Model Checking of Concurrent Systems - Part II -

Monika Heiner

**BTU Cottbus,
Computer Science Institute**

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)

- **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 verificatio technology, widely adopted in the hardware and software industries."

2007

Temporal Logics,

CTL - a crash cours

MOTIVATION (1)

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

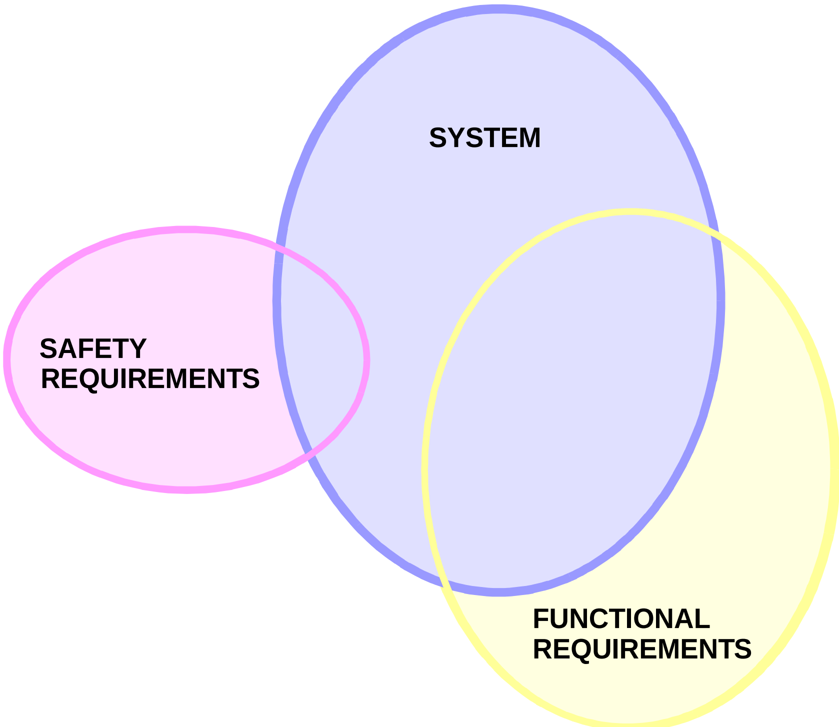
  - -> *boundedness*
  - -> *liveness*
  - -> *reversibility*
  - *. . .*

- SPECIAL PROPERTIES (REQUIREMENTS)
  - -> *reflect the special functionality*

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

  } ***functional requirements*** (applies to (c) and (p))

SAFETY VERSUS FUNCTIONAL REQUIREMENTS



Special Properties, Examples

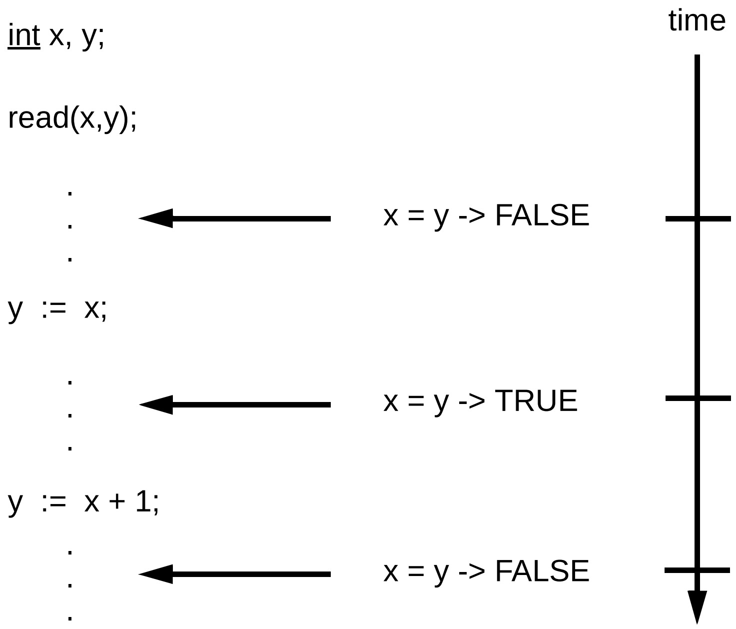
- **insights (i)**
  - -> *Is it possible that both robot arms hold a plate simultaneously ?*
  - -> *How many plates can be concurrently inside the system ?*
- **consistency (c)**
  - -> *The robot swivel is always positioned at exactly one angle.*
  - -> *At any time, a robot arm can be driven in one direction only.*
- **progress (p)**
  - -> *Any plate at the input position will finally reach the cell's output position.*
- **safety (s)**
  - -> *To avoid machine collisions,*
    the robot will only rotate with arms retracted.
  - -> *The press will only be closed, when no robot arm is positioned inside.*
  - -> *The feed belt may only convey a blank through its light barrier,*
    if the table is in its loading position.

Motivation (2)

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

  } ***temporal logics***

TEMPORAL LOGICS, INTRODUCTION



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

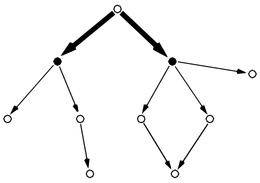
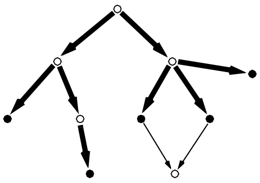
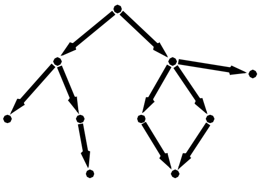
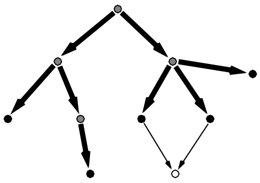
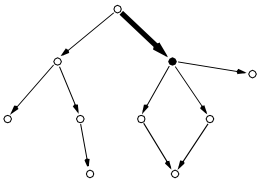
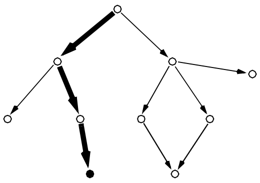
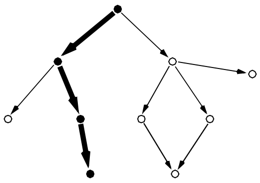
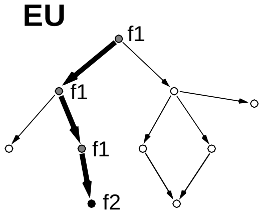
TEMPORAL LOGICS, BASICS

- **extension of classical (propositional) logics by temporal operators**
- **atomic propositions**
  -> *elementary statements, having - in a given state - a well-defined truth value*
  -> *e. g. fork1, for 1-bounded Petri net (Boolean)*
  -> *e. g. buffer = 2, buffer > 2, else (Integer)*
- **constants**
  -> *TRUE, FALSE*
- **classical Boolean operators**

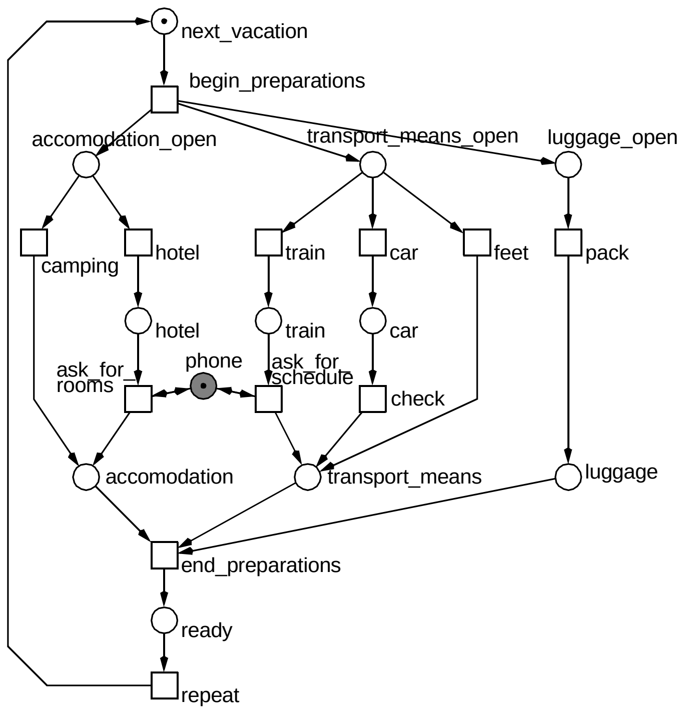
negation	*!*	*conjunction*	*
disjunction	+	*implication*	->

- **temporal operators**
  -> *to refer to sequences of states*

CTL Operators, Interleaving Semantics

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

Example 3 - Travel Planing



Example 3 - Some Simple Properties (1)

- **this is a possible combination of choices**
  EF *(hotel * car);* *-> TRUE*
- **this is the only possible combination of choices**
  AF *(hotel * car);* *-> FALSE*
- **this is an impossible combination of choices**
  ! ***EF*** *(train * car);* *-> TRUE*
- **any travel planning will be finished finally**
  AG *(next_vacation -> AF (ready));* *-> TRUE*
- **atomic use of the phone only,**
  the phone is never reserved over several steps
  AG *(phone);* *-> TRUE*
- **liveness of transition end_preparations**
  AG *(****EF*** *(accommodation * transport_means * luggage));* *-> TRUE*

Example 3 - Some Simple Properties (2)

- **livelock freedom/progress of transition end_preparations**
  AG *(****AF*** *(accommodation * transport_means * luggage));* *-> TRUE*
- **undone luggage will always be done in one step**
  AG *(luggage_open ->* ***AX*** *(luggage));* *-> FALSE*
- **it's possible that undone luggage will be done in one step**
  AG *(luggage_open ->* ***EX*** *(luggage));* *-> TRUE*
- **undone luggage will be done finally**
  AG *(luggage_open ->* ***AF*** *(luggage));* *-> TRUE*
- **luggage will remain undone until it is done**
  A *(luggage_open* ***U*** *luggage);* *-> FALSE*
- **if the luggage is undone, it remains undone until it is done**
  AG *(luggage_open ->* ***A*** *(luggage_open* ***U*** *luggage));* *-> TRUE*

Temporal Logics, Typical Questions

- **reachability-related** **EF (φ)**
  -> *There exists at least one computational path to reach eventually a state, where φ will be true.*

- **safety-related** **AG (! φ)** **-> equivalent to ! EF (φ)**
  -> *For every computational path, φ will never be true.*

- **invariant-related** **AG (φ)** **-> equivalent to ! EF (! φ)**
  -> *For every computational path, φ will be true for ever.*

- **liveness-related** **AG (EF (φ))**
  -> *What ever happens, there exists the chance (at least one computational path) that φ will be true again.*

- **progress-related** **AG (AF (φ))**
  -> *For every computational path, φ will eventually be true.*

Model Checking, More Examples

- **insights / reachability**
  -> *Is it possible, that both robot arms carry a plate at the same time?*
  ***EF** (arm1_mag_on * arm2_mag_on)*

- **consistency property**
  -> *The swivel is always either stopped or moves in exactly one direction.*
  ***G** (robot_stop xor robot_left xor robot_right)*

- **safety property**
  -> *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)), with*
  *φ = arm1_mag_on * arm1_pickup_angle * arm1_pickup_ext*
  *ψ = arm1_release_angle * arm1_release_ext*
  -> *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))*

Bottleneck:
state explosion
(no primitive recursive function ...)

DINING PHILOSOPHERS 1000, BDD-BASED

Number of places/marked places/transitions: 7,000/2,000/5,000 *[JSP 2001]*

Number of states:

113751760865620516280672035436276768405854187694780001109285823216991\8\\
159959588122031332641120690971790713407413960379370132051412946235771\0\\
244289522738424241885324723952294300718880861927052755597203329394869\1\\
334498271287409035878953318171137286359195790723689557093738307422542\1\\
493299735055934871120872608511650262781852464476299128123872281683542\6\\
439043702222222716712699874004961590120093014497021663026892511863169\6\\
792192797756430854076755677722422066045029462353435568315492194903488\7\\
413893510872611522753508464671945735340847108696533249480549775338294\2\\
171781101168772051021154169003921176627995642292903237688541475038527\5\\
512488192401053636525511904747774118 74

ca. 1.1 * 10^667

Number of places/marked places/transitions:	*7000/2000/5000*
Time to compute P-Invariants:	*45885.66 sec*
Number of P-Invariants:	*3000*
Time to compute compact coding:	*385.59 sec*
Number of Clusters:	*3000*
Number of Variables:	*4000*
Time:	*3285.73 sec, ca. 54.75 min*

POSSIBLE ANSWERS TO STATE EXPLOSION

- **compositional methods**
  -> *simple interfaces between modules*
- **abstraction by ignoring some state information**
  -> *conservative approximation*

'BASE CASE' TECHNIQUES

- **different logics -> different algorithms, e.g. LTL**
- **integer programming**
- **compressed state space representations**
  -> *BDD, IDD, ...*
- **lazy state space construction**
  -> *partial order methods*
- **alternative state spaces**
  -> *partial order representations, e.g. prefix*

'ALTERNATIVE' METHODS (PN)

TEMPORAL LOGICS, OVERVIEW

time \ semantics	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,* ***Charlie, BDD/IDD-CTL, ...***	**prefix** (conflicts & concurrency) McMillan, Esparza, pd 2001 *PEP*

TECHNIQUES & TOOLS, OVERVIEW

technique	CTL	LTL
reachability graph	INA, Charlie	PROD, MARIA
stubborn set reduced reachability graph	LoLA	PROD (LTL\X)
symmetrically reduced reachability graph	LoLA (symmetric formulas)	?
BDD, NDD, ..., xDD	BDD-CTL, SMART, IDD-CTL	BDD-LTL
Kronecker algebra	[Kemper]	?
prefix	PEP (CTL_0)	QQ (LTL\X)
process automata	[pd]	?

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’

-> control model, 5 plates, full state space: 1,657,242 [7,185,779]

State Spaces, Examples - BDD/IDD

case study	net size	# of states
production cell, 5 plates	231 P / 202 T	1.6 * 10^6
production cell, 3 plates		7.1 * 10^6
1000 dining philosophers	7,000 P / 5,000 T	1.1 * 10^667
solitaire, standard	65 P / 75 T	1.8 * 10^8
solitaire, non-standard	73 P / 91 T	2.9 * 10^9
Halobacterium, motor 44	38 P / 60 T	4.8 * 10^7
MAPK cascade, 80 levels	22 P / 30 T	5.6 *10^18
MAPK cascade, 120 levels		1.7 * 10^21
gene regulation, 6 cells, m01	144 P / 307 T / 2146 A	7.4 * 10^21
gene regulation, 6 cells, m02		3,3 * 10^25

Summary Model Checking (1)

- **model checking can only be as good as the model specification**
  - -> *readable <-> unambiguous*
  - -> *complete <-> limited size*
- **correct model checking requires correct requirement specifications**
  - -> *take your time, think twice*
- **model checking proves consistency of**
  - -> *model & requirements*
- **if the answer is NO (-> FALSE)**
  - -> *the model can be wrong*
  - -> *the requirements can be wrong*
  - -> *or both*
- **if the answer is YES (-> TRUE)**
  - -> *model & requirements can still contain (same) logical faults* -> *unlikely !?*

Summary Model Checking (2)

- **(up to now) restricted to bounded systems**
  - -> *numerous ongoing research*
  - -> *CTL - undecidable*
  - -> *LTL - decidable, but no tools (not yet ?)*
  - -> *unboundedness + inhibitor arcs = undecidability*
- **model checking is extremely time and resource consuming**
  - -> *'external' quality pressure*
- **model checking is not manageable without theory supported by tools**
- **model checking needs knowledgeable professionals**
  - -> *study / job specialization*
  - -> *profession of "software validator", "software tester"*
- **There is no such thing as a fault-free model (system) !**
  - -> *sufficient dependability for a given user profile*

Model checking
is no substitute for thinking !

REFERENCES, METHODS

- Pnueli, A.:
  The Temporal Logic of Concurrent Programs;
  18th. IEEE Symp. on Foundation of CS, 46-57, IEEE Computer Society Press 1977.
- Emerson, E. A.:
  Branching Time Temporal Logic and the Design of Correct Concurrent Programs.
  PhD thesis, Harvard Univ. 1981.
- MacMillan, K. L.:
  Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits;
  Proc. of the 4th Workshop on Computer Aided Verification, 164-174, 1992.
- Esparza, J.:
  Model Checking Using Net Unfoldings;
  Science of Computer Programming, 23(1994), 151-195.
- EM Clarke, O Grumberg, and DA Peled:
  Model checking;
  MIT Press 1999, third printing, 2001.

REFERENCES, TOOLS

- J Spranger:
  Symbolic LTL verification of Petri nets (in German);
  PhD thesis, BTU Cottbus, Dep. of CS, 2001.
- Martin Schwarick:
  A Tool to Analyse Petri Net Models (in German);
  Master thesis, BTU Cottbus, Dep. of CS, 2006.
- A Tovchigrechko:
  Model checking using interval decision diagrams;
  PhD thesis, BTU Cottbus, Dep. of CS, 2008.
- A Franzke:
  Charlie 2.0 - a multi-threaded Petri net analyzer;
  Master thesis, BTU Cottbus, Dep. of CS, 2009.
- 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.
- 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

- 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.

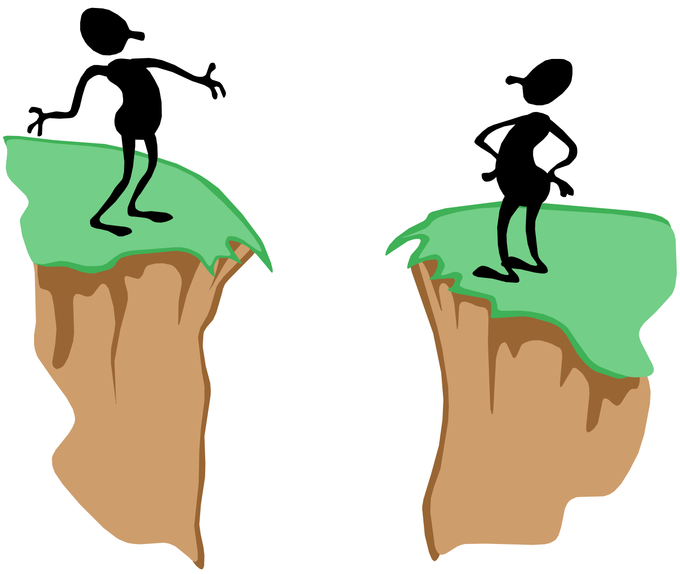
- M Heiner, D Gilbert, R Donaldson:
  Petri Nets for Systems and Synthetic Biology;
  SFM 2008, Springer LNCS 5016, pp. 215-264, 2008.

- 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.

- 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.

- 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.

dependability engineering



Thanks !

http://www-dssz.informatik.tu-cottbus.de

monika.heiner(at)b-tu.de July 2017