$\mathbf{AG}\ (\neg Pi\_too\_near),\ \forall i \quad \mathbf{AG}\ (\neg Pi\_too\_far),\ \forall i$$

- **generic properties**
  -> have to be expanded for analysis



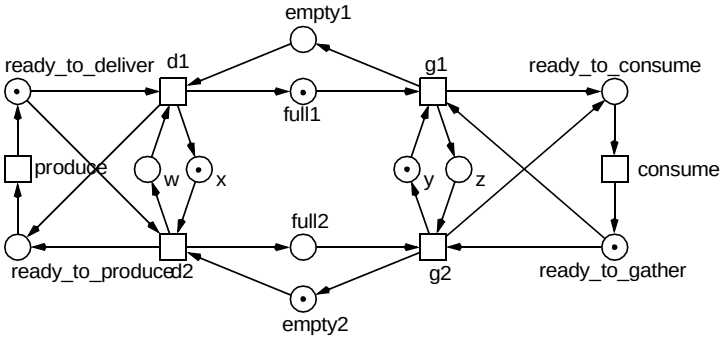
PROCON, TWO VERSIONS (1)

- **sequential buffer of capacity 2**

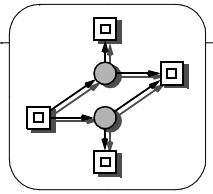


there is one empty buffer space,
but the buffer needs first some self-organization
before accepting a new item

- **parallel buffer of capacity 2**



always:
if ready_to_deliver & there is empty buffer space
then the buffer is ready immediately to accept a new item



PROCON, TWO VERSIONS (2)

- how to express the essential difference in temporal logics ?

- parallel buffer:
  if ready_to_deliver & there is empty buffer space
  then the buffer is ready immediately to accept a new item

  AG *(ready_to_deliver * (empty1 + empty2)*
  *-> **AX** ready_to_produce)*

  -> *TRUE for parallel buffer* *(should be)*

  -> *FALSE for sequential buffer*

- BUT: RG - interleaving semantics

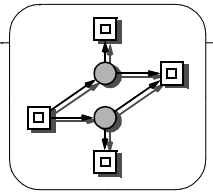
  -> *formula is FALSE for both systems*

- the difference in the behaviour of both systems

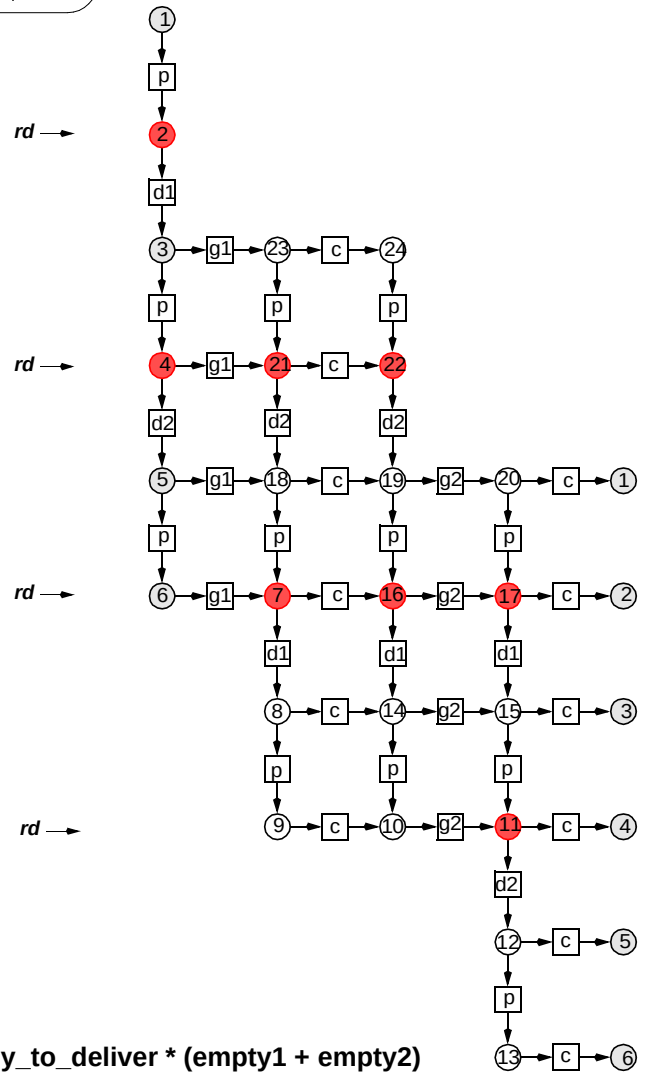
  -> *concurrent versus sequential behaviour*

  -> *needs true concurrency (partial order) semantics, too*

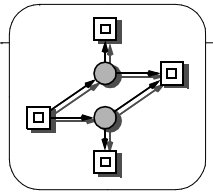
- there are MC tools for true concurrency semantics, but none of them can handle the **X**-operator



PROCON, PARALLEL BUFFER



ready_to_deliver * (empty1 + empty2)



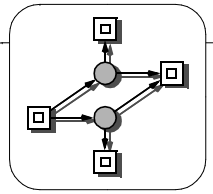
LTL MODEL CHECKING

- LTL - Linear Time Temporal Logic
  - -> *a formula must be true for all (linear) execution sequences*
  - -> ***X***, ***F***, ***G***, ***U***
- rule of the thumb
  - -> *linear OP ~ branching "on all branch" OP*

  ***BUT**: CTL =//= LTL*
- LTL\X model checking can be combined with stubborn set reduction
  - -> *on-the-fly model checking*
- two sets
  - -> *visible transitions of a formula:*

    all pre- and posttransitions of places appearing in the formula
  - -> *invisible transitions of a formula:*

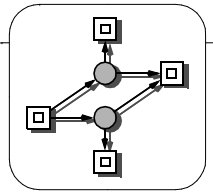
    all other transitions
- a formula-conform stubborn set contains only invisible transitions or all enable transitions
- very successful for "local" formulae



PRODUCTION CELL ON-THE-FLY LTL MODEL CHECKING

requirement formula	# states generated	time effort
8 a	2259	4.16'
8 b	1775	3.76'
9	2305	4.34'
10	1879	3.10'
15	1184	2.55'
29	704	2.16'
30	703	2.46'
31 a	27104	23.80'
31 b	6433	4.02'
31 c	28285	24.30'
32 a	3940	10.26'
32 b	3113	9.21'

control model, 5 plates, full state space: 1.657.242
-> [BTU Report I-08/1995]



LTL PROPERTIES, EXAMPLES

- requirement 30 -> 703 states

  The swivel is always positioned at exactly one angle.

  G *(arm1_pick_up_angle*
  xor arm1_release_angle
  xor arm2_pick_up_angle
  xor arm2_release_angle)

- requirement 29 -> 704 states

  The swivel is always either stopped or moves in exactly one direction.

  G *(robot_stop xor robot_left xor robot_right)*

- requirement 15 -> 1184 states

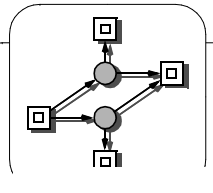
  The feed belt may only convey a blank through its light barrier, if the table is in its loading position.

  G *(belt1_light_barrier_true ->*
  *(table_load_angle * table_bottom_pos))*

- requirement 10 -> 1879 states

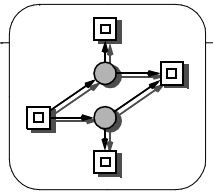
  The travelling crane is not allowed to knock against a belt laterally, i. e. the crane never moves without performing a vertically translation.

  G *((crane_to_belt1 + crane_to_belt2) ->*
  crane_at_transport_height)

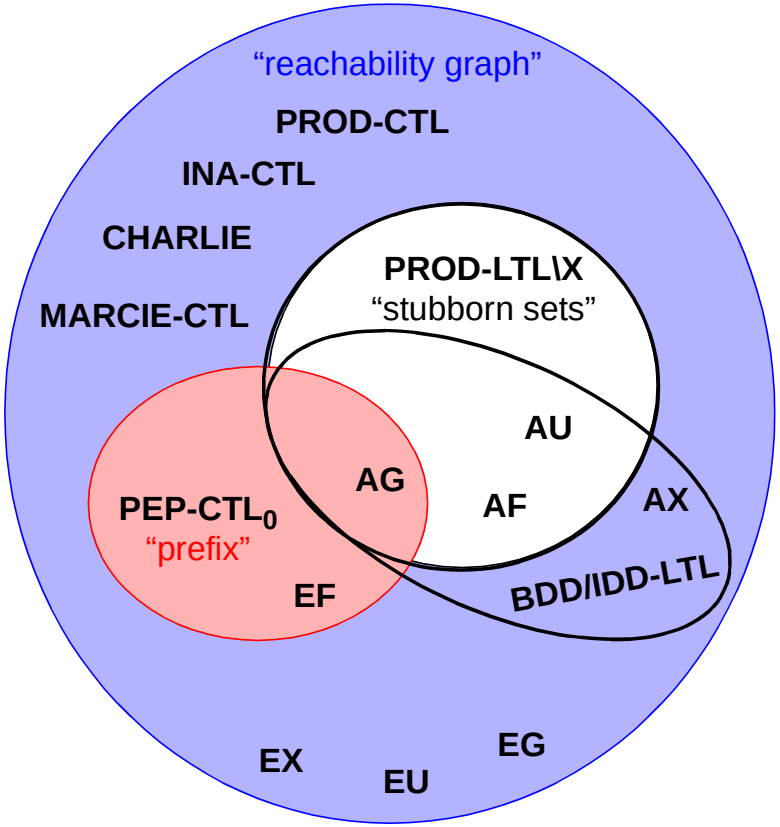


TEMPORAL LOGICS, OVERVIEW

semantics / time	interleaving	partial order
linear (LTL)	**traces** (no conflict, no concurrency) Manna & Pnueli, Kröger, jsp 2001 *BDD/IDD-LTL*	**runs** (no conflict, but concurrency) Reisig *tools: ?*
branching (CTL)	**reachability graph** (conflict & concurrency not distinguishable) Emmerson, Clarke *PROD, INA,* **Charlie** **BDD/IDD-Marcie**	**prefix** (conflicts & concurrency) McMillan, Esparza, pd 2001 *PEP*

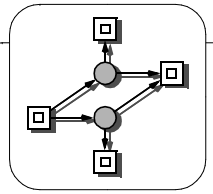


TEMPORAL LOGICS, SOME TOOLS



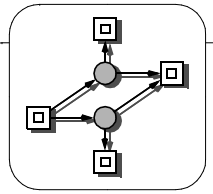
rule of the thumb:

linear OPERATOR
~ branching „OnAllBranch“ OPERATOR



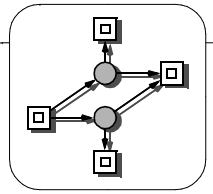
MODEL CHECKING TECHNIQUES, OVERVIEW

technique	CTL	LTL
RG	INA, **Charlie**	PROD, MARIA, Charlie
stubborn reduced RG	LoLA	PROD (LTL\X)
symmetrically reduced RG	LoLA (sym. formulae)	?
BDD / IDD	SMART, **Marcie**	*BDD-LTL, IDD-LTL*
Kronecker	[Kemper]	?
prefix	PEP (CTL_0)	QQ (LTL\X)
process automata	[pd]	?
invariants state equation	INA (not EF f)	INA (G not f)



SUMMARY

- correct model checking
  requires correct requirement specifications
  - -> *take your time*
  - -> *think twice*
- model checking proves consistency of
  - -> *model & specification*
- if the answer is NO (-> FALSE),
  - -> *the model can be wrong*
  - -> *the specification can be wrong*
  - -> *or both*
- if the answer is YES (-> TRUE),
  - -> *model & specification*
    can still contain (same) logical faults
  - -> *unlikely !?*
- (up to now) restricted to bounded systems
  - -> *numerous ongoing research*
  - -> *CTL - undecidable*
  - -> *LTL - decidable, but no tools (not yet ?)*
  - -> *unboundedness + inhibitor arcs = undecidability*



REFERENCES, METHODS

[Bérard 2001]
B BÉRARD ET AL.:
System and Software Verification, Model Checking Techniques and Tools;
Springer 2001.

[Clarke 1986]
EM CLARKE, EA EMERSON, AP SISTLA:
Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications;
ACM Trans. on Programming Languages and Systems 8(86)2, pp. 244-263.

[Clarke 2001]
EM CLARKE, O GRUMBERG, DA PELED:
Model Checking;
MIT Press 1999, third printing 2001.

[Emerson 1981]
EA EMERSON:
BRANCHING TIME TEMPORAL LOGIC AND THE DESIGN OF CORRECT CONCURRENT PROGRAMS.
PhD thesis, Harvard Univ. 1981.

[Emerson 1990]
EA EMERSON:
Temporal and Modal Logic;
in: J. v. Leeuwen, ed.: Handbook of Theoretical Computer Science, Vol. B; Elsivier, Amsterdam 1990, 995-1072.

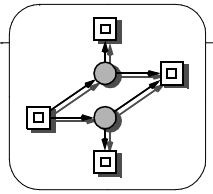
[Esparza 1994]
J ESPARZA:
Model Checking Using Net Unfoldings;
Science of Computer Programming, 23(1994), 151-195.

[MacMillan 1992]
KL MACMILLAN:
Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits;
Proc. of the 4th Workshop on Computer Aided Verification, Montreal 1992, 164-174.

[Manna 1992]
Z MANNA, A PNUELI:
The Temporal Logic of Reactive and Concurrent Systems - Specification;
Springer 1992.

[Kröger 1987]
F KRÖGER:
Temporal Logic of Programs;
Spinger 1987.

[Pnueli 1977]
A PNUELI:
The Temporal Logic of Concurrent Programs;
18th. IEEE Symp. on Foundation of CS, 46-57, IEEE Computer Society Press 1977.



REFERENCES, TOOLS

[Spranger 2001]
J SPRANGER:
Symbolic LTL verification of Petri nets (in German);
PhD thesis, BTU Cottbus, Dep. of CS, 2001.

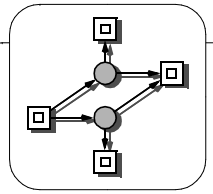
[Schwarick 2006]
M SCHWARICK:
A Tool to Analyse Petri Net Models (in German);
Master thesis, BTU Cottbus, Dep. of CS, 2006.

[Tovchigrechko 2008]
A TOVCHIGRECHKO:
Model checking using interval decision diagrams;
PhD thesis, BTU Cottbus, Dep. of CS, 2008.

[Franzke 2009]
A FRANZKE:
Charlie 2.0 - a multi-threaded Petri net analyzer;
Master thesis, BTU Cottbus, Dep. of CS, 2009.

[Heiner 2013]
M HEINER, C ROHR AND M SCHWARICK:
MARCIE - Model checking And Reachability analysis done effiCIEntly;
Proc. PETRI NETS 2013, Milano, LNCS 7927, 389–399, 2013.

[Heiner 2016]
M HEINER, C ROHR, M SCHWARICK AND A TOVCHIGRECHKO:
MARCIE's secrets of efficient model checking;
Trans. on Petri Nets and Other Models of Concurrency XI, LNCS 9930, 286-296, 2016.



REFERENCES, SOME CASE STUDIES

[MCC]
MODEL CHECKING CONTEST;
a satellite event of PETRI NETS conference series;
https://mcc.lip6.fr

[Heiner 1999]
M HEINER, P DEUSSEN, J SPRANGER:
A Case Study in Design and Verification of Manufacturing Systems with Hierarchical Petri Nets;
Journal of Advanced Manufacturing Technology (1999), 15, pp. 139-152.

[Heiner 2008]
M HEINER, D GILBERT, R DONALDSON:
Petri Nets for Systems and Synthetic Biology;
SFM 2008, Springer LNCS 5016, pp. 215-264, 2008.

[Heiner 2009]
M HEINER, S LEHRACK, D GILBERT, W MARWAN:
Extended Stochastic Petri Nets for Model-based Design of Wetlab Experiments;
Trans. on Computational Systems Biology XI, Springer LNCS/LNBI 5750, pp. 138-163, 2009.

[Heiner 2010]
M HEINER; R DONALDSON; D GILBERT:
Petri Nets for Systems Biology;
in MS Iyengar (ed.): Symbolic Systems Biology: Theory and Methods, Jones & Bartlett Publishers, LLC, 2010.

[Blätke 2015]
MA BLÄTKE, M HEINER AND W MARWAN:
BioModel Engineering with Petri Nets;
In Algebraic and Discrete Mathematical Methods for Modern Biology, (R Robeva, Ed.), Elsevier Inc., pages 141–193, March 2015.