Parameter Synthesis with IC3

Alessandro Cimatti, Alberto Griggio, **Sergio Mover**, Stefano Tonetta

Fondazione Bruno Kessler, Trento, Italy

April 11,
SynCoP 2015

This work was presented at FMCAD 2013 [CGMT13]

Motivations

- Context: automatic-design of safety-critical systems
- Systems have parameters (e.g. unknown constants):
  - instantiated only in the implementation
  - represent uncertainty (e.g. environment, robustness of the system)

Problem: automatically synthesize parameter valuations that satisfy some requirements

Parameter synthesis - several possible instance

- infinite vs. finite domain parameters
- satisfy invariant vs. LTL properties
- universal vs. existential problem
  (for all executions the property hold vs.
  there exists an execution such that the property does not hold)
- all the valuations vs. some
- optimal vs. no requirements

Settings in this talk

- infinite vs. finite domain parameters
  Real-type parameters
- satisfy invariant vs. LTL properties
- universal vs. existential problem
  property holds for all the possible traces
- all the valuations vs. some
- optimal vs. no requirements
  we do not focus on optimal
  (the set found could be further refined, e.g. using OptiMathsat)

Settings in this talk

- infinite vs. finite domain parameters
  Real-type parameters
- satisfy invariant vs. LTL properties
- universal vs. existential problem
  property holds for all the possible traces
- all the valuations vs. some
- optimal vs. no requirements
  we do not focus on optimal
  (the set found could be further refined, e.g. using OptiMathsat)

Synthesize the set of all the parameters valuations that guarantees that an invariant property hold

More general view

Several problems can be casted to parameter synthesis, e.g.:

- sensor synthesis for diagnosability
  parameters are sensors, property to fulfill "the system is diagnosable"
  E.g. [Bittner et.al., FMCAD14]
- fault tree generation
  parameters are fault variables, properties are top-level events

In this talk

We extend IC3 [Bradley, VMCAI11] to perform parameter synthesis:

- IC3 is an invariant verification algorithm
- if infinite state transition system
  good representation of sw, real-time and hybrid systems
  it uses Satisfiability Modulo Theories (SMT)
- we exploit some of the IC3 features:
  - it is easy to use in an incremental way
  - it finds a set of traces
    the IC3 extension to SMT that we use

Outline

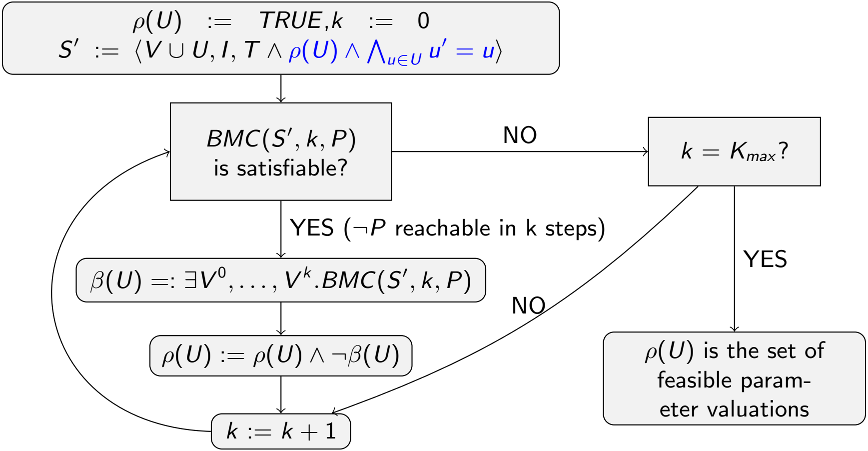
Outline

Parametric transition system

$S = \langle U, V, I, T \rangle$ is a parametric transition system

- U: set of parameters (of real type) ,
- V: set of variables,
- $I(V \cup U)$: initial states,
- $T(V \cup U \cup V')$: transitions (semantic: parameters never change);

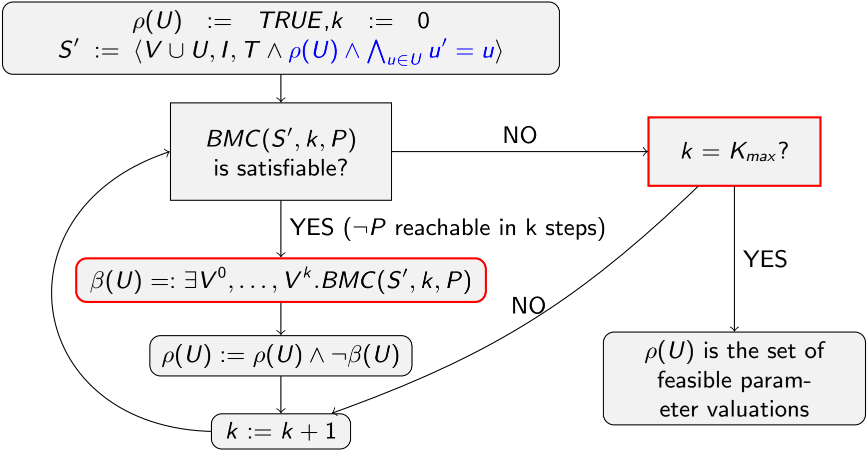
Our starting point [CPR08]



Algorithm:

- iteratively finds a set of unfeasible parameter valuations
- uses Bounded Model Checking (BMC) to find a counterexample at the k-th step

Our starting point [CPR08]



Weaknesses:

- the termination by assuming the existence of a maximum bound K_{max}
- quantifier elimination to compute $\beta(U)$ is a bottleneck.

IC3 in a nutshell

Disclaimer: I am skipping most of the details of IC3

IC3 in a nutshell

Disclaimer: I am skipping most of the details of IC3

IC3 builds a sequence of sets of states (frames) $F_0; F_1; \ldots; F_k$:

- $i \leq k$, $F_i \models P$
- $i < k$, $F_i \rightarrow F_{i+1}$
- $i < k$, $F_i \wedge T \rightarrow F_{i+1}$

IC3 in a nutshell

Disclaimer: I am skipping most of the details of IC3

IC3 builds a sequence of sets of states (frames) $F_0; F_1; \ldots; F_k$:

- $i \leq k$, $F_i \models P$
- $i < k$, $F_i \rightarrow F_{i+1}$
- $i < k$, $F_i \wedge T \rightarrow F_{i+1}$

How does it work?

- it iterates 2 phases:
  1. Blocking phase: proves that the frame F_k cannot reach $\neg P$
  2. Propagation phase: proves that some facts that hold at F_i also holds at F_{i+1}
- it stops if either finds a counterexample or an inductive invariant

IC3 in a nutshell

Disclaimer: I am skipping most of the details of IC3

IC3 builds a sequence of sets of states (frames) $F_0; F_1; \ldots; F_k$:

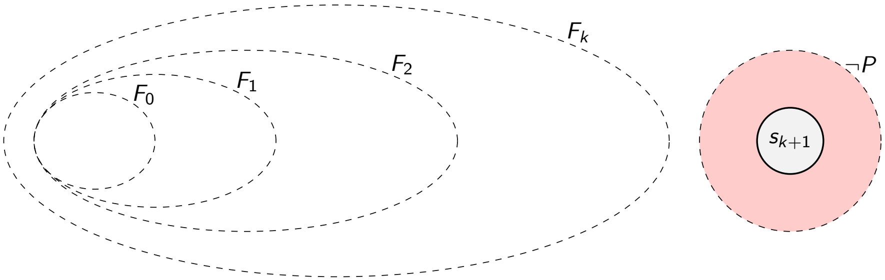
- $i \leq k$, $F_i \models P$
- $i < k$, $F_i \rightarrow F_{i+1}$
- $i < k$, $F_i \wedge T \rightarrow F_{i+1}$

How does it work?

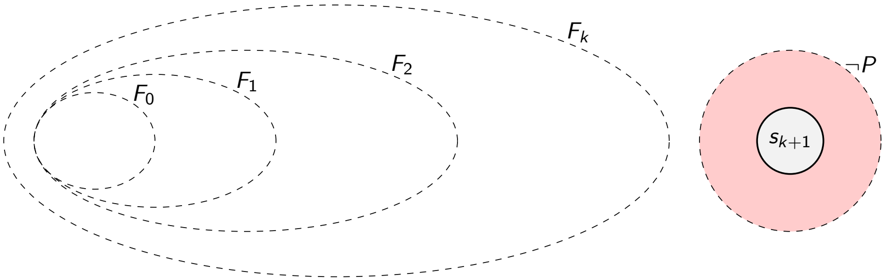
- it iterates 2 phases:
  1. Blocking phase: proves that the frame F_k cannot reach $\neg P$
  2. Propagation phase: proves that some facts that hold at F_i also holds at F_{i+1}
- it stops if either finds a counterexample or an inductive invariant

we are interested in the blocking phase, which finds counterexamples to P

Snapshot of the blocking phase

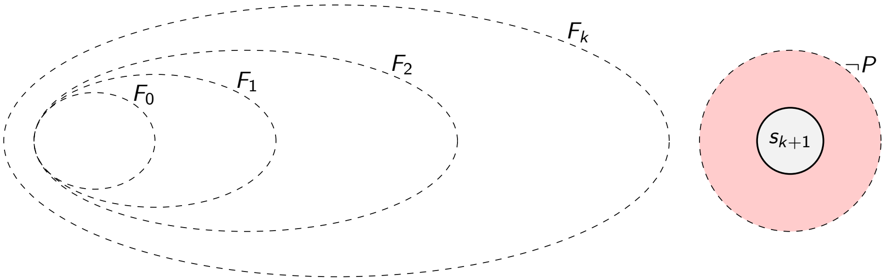


Snapshot of the blocking phase



Can we reach s_{k+1} from F_k? (i.e. $F_k \wedge \neg s_{k+1} \wedge T \rightarrow s'_{k+1}$?)

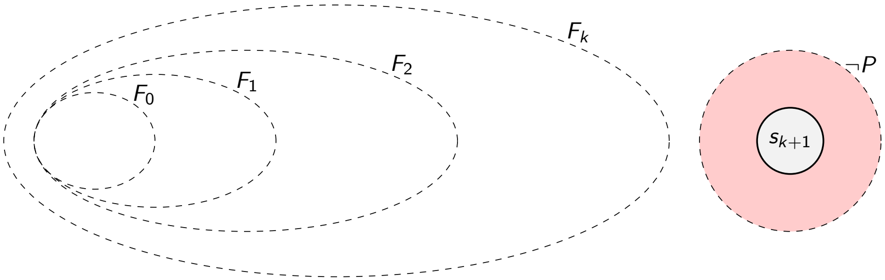
Snapshot of the blocking phase



Can we reach s_{k+1} from F_k? (i.e. $F_k \wedge \neg s_{k+1} \wedge T \rightarrow s'_{k+1}$?)

- No: s_{k+1} cannot be reached from F_k

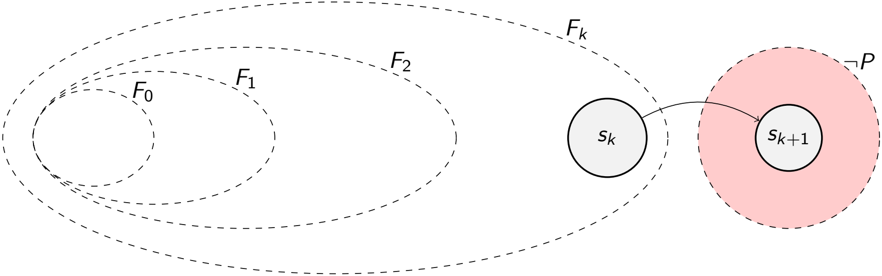
Snapshot of the blocking phase



Can we reach s_{k+1} from F_k? (i.e. $F_k \wedge \neg s_{k+1} \wedge T \rightarrow s'_{k+1}$?)

- No: s_{k+1} cannot be reached from F_k
- Yes: s_{k+1} can be reached from F_k.

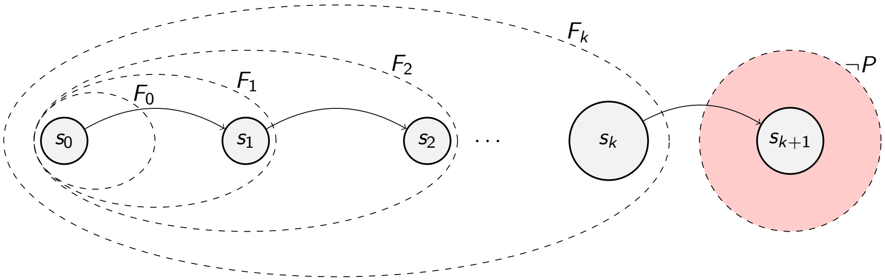
Snapshot of the blocking phase



Can we reach s_{k+1} from F_k? (i.e. $F_k \wedge \neg s_{k+1} \wedge T \rightarrow s'_{k+1}$?)

- No: s_{k+1} cannot be reached from F_k
- Yes: s_{k+1} can be reached from F_k.
  We find a set of predecessors $s_k = \exists V.(F_k \wedge \neg s_{k+1} \wedge T \wedge s'_{k+1})$
  s_k is computed with an under-approximated qelim

Snapshot of the blocking phase

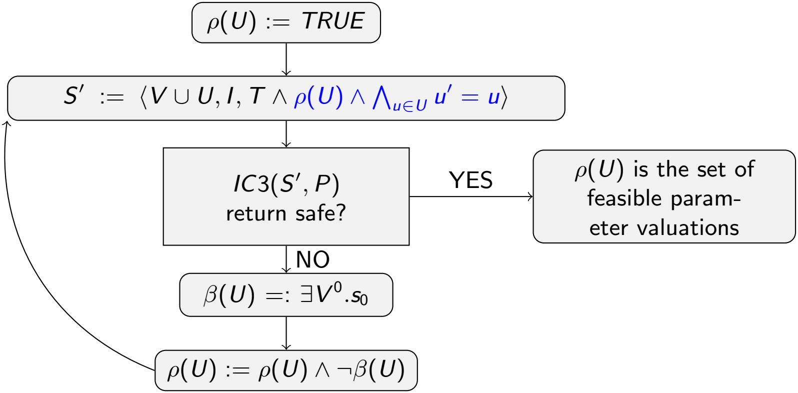


Can we reach s_{k+1} from F_k? (i.e. $F_k \wedge \neg s_{k+1} \wedge T \rightarrow s'_{k+1}$?)

- No: s_{k+1} cannot be reached from F_k
- Yes: s_{k+1} can be reached from F_k.
  We find a set of predecessors $s_k = \exists V.(F_k \wedge \neg s_{k+1} \wedge T \wedge s'_{k+1})$
  s_k is computed with an under-approximated qelim

$s_0, \ldots, s_k$ are sets of states $\rightarrow \pi = s_0; \ldots; s_k$ represents a set of traces!

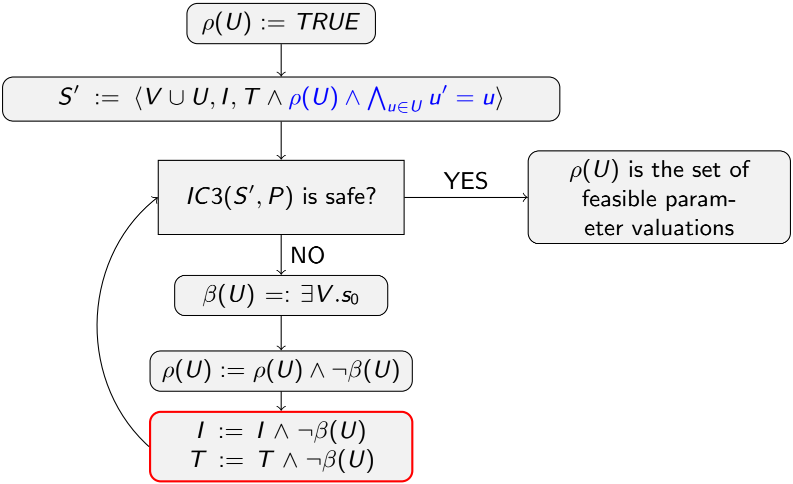
IC3 for synthesis - algorithm



Cheap quantifier elimination

- s_0 represents a set of states
- All the states in s_0 can reach $\neg P$ (they are "bad" states)
- cheap quantifier elimination $\exists V^0 . s_0(V^0, U)$

Optimizations 1/2 - Incrementality



- No need to restart from scratch $\rightarrow$ keep all the previously computed frames
- Exploits the incrementality in the SMT solver

Optimizations 2/2 - Trade-off generality/cost of qelim

- $\pi = s_0; \ldots ; s_k$ represents a set of traces, each s_i is bad!
- we can try several heuristics:
  - $\exists V.s_0$
  - $j \leq k$, $\exists V^0, \ldots, V^j.I \wedge \bigwedge_{i=0}^{i<j}(T^i \wedge s_i) \wedge s_j$
  - $\exists V^0, \ldots, V^k.I \wedge \bigwedge_{i=0}^{j<k}(T^j \wedge \neg P^j)$
- we can trade results' generality with the cost of the quantifier elimination
- in practice: start with $V.s_0$ and switch to $\exists V^0, \ldots, V^k.I \wedge \bigwedge_{i=0}^{j<k}(T^j \wedge \neg P^j)$ after enumerating too many counterexamples

Outline

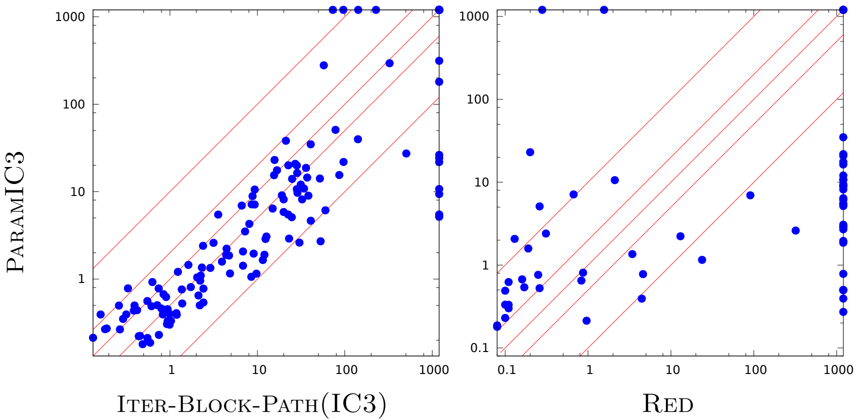
Experimental evaluation

- We use linear hybrid automata benchmarks
- Our implementation:
  - uses the MATHSAT SMT solver
  - done inside the NUXMV model checker
  - it is integrated in HYCOMP, a model checker for hybrid systems

more on HYCOMP at TACAS

- Competitors:
  - ITER-BLOCK-PATH(IC3): non-incremental version of [CPR08], using IC3 as a "black box"
  - RED [Wan05]: tool for parameter synthesis for linear hybrid automata:
    - it computes all the reachable states of the system
    - symbolic representation of the state space

Results


ParamIC3
1000
100
10
1
1
10
100
1000
Iter-Block-Path(IC3)
1000
100
10
1
0.1
0.1
1
10
100
1000
Red

Outline

Conclusion

In this talk we shown a simple extension of IC3 to synthesize parameters

- the approach works for infinite-state transition systems
- it exploits IC3 for incrementality and to have a cheap quantifier elimination
- good performance results

Future works

- directly solve the “universal” problem
- a lot of future works are motivated by recent extensions of IC3:
  - to use predicate abstraction: more efficient algorithm
  - to prove LTL formulas: synthesize parameters that preserve liveness
- find the optimal set of parameter regions (e.g. consider pareto optimality)
- extend the approach to hybrid systems with more complex dynamics

Bibliography I

Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta.
Parameter synthesis with ic3.
In *FMCAD*, pages 165–168, 2013.

Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian.
Symbolic computation of schedulability regions using parametric timed automata.
In *RTSS*, pages 80–89, 2008.

Farn Wang.
Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures.
IEEE Trans. Software Eng., 31(1):38–51, 2005.

Parameter synthesis - formal definition

$S = \langle U, V, I, T \rangle$ is a parametric transition system

- U: set of parameters, V: set of variables, $I(V \cup U)$: initial states, $T(V \cup U \cup V')$: transitions;

$S_\gamma = \langle V, \gamma(I), \gamma(T) \rangle$: TS induced by the valuation γ

- γ assigns a value to each parameter in U
- $\gamma(\phi)$: substitute in ϕ each occurrence of a paramater with its value *(e.g.* $\gamma = \{p = 3\}$, $\gamma(x + p \leq 0) := x + 3 \leq 0$*)*

Parameter synthesis problem:

- $P(U, X)$ is an invariant property
- γ is feasible for the property iff $S_\gamma \models \gamma(P)$.
- find the set $\rho(U)$ of all the feasible parameter valuations.

Tradeoff generalty/costs of quantifier elimination