Model Checking of Array-Based Systems: from Foundations to Implementation

S. Ghilardi[1] and S. Ranise[2]

[1]Università degli Studi di Milano

[2]Università di Verona

Oslo - July 6th, 2009

Outline

Outline

Outline

Outline

Verification of Parametrised Systems

- Parametrised system = bunch of concurrent processes (topology may vary, can be e.g., set-like, linear-like, tree-like, ring-like, ...)
- Process = instance of the same state-machine
- Configuration = state of a parametrised system
- Transition = either a process changing its locations/data or several processes simultaneously changing their respective locations/data (e.g., broadcast) [interleaving semantics]
- CHALLENGE: automatically verify a property regardless of the number of processes
- A state machine has finitely many control locations and can manipulate finitely many variables over possibly unbounded domains

Verification of Parametrised Systems

- Parametrised system = bunch of concurrent processes (topology may vary, can be e.g., set-like, linear-like, tree-like, ring-like, ...)
- Process = instance of the same state-machine
- Configuration = state of a parametrised system
- Transition = either a process changing its locations/data or several processes simultaneously changing their respective locations/data (e.g., broadcast) [interleaving semantics]
- CHALLENGE: automatically verify a property regardless of the number of processes
- A state machine has finitely many control locations and can manipulate finitely many variables over possibly unbounded domains

Verification of Parametrised Systems

- Parametrised system = bunch of concurrent processes (topology may vary, can be e.g., set-like, linear-like, tree-like, ring-like, ...)
- Process = instance of the same state-machine
- Configuration = state of a parametrised system
- Transition = either a process changing its locations/data or several processes simultaneously changing their respective locations/data (e.g., broadcast) [interleaving semantics]
- CHALLENGE: automatically verify a property regardless of the number of processes
- A state machine has finitely many control locations and can manipulate finitely many variables over possibly unbounded domains

Verification of Parametrised Systems

- Parametrised system = bunch of concurrent processes (topology may vary, can be e.g., set-like, linear-like, tree-like, ring-like, ...)
- Process = instance of the same state-machine
- Configuration = state of a parametrised system
- Transition = either a process changing its locations/data or several processes simultaneously changing their respective locations/data (e.g., broadcast) [interleaving semantics]
- CHALLENGE: automatically verify a property regardless of the number of processes
- A state machine has finitely many control locations and can manipulate finitely many variables over possibly unbounded domains

Verification of Parametrised Systems

- Parametrised system = bunch of concurrent processes (topology may vary, can be e.g., set-like, linear-like, tree-like, ring-like, ...)
- Process = instance of the same state-machine
- Configuration = state of a parametrised system
- Transition = either a process changing its locations/data or several processes simultaneously changing their respective locations/data (e.g., broadcast) [interleaving semantics]
- CHALLENGE: automatically verify a property regardless of the number of processes
- A state machine has finitely many control locations and can manipulate finitely many variables over possibly unbounded domains

Verification of Parametrised Systems

- Parametrised system = bunch of concurrent processes (topology may vary, can be e.g., set-like, linear-like, tree-like, ring-like, ...)
- Process = instance of the same state-machine
- Configuration = state of a parametrised system
- Transition = either a process changing its locations/data or several processes simultaneously changing their respective locations/data (e.g., broadcast) [interleaving semantics]
- CHALLENGE: automatically verify a property regardless of the number of processes
- A state machine has finitely many control locations and can manipulate finitely many variables over possibly unbounded domains

Well-Structured Transition Systems

Seminal paper [ACJT - LICS96]

$$(S, \tau, \preceq)$$

- S: set of states;
- $\tau = \{\rightarrow_\lambda \subseteq S \times S\}_\lambda$: labelled directed graph;
- $\preceq$: well quasi ordering[1] (wqo) on S;
- each τ_λ is monotonic:

$$\begin{array}{ccc} s_1 & \preceq & s_2 \\ \downarrow_\lambda & & \downarrow_\lambda \\ s_3 & \preceq\ \exists & s_4 \end{array}$$

[1] Reflexive, transitive binary relation that neither contains infinite strictly decreasing sequences nor infinite sequences of pairwise incomparable elements

Well-Structured Transition Systems

Seminal paper [ACJT - LICS96]

$$(S, \tau, \preceq)$$

- S: set of states;
- $\tau = \{\rightarrow_\lambda \subseteq S \times S\}_\lambda$: labelled directed graph;
- $\preceq$: well quasi ordering[1] (wqo) on S;
- each τ_λ is monotonic:

$$\begin{array}{ccc} s_1 & \preceq & s_2 \\ \downarrow_\lambda & & \downarrow_\lambda \\ s_3 & \preceq\ \exists & s_4 \end{array}$$

[1] Reflexive, transitive binary relation that neither contains infinite strictly decreasing sequences nor infinite sequences of pairwise incomparable elements

Well-Structured Transition Systems

Seminal paper [ACJT - LICS96]

$$(S, \tau, \preceq)$$

- S: set of states;
- $\tau = \{\rightarrow_\lambda \subseteq S \times S\}_\lambda$: labelled directed graph;
- $\preceq$: well quasi ordering[1] (wqo) on S;
- each τ_λ is monotonic:

$$\begin{array}{ccc} s_1 & \preceq & s_2 \\ \downarrow_\lambda & & \downarrow_\lambda \\ s_3 & \preceq\ \exists & s_4 \end{array}$$

[1] Reflexive, transitive binary relation that neither contains infinite strictly decreasing sequences nor infinite sequences of pairwise incomparable elements

Well-Structured Transition Systems

Seminal paper [ACJT - LICS96]

$$(S, \tau, \preceq)$$

- S: set of states;
- $\tau = \{\rightarrow_\lambda \subseteq S \times S\}_\lambda$: labelled directed graph;
- $\preceq$: well quasi ordering[1] (wqo) on S;
- each τ_λ is monotonic:

$$\begin{array}{ccc} s_1 & \preceq & s_2 \\ \downarrow_\lambda & & \downarrow_\lambda \\ s_3 & \preceq\ \exists & s_4 \end{array}$$

[1] Reflexive, transitive binary relation that neither contains infinite strictly decreasing sequences nor infinite sequences of pairwise incomparable elements

Well-Structured Transition Systems

Seminal paper [ACJT - LICS96]

$$(S, \tau, \preceq)$$

- S: set of states;
- $\tau = \{\rightarrow_\lambda \subseteq S \times S\}_\lambda$: labelled directed graph;
- $\preceq$: well quasi ordering[1] (wqo) on S;
- each τ_λ is monotonic:

$$\begin{array}{ccc} s_1 & \preceq & s_2 \\ \downarrow_\lambda & & \downarrow_\lambda \\ s_3 & \preceq\ \exists & s_4 \end{array}$$

[1] Reflexive, transitive binary relation that neither contains infinite strictly decreasing sequences nor infinite sequences of pairwise incomparable elements

Well-Structured Transition Systems

Seminal paper [ACJT - LICS96]

$$(S, \tau, \preceq)$$

- S: set of states;
- $\tau = \{\rightarrow_\lambda \subseteq S \times S\}_\lambda$: labelled directed graph;
- $\preceq$: well quasi ordering[1] (wqo) on S;
- each τ_λ is monotonic:

$$\begin{array}{ccc} s_1 & \preceq & s_2 \\ \downarrow_\lambda & & \downarrow_\lambda \\ s_3 & \preceq \exists & s_4 \end{array}$$

[1] Reflexive, transitive binary relation that neither contains infinite strictly decreasing sequences nor infinite sequences of pairwise incomparable elements

Representing sets of states: key properties

- Set of unsafe states represented by an upset K:

$$s \in K \wedge s \preceq s' \rightarrow s' \in K$$

- Monotonicity implies that the pre-image of an upset is still an upset

$$Pre(\tau, K) \quad := \quad \{s \mid \exists \lambda \, \exists s' \, (s \xrightarrow{\lambda} s') \wedge s' \in K\}$$

- Since $\preceq$ is a wqo, upsets can be finitely represented by their *finitely many* minimal elements

Backward Reachability

Checking that a set K of unsafe states is (un-)reachable from a set I of initial states

function BReach(K)
  $i \longleftarrow 0$; $BR^0(\tau, K) \longleftarrow K$; $K^0 \longleftarrow K$
  if $BR^0(\tau, K) \cap I \neq \emptyset$ **then return** unsafe
  repeat
    $K^{i+1} \longleftarrow \mathrm{Pre}(\tau, K^i)$
    $BR^{i+1}(\tau, K) \longleftarrow BR^i(\tau, K) \cup K^{i+1}$
    if $BR^{i+1}(\tau, K) \cap I \neq \emptyset$ **then return** unsafe
    else $i \longleftarrow i + 1$
  until $BR^{i+1}(\tau, K) \subseteq BR^i(\tau, K)$
  return safe
end

Backward Reachability

Checking that a set K of unsafe states is (un-)reachable from a set I of initial states

function BReach(K)
  $i \longleftarrow 0$; $BR^0(\tau, K) \longleftarrow K$; $K^0 \longleftarrow K$
  if $BR^0(\tau, K) \cap I \neq \emptyset$ **then return** unsafe
  repeat
    $K^{i+1} \longleftarrow \mathrm{Pre}(\tau, K^i)$
    $BR^{i+1}(\tau, K) \longleftarrow BR^i(\tau, K) \cup K^{i+1}$
    if $BR^{i+1}(\tau, K) \cap I \neq \emptyset$ **then return** unsafe
    else $i \longleftarrow i + 1$
  until $BR^{i+1}(\tau, K) \subseteq BR^i(\tau, K)$
  return safe
end

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Termination and Empirical Success

- Since $\preceq$ is a wqo, the algorithm terminates.
- Extensions to cases in which $\preceq$ is not a wqo often terminate 'in practice'.
- Lot of success for the verification of safety properties of a variety of systems:
  - broadcast protocols [DEP - CSL99],[EFM - LICS99];
  - lossy channels systems [AG - 96];
  - various mutual exclusion (e.g. 'bakery-like'), cache coherence, broadcast protocols [ABDR - TACAS07], [ADR - CAV07];
  - parametrized systems with non-atomic global conditions [ABDR - VMCAI08]
  - parametrized systems with tree-like topologies [ABDR - FORTE08];
  - ...

Discussion

States of parametrised systems are encoded both

- symbolically (e.g., using constraints for unbounded variables) and
- non-symbolically (e.g., data structures for the topology)

DRAWBACK: every time you change topology, the computation of the pre-image must be changed accordingly; difficulties arise in the integration of constraint solving and data-structure manipulation for safety and fix-point checking

OUR GOAL: to obtain a fully symbolic description of the configurations of a parametrised system; to get an efficient backward reachability analysis by using state-of-the-art SMT solving for both safety and fix-point checking

Discussion

States of parametrised systems are encoded both

- symbolically (e.g., using constraints for unbounded variables) and
- non-symbolically (e.g., data structures for the topology)

DRAWBACK: every time you change topology, the computation of the pre-image must be changed accordingly; difficulties arise in the integration of constraint solving and data-structure manipulation for safety and fix-point checking

OUR GOAL: to obtain a fully symbolic description of the configurations of a parametrised system; to get an efficient backward reachability analysis by using state-of-the-art SMT solving for both safety and fix-point checking

Discussion

States of parametrised systems are encoded both

- symbolically (e.g., using constraints for unbounded variables) and
- non-symbolically (e.g., data structures for the topology)

DRAWBACK: every time you change topology, the computation of the pre-image must be changed accordingly; difficulties arise in the integration of constraint solving and data-structure manipulation for safety and fix-point checking

OUR GOAL: to obtain a fully symbolic description of the configurations of a parametrised system; to get an efficient backward reachability analysis by using state-of-the-art SMT solving for both safety and fix-point checking

Key features of previous work

Model-theoretic (see [RV - PSI03], Brain, ALV, CLP-model-checking)

- a first-order (possibly many-sorted) domain structure is fixed;
- a fixed (finite) set of variables $\underline{x}$ models system variables;
- states of the system are assignments to $\underline{x}$;
- initial and unsafe states are described via formulae $I(\underline{x}), K(\underline{x})$;
- transitions are also modeled by formulae $\tau(\underline{x}, \underline{x}')$ relating current and updated values of system variables;
- reachability is symbolically executed through satisfiability (modulo theory) tests.

Key features of previous work

Model-theoretic (see [RV - PSI03], Brain, ALV, CLP-model-checking)

- a first-order (possibly many-sorted) domain structure is fixed;
- a fixed (finite) set of variables $\underline{x}$ models system variables;
- states of the system are assignments to $\underline{x}$;
- initial and unsafe states are described via formulae $I(\underline{x}), K(\underline{x})$;
- transitions are also modeled by formulae $\tau(\underline{x}, \underline{x}')$ relating current and updated values of system variables;
- reachability is symbolically executed through satisfiability (modulo theory) tests.

Key features of previous work

Model-theoretic (see [RV - PSI03], Brain, ALV, CLP-model-checking)

- a first-order (possibly many-sorted) domain structure is fixed;
- a fixed (finite) set of variables $\underline{x}$ models system variables;
- states of the system are assignments to $\underline{x}$;
- initial and unsafe states are described via formulae $I(\underline{x}), K(\underline{x})$;
- transitions are also modeled by formulae $\tau(\underline{x}, \underline{x}')$ relating current and updated values of system variables;
- reachability is symbolically executed through satisfiability (modulo theory) tests.

Key features of previous work

Model-theoretic (see [RV - PSI03], Brain, ALV, CLP-model-checking)

- a first-order (possibly many-sorted) domain structure is fixed;
- a fixed (finite) set of variables $\underline{x}$ models system variables;
- states of the system are assignments to $\underline{x}$;
- initial and unsafe states are described via formulae $I(\underline{x}), K(\underline{x})$;
- transitions are also modeled by formulae $\tau(\underline{x}, \underline{x}')$ relating current and updated values of system variables;
- reachability is symbolically executed through satisfiability (modulo theory) tests.

Key features of previous work

Model-theoretic (see [RV - PSI03], Brain, ALV, CLP-model-checking)

- a first-order (possibly many-sorted) domain structure is fixed;
- a fixed (finite) set of variables $\underline{x}$ models system variables;
- states of the system are assignments to $\underline{x}$;
- initial and unsafe states are described via formulae $I(\underline{x}), K(\underline{x})$;
- transitions are also modeled by formulae $\tau(\underline{x}, \underline{x}')$ relating current and updated values of system variables;
- reachability is symbolically executed through satisfiability (modulo theory) tests.

Key features of previous work

Model-theoretic (see [RV - PSI03], Brain, ALV, CLP-model-checking)

- a first-order (possibly many-sorted) domain structure is fixed;
- a fixed (finite) set of variables $\underline{x}$ models system variables;
- states of the system are assignments to $\underline{x}$;
- initial and unsafe states are described via formulae $I(\underline{x}), K(\underline{x})$;
- transitions are also modeled by formulae $\tau(\underline{x}, \underline{x}')$ relating current and updated values of system variables;
- reachability is symbolically executed through satisfiability (modulo theory) tests.

Towards Array-based Systems

- Topology of the parametrised system: theory[2] $T_I = (\Sigma_I, \mathcal{C}_I)$
  E.g.: $\mathcal{C}_I$ consists of all (finite) sets, linear orders, forests/trees, graphs, ...
- Data manipulated by the parametrised system: theory $T_E = (\Sigma_E, \mathcal{C}_E)$
  Usually $\mathcal{C}_E$ contains just one structure: (products of) integers, reals, Booleans, control locations, ...
- We assume the availability of SMT solvers deciding the satisfiability of quantifier-free formulae modulo T_I and T_E

[2] A *theory* T is a pair $(\Sigma, \mathcal{C})$, where Σ is a first-order signature and $\mathcal{C}$ is a class of Σ-structures (the models of T).

Towards Array-based Systems

- Topology of the parametrised system: theory[2] $T_I = (\Sigma_I, \mathcal{C}_I)$
  E.g.: $\mathcal{C}_I$ consists of all (finite) sets, linear orders, forests/trees, graphs, ...
- Data manipulated by the parametrised system: theory $T_E = (\Sigma_E, \mathcal{C}_E)$
  Usually $\mathcal{C}_E$ contains just one structure: (products of) integers, reals, Booleans, control locations, ...
- We assume the availability of SMT solvers deciding the satisfiability of quantifier-free formulae modulo T_I and T_E

[2] A *theory* T is a pair $(\Sigma, \mathcal{C})$, where Σ is a first-order signature and $\mathcal{C}$ is a class of Σ-structures (the models of T).

Towards Array-based Systems

- Topology of the parametrised system: theory[2] $T_I = (\Sigma_I, \mathcal{C}_I)$
  E.g.: $\mathcal{C}_I$ consists of all (finite) sets, linear orders, forests/trees, graphs, ...
- Data manipulated by the parametrised system: theory $T_E = (\Sigma_E, \mathcal{C}_E)$
  Usually $\mathcal{C}_E$ contains just one structure: (products of) integers, reals, Booleans, control locations, ...
- We assume the availability of SMT solvers deciding the satisfiability of quantifier-free formulae modulo T_I and T_E

[2] A *theory* T is a pair $(\Sigma, \mathcal{C})$, where Σ is a first-order signature and $\mathcal{C}$ is a class of Σ-structures (the models of T).

Array-Based Systems: background theory A_I^E

- the sort `INDEX` is constrained by T_I;
- the sort `ELEM` is constrained by T_E;
- the sort `ARRAY` represents arrays of `ELEM` defined on `INDEX`;
- the 'read' operation _[_] is added to $\Sigma_I \cup \Sigma_E$;
- the class of models of A_I^E consists of the three-sorted structures whose reducts are models of T_I, T_E and the sort `ARRAY` is interpreted as the set of total functions from indexes to elements and the read operation is interpreted as function application

Array-Based Systems and their safety problem

- An array-based system on A_I^E with array state variable a is the following pair of formulae:

$$\mathcal{S} = \langle I(a), \tau(a, a') \rangle.$$

- A state of an array-based system is an assignment to the variable a in a model of A_I^E
- A safety problem for $\mathcal{S}$ is the following: given a formula $K(a)$, is

$$I(a_0) \wedge \tau(a_0.a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge K(a_n)$$

A_I^E-satisfiable for some n?

Array-Based Systems and their safety problem

- An array-based system on A_I^E with array state variable a is the following pair of formulae:

$$\mathcal{S} = \langle I(a), \tau(a, a') \rangle.$$

- A state of an array-based system is an assignment to the variable a in a model of A_I^E
- A safety problem for $\mathcal{S}$ is the following: given a formula $K(a)$, is

$$I(a_0) \wedge \tau(a_0.a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge K(a_n)$$

A_I^E-satisfiable for some n?

Array-Based Systems and their safety problem

- An array-based system on A_I^E with array state variable a is the following pair of formulae:

$$\mathcal{S} = \langle I(a), \tau(a, a') \rangle.$$

- A state of an array-based system is an assignment to the variable a in a model of A_I^E
- A safety problem for $\mathcal{S}$ is the following: given a formula $K(a)$, is

$$I(a_0) \wedge \tau(a_0.a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge K(a_n)$$

A_I^E-satisfiable for some n?

Revisiting Backward Reachability

Idea: recast symbolically the backward reachability algorithm

function BReach(K)
  $i \longleftarrow 0$; $BR^0(\tau, K) \longleftarrow K$; $K^0 \longleftarrow K$
  if $BR^0(\tau, K) \cap I \neq \emptyset$ **then return** unsafe
  repeat
    $K^{i+1} \longleftarrow \mathrm{Pre}(\tau, K^i)$
    $BR^{i+1}(\tau, K) \longleftarrow BR^i(\tau, K) \cup K^{i+1}$
    if $BR^{i+1}(\tau, K) \cap I \neq \emptyset$ **then return** unsafe
    else $i \longleftarrow i + 1$
  until $BR^{i+1}(\tau, K) \subseteq BR^i(\tau, K)$
  return safe
end

Revisiting Backward Reachability

Idea: recast symbolically the backward reachability algorithm

function BReach(K)
  $i \longleftarrow 0$; $BR^0(\tau, K) \longleftarrow K$; $K^0 \longleftarrow K$
  if A_I^E-check$(BR^0(\tau, K) \wedge I) = \text{sat}$ **then return** unsafe
  repeat
    $K^{i+1} \longleftarrow \text{Pre}(\tau, K^i)$
    $BR^{i+1}(\tau, K) \longleftarrow BR^i(\tau, K) \vee K^{i+1}$
    if A_I^E-check$(BR^{i+1}(\tau, K) \wedge I) = \text{sat}$ **then return** unsafe
    else $i \longleftarrow i + 1$
  until A_I^E-check$(\neg(BR^{i+1}(\tau, K) \rightarrow BR^i(\tau, K)) = \text{unsat}$
  return safe
end

But this is somehow problematic...

Revisiting Backward Reachability

Idea: recast symbolically the backward reachability algorithm

function BReach(K)
  $i \longleftarrow 0$; $BR^0(\tau, K) \longleftarrow K$; $K^0 \longleftarrow K$
  if A_I^E-check$(BR^0(\tau, K) \wedge I) = \text{sat}$ **then return** unsafe
  repeat
    $K^{i+1} \longleftarrow \text{Pre}(\tau, K^i)$
    $BR^{i+1}(\tau, K) \longleftarrow BR^i(\tau, K) \vee K^{i+1}$
    if A_I^E-check$(BR^{i+1}(\tau, K) \wedge I) = \text{sat}$ **then return** unsafe
    else $i \longleftarrow i + 1$
  until A_I^E-check$(\neg(BR^{i+1}(\tau, K) \rightarrow BR^i(\tau, K)) = \text{unsat}$
  return safe
end

But this is somehow problematic...

Key problems of the logical approach

- no termination (as expected);
- need to use quantifiers—both universal and existential—to express I, K in concrete applications;
- for τ, an alternation of quantifiers seems to be required (one typically needs an existential quantifier for a guard and an inner universal quantifier for the update);
  - as a consequence: no guarantee that satisfiability modulo A_I^E-checks (needed both for fix-point and safety) are effective;
  - no computationally reasonable implementation: e.g., if the satisfiability tests are r.e., then safety becomes Π_2^0 .....
- CHALLENGE: find syntactic restrictions on I, τ, K so as to make symbolic backward reachability feasible

Key problems of the logical approach

- no termination (as expected);
- need to use quantifiers—both universal and existential—to express I, K in concrete applications;
- for τ, an alternation of quantifiers seems to be required (one typically needs an existential quantifier for a guard and an inner universal quantifier for the update);
  - as a consequence: no guarantee that satisfiability modulo A_I^E-checks (needed both for fix-point and safety) are effective;
  - no computationally reasonable implementation: e.g., if the satisfiability tests are r.e., then safety becomes Π_2^0 .....
- CHALLENGE: find syntactic restrictions on I, τ, K so as to make symbolic backward reachability feasible

Key problems of the logical approach

- no termination (as expected);
- need to use quantifiers—both universal and existential—to express I, K in concrete applications;
- for τ, an alternation of quantifiers seems to be required (one typically needs an existential quantifier for a guard and an inner universal quantifier for the update);
  - as a consequence: no guarantee that satisfiability modulo A_I^E-checks (needed both for fix-point and safety) are effective;
  - no computationally reasonable implementation: e.g., if the satisfiability tests are r.e., then safety becomes Π_2^0 .....
- CHALLENGE: find syntactic restrictions on I, τ, K so as to make symbolic backward reachability feasible

Key problems of the logical approach

- no termination (as expected);
- need to use quantifiers—both universal and existential—to express I, K in concrete applications;
- for τ, an alternation of quantifiers seems to be required (one typically needs an existential quantifier for a guard and an inner universal quantifier for the update);
  - as a consequence: no guarantee that satisfiability modulo A_I^E-checks (needed both for fix-point and safety) are effective;
  - no computationally reasonable implementation: e.g., if the satisfiability tests are r.e., then safety becomes Π_2^0 .....
- CHALLENGE: find syntactic restrictions on I, τ, K so as to make symbolic backward reachability feasible

Key problems of the logical approach

- no termination (as expected);
- need to use quantifiers—both universal and existential—to express I, K in concrete applications;
- for τ, an alternation of quantifiers seems to be required (one typically needs an existential quantifier for a guard and an inner universal quantifier for the update);
  - as a consequence: no guarantee that satisfiability modulo A_I^E-checks (needed both for fix-point and safety) are effective;
  - no computationally reasonable implementation: e.g., if the satisfiability tests are r.e., then safety becomes Π_2^0 .....
- CHALLENGE: find syntactic restrictions on I, τ, K so as to make symbolic backward reachability feasible

Key problems of the logical approach

- no termination (as expected);
- need to use quantifiers—both universal and existential—to express I, K in concrete applications;
- for τ, an alternation of quantifiers seems to be required (one typically needs an existential quantifier for a guard and an inner universal quantifier for the update);
  - as a consequence: no guarantee that satisfiability modulo A_I^E-checks (needed both for fix-point and safety) are effective;
  - no computationally reasonable implementation: e.g., if the satisfiability tests are r.e., then safety becomes Π_2^0 .....
- CHALLENGE: find syntactic restrictions on I, τ, K so as to make symbolic backward reachability feasible

Requirements

Format of formulae

1. should be expressive enough to cover interesting classes of systems (mutual exclusion, broadcast, ...)
2. reduces all satisfiability tests to instances of quantifier-free combination problems
3. captures termination by *re-introducing model-theoretically* a notion of configuration and of a configuration pre-order
4. allows complete and automatic invariant synthesis

Requirements

Format of formulae

1. should be expressive enough to cover interesting classes of systems (mutual exclusion, broadcast, ...)
2. reduces all satisfiability tests to instances of quantifier-free combination problems
3. captures termination by *re-introducing model-theoretically* a notion of configuration and of a configuration pre-order
4. allows complete and automatic invariant synthesis

Requirements

Format of formulae

1. should be expressive enough to cover interesting classes of systems (mutual exclusion, broadcast, ...)
2. reduces all satisfiability tests to instances of quantifier-free combination problems
3. captures termination by *re-introducing model-theoretically* a notion of configuration and of a configuration pre-order
4. allows complete and automatic invariant synthesis

Requirements

Format of formulae

1. should be expressive enough to cover interesting classes of systems (mutual exclusion, broadcast, ...)
2. reduces all satisfiability tests to instances of quantifier-free combination problems
3. captures termination by *re-introducing model-theoretically* a notion of configuration and of a configuration pre-order
4. allows complete and automatic invariant synthesis

Formulae for initial sets of states

Proposed format for I: $\forall^I$-formulae

$$\forall \underline{i}\ \phi(\underline{i}, a[\underline{i}])$$

where $\underline{i}$ is a tuple of variables of sort `INDEX` and ϕ is a quantifier-free $\Sigma_I \cup \Sigma_E$-formula[3]

For instance, the formula $\forall i.\ a[i] = \texttt{idle}$ says that all processes are in state `idle`.

$\forall^I$-formulae can also be used to express invariants

[3] If $\underline{i} = i_1, \ldots, i_n$, then $a[\underline{i}]$ is the tuple of terms $a[i_1], \ldots, a[i_n]$ having sort `ELEM`.

Formulae for initial sets of states

Proposed format for I: $\forall^I$-formulae

$$\forall \underline{i}\ \phi(\underline{i}, a[\underline{i}])$$

where $\underline{i}$ is a tuple of variables of sort `INDEX` and ϕ is a quantifier-free $\Sigma_I \cup \Sigma_E$-formula[3]

For instance, the formula $\forall i.\ a[i] = \texttt{idle}$ says that all processes are in state `idle`.

$\forall^I$-formulae can also be used to express invariants

[3] If $\underline{i} = i_1, \ldots, i_n$, then $a[\underline{i}]$ is the tuple of terms $a[i_1], \ldots, a[i_n]$ having sort `ELEM`.

Formulae for initial sets of states

Proposed format for I: $\forall^I$-formulae

$$\forall \underline{i}\ \phi(\underline{i}, a[\underline{i}])$$

where $\underline{i}$ is a tuple of variables of sort `INDEX` and ϕ is a quantifier-free $\Sigma_I \cup \Sigma_E$-formula[3]

For instance, the formula $\forall i.\ a[i] = \texttt{idle}$ says that all processes are in state `idle`.

$\forall^I$-formulae can also be used to express invariants

[3] If $\underline{i} = i_1, \ldots, i_n$, then $a[\underline{i}]$ is the tuple of terms $a[i_1], \ldots, a[i_n]$ having sort `ELEM`.

Formulae for unsafe sets of states

Proposed format for K: $\exists^I$-formulae

$$\exists \underline{i}\ \phi(\underline{i}, a[\underline{i}])$$

where $\underline{i}$ is a tuple of variables of sort INDEX and ϕ is a quantifier-free $\Sigma_I \cup \Sigma_E$-formula.

For instance, the formula

$$\exists i_1 \exists i_2.\ (i_1 \neq i_2 \wedge a[i_1] = \mathtt{use} \wedge a[i_2] = \mathtt{use})$$

expresses that mutual exclusion is violated.

Formulae for unsafe sets of states

Proposed format for K: $\exists^I$-formulae

$$\exists \underline{i}\ \phi(\underline{i}, a[\underline{i}])$$

where $\underline{i}$ is a tuple of variables of sort INDEX and ϕ is a quantifier-free $\Sigma_I \cup \Sigma_E$-formula.

For instance, the formula

$$\exists i_1 \exists i_2.\ (i_1 \neq i_2 \wedge a[i_1] = \mathtt{use} \wedge a[i_2] = \mathtt{use})$$

expresses that mutual exclusion is violated.

Formulae for transitions

Proposed format for τ: we use disjunctions of formulae of the kind

$$\exists \underline{i} \left(\phi_L(\underline{i}, a[\underline{i}], a'[\underline{i}]) \wedge \forall j \, \phi_G(\underline{i}, a[\underline{i}], a'[\underline{i}], j, a[j], a'[j]) \right) \quad (1)$$

where ϕ_L, ϕ_G are quantifier-free $\Sigma_i \cup \Sigma_E$-formulae.

- ϕ_L is the local guard
- ϕ_G is the global update [4]

[4] ϕ_G is required to satisfy a 'seriality' requirement (as a consequence of our results, this seriality requirement turns out to be decidable).

Formulae for transitions: a refinement

In most cases, ϕ_L and ϕ_G do not depend on $a'[\underline{i}]$ and ϕ_G is a *case-definable* function F; hence (1) can be simplified to

$$\exists \underline{i} \left(\phi_L(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j \, F(\underline{i}, a[\underline{i}], j, a[j]) \right) \tag{2}$$

For instance, the formula (in simplified format)

$$\exists i. \left(a[i] = \texttt{use} \wedge a' = \lambda j \, (\mathit{if}\ j = i\ \mathit{then}\ \texttt{idle}\ \mathit{else}\ a[j]) \right)$$

is one of the disjunctions of the transition of the 'bakery' algorithm.

Formulae for transitions: a refinement

In most cases, ϕ_L and ϕ_G do not depend on $a'[\underline{i}]$ and ϕ_G is a *case-definable* function F; hence (1) can be simplified to

$$\exists \underline{i} \left(\phi_L(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j \, F(\underline{i}, a[\underline{i}], j, a[j]) \right) \tag{2}$$

For instance, the formula (in simplified format)

$$\exists i. \left(a[i] = \texttt{use} \wedge a' = \lambda j \, (\mathit{if}\ j = i\ \mathit{then}\ \texttt{idle}\ \mathit{else}\ a[j]) \right)$$

is one of the disjunctions of the transition of the 'bakery' algorithm.

Closure under pre-image

The following result guarantees that backward reachability can only produce $\exists^I$-formulae:

Theorem

Suppose that T_E has quantifier elimination; then if $H(a)$ is an $\exists^I$-formula, the formula

$$Pre(\tau, H) \;:=\; \exists a' \; (\tau(a, a') \wedge H(a'))$$

is A^E_I-equivalent to an effectively computable $\exists^I$-formula.

Quantifier elimination requirement is not needed if τ is in the simplified format (2).

Closure under pre-image

The following result guarantees that backward reachability can only produce $\exists^I$-formulae:

Theorem

Suppose that T_E has quantifier elimination; then if $H(a)$ is an $\exists^I$-formula, the formula

$$Pre(\tau, H) \;:=\; \exists a' \; (\tau(a, a') \wedge H(a'))$$

is A^E_I-equivalent to an effectively computable $\exists^I$-formula.

Quantifier elimination requirement is not needed if τ is in the simplified format (2).

Closure under pre-image

The following result guarantees that backward reachability can only produce $\exists^I$-formulae:

Theorem

Suppose that T_E has quantifier elimination; then if $H(a)$ is an $\exists^I$-formula, the formula

$$Pre(\tau, H) \;:=\; \exists a' \; (\tau(a, a') \wedge H(a'))$$

is A^E_I-equivalent to an effectively computable $\exists^I$-formula.

Quantifier elimination requirement is not needed if τ is in the simplified format (2).

Decidability of SMT problems for safety and fix-point checking

Theorem

Suppose that Σ_I does not contain function symbols[a] and that $\mathcal{C}_I$ is closed under substructures. Then the formulae of the kind

$$\exists \underline{i}\ \forall \underline{j}\ \psi(\underline{i}, \underline{j}, a[\underline{i}], a[\underline{j}]) \tag{3}$$

(where ψ is a quantifier-free $\Sigma_I \cup \Sigma_E$-formula) are decidable for A_I^E-satisfiability.

[a] More generally, that T_I is locally finite.

Remarks on decidability

- Assumptions on T_I are on the 'topology' of the parametrised system
- Closure under substructures means that you can 'delete' processes while maintaining the topology; e.g., deleting processes from a finite set, linear order, graph, forest, one still gets a finite set, linear order, graph, forest (but this is not true for rings)
- The proof of the theorem about decidability suggests a quantifier instantiation procedure, followed by purification and standard combination techniques

Remarks on decidability

- Assumptions on T_I are on the 'topology' of the parametrised system
- Closure under substructures means that you can 'delete' processes while maintaining the topology; e.g., deleting processes from a finite set, linear order, graph, forest, one still gets a finite set, linear order, graph, forest (but this is not true for rings)
- The proof of the theorem about decidability suggests a quantifier instantiation procedure, followed by purification and standard combination techniques

Remarks on decidability

- Assumptions on T_I are on the 'topology' of the parametrised system
- Closure under substructures means that you can 'delete' processes while maintaining the topology; e.g., deleting processes from a finite set, linear order, graph, forest, one still gets a finite set, linear order, graph, forest (but this is not true for rings)
- The proof of the theorem about decidability suggests a quantifier instantiation procedure, followed by purification and standard combination techniques

Configurations in the logical approach

- A configuration is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E
- Given s, the Σ_I-structure s_I is the Σ_I-reduct of $\mathcal{M}$ and the Σ_E-structure s_E is the smallest Σ_E-substructure of the Σ_E-reduct of $\mathcal{M}$ containing the image of s
- Let s, s' be configurations, we define a preorder as follows $s' \preceq s$ holds iff there exists
  - a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and
  - a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that
  - the set-theoretical compositions of μ with s and of s' with ν are equal

Configurations in the logical approach

- A configuration is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E
- Given s, the Σ_I-structure s_I is the Σ_I-reduct of $\mathcal{M}$ and the Σ_E-structure s_E is the smallest Σ_E-substructure of the Σ_E-reduct of $\mathcal{M}$ containing the image of s
- Let s, s' be configurations, we define a preorder as follows $s' \preceq s$ holds iff there exists
  - a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and
  - a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that
  - the set-theoretical compositions of μ with s and of s' with ν are equal

Configurations in the logical approach

- A configuration is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E
- Given s, the Σ_I-structure s_I is the Σ_I-reduct of $\mathcal{M}$ and the Σ_E-structure s_E is the smallest Σ_E-substructure of the Σ_E-reduct of $\mathcal{M}$ containing the image of s
- Let s, s' be configurations, we define a preorder as follows $s' \preceq s$ holds iff there exists
  - a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and
  - a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that
  - the set-theoretical compositions of μ with s and of s' with ν are equal

Configurations in the logical approach

- A configuration is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E
- Given s, the Σ_I-structure s_I is the Σ_I-reduct of $\mathcal{M}$ and the Σ_E-structure s_E is the smallest Σ_E-substructure of the Σ_E-reduct of $\mathcal{M}$ containing the image of s
- Let s, s' be configurations, we define a preorder as follows $s' \preceq s$ holds iff there exists
  - a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and
  - a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that
  - the set-theoretical compositions of μ with s and of s' with ν are equal

Configurations in the logical approach

- A configuration is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E
- Given s, the Σ_I-structure s_I is the Σ_I-reduct of $\mathcal{M}$ and the Σ_E-structure s_E is the smallest Σ_E-substructure of the Σ_E-reduct of $\mathcal{M}$ containing the image of s
- Let s, s' be configurations, we define a preorder as follows $s' \preceq s$ holds iff there exists
  - a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and
  - a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that
  - the set-theoretical compositions of μ with s and of s' with ν are equal

Configurations in the logical approach

- A configuration is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E
- Given s, the Σ_I-structure s_I is the Σ_I-reduct of $\mathcal{M}$ and the Σ_E-structure s_E is the smallest Σ_E-substructure of the Σ_E-reduct of $\mathcal{M}$ containing the image of s
- Let s, s' be configurations, we define a preorder as follows $s' \preceq s$ holds iff there exists
  - a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and
  - a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that
  - the set-theoretical compositions of μ with s and of s' with ν are equal

Wqo and Termination

Theorem

If $\preceq$ is a wqo, backward search terminates.

The above result is the symbolic version of the known result for well-structured transition systems.

If $\preceq$ is not a wqo, then termination is no more guaranteed and safety is undecidable in general. However... it might happen that safety can be certified by finding some invariant (see TABLEAUX talk wednesday ...!)

Wqo and Termination

Theorem

If $\preceq$ is a wqo, backward search terminates.

The above result is the symbolic version of the known result for well-structured transition systems.

If $\preceq$ is not a wqo, then termination is no more guaranteed and safety is undecidable in general. However... it might happen that safety can be certified by finding some invariant (see TABLEAUX talk wednesday ...!)

Wqo and Termination

Theorem

If $\preceq$ is a wqo, backward search terminates.

The above result is the symbolic version of the known result for well-structured transition systems.

If $\preceq$ is not a wqo, then termination is no more guaranteed and safety is undecidable in general. However... it might happen that safety can be certified by finding some invariant (see TABLEAUX talk wednesday ...!)

Initial experience

- Obvious client-server architecture
- Client generates proof obligations (satisfiability modulo theories problems)
- Server = state-of-the-art SMT solver (invoked via API)[5]
- Bottle-neck: huge number of calls to SMT solver had dramatic slow-down of system performances
- Need for techniques to significantly lower the number of invocations to SMT solver

[5] `Yices` is the SMT-solver employed in MCMT.

MCMT: first key problem

- To prevent implicit redundancies, backward reachable states are represented by disjunctions of primitive differentiated formulae

$$\exists i_1 \cdots \exists i_n \, (L_1(\underline{i}) \wedge \cdots \wedge L_m(\underline{i}) \wedge \bigwedge_{k<l} i_k \neq i_l)$$

where $L_1, \ldots, L_m$ are literals and $\underline{i} = i_1 \cdots i_n$.

- Fixpoint and safety tests amount to check consistency of sets like

$$\exists \underline{i} P, \ \neg \exists \underline{j}_1 Q_1, \cdots, \ \neg \exists \underline{j}_n Q_n,$$

where the existentially quantified formulae are all primitive differentiated.

MCMT: first key problem

- To prevent implicit redundancies, backward reachable states are represented by disjunctions of primitive differentiated formulae

$$\exists i_1 \cdots \exists i_n \, (L_1(\underline{i}) \wedge \cdots \wedge L_m(\underline{i}) \wedge \bigwedge_{k<l} i_k \neq i_l)$$

  where $L_1, \ldots, L_m$ are literals and $\underline{i} = i_1 \cdots i_n$.
- Fixpoint and safety tests amount to check consistency of sets like

$$\exists \underline{i} P, \ \neg \exists \underline{j}_1 Q_1, \cdots, \ \neg \exists \underline{j}_n Q_n,$$

  where the existentially quantified formulae are all primitive differentiated.

MCMT: first key problem

- According to our decision procedure, we should skolemize the $\underline{i}$ and instantiate the $\underline{j}_1, \ldots, \underline{j}_n$ to the $\underline{i}$ in all possible ways.
- Most such instances however do not contribute to inconsistency.
- To illustrate, suppose that

$$\begin{aligned} P(\underline{i}) &:= a[i_1] = c \wedge P'(\underline{i}) \\ Q_1(\underline{j}_1) &:= a[j_1] = d \wedge Q' \end{aligned}$$

Then, any instance containing $(j_1 \mapsto i_1)$ is useless, because

$$a[i_1] = c \wedge P'(\underline{i}) \wedge \neg(a[i_1] = d \wedge Q')$$

is equivalent to $a[i_1] = c \wedge P'(\underline{i})$, in case $T_E \models c \neq d$ (this is the case e.g. when c, d are distinct program locations).

MCMT: first key problem

- According to our decision procedure, we should skolemize the $\underline{i}$ and instantiate the $\underline{j}_1, \ldots, \underline{j}_n$ to the $\underline{i}$ in all possible ways.
- Most such instances however do not contribute to inconsistency.
- To illustrate, suppose that

$$\begin{aligned} P(\underline{i}) &:= a[i_1] = c \wedge P'(\underline{i}) \\ Q_1(\underline{j}_1) &:= a[j_1] = d \wedge Q' \end{aligned}$$

Then, any instance containing $(j_1 \mapsto i_1)$ is useless, because

$$a[i_1] = c \wedge P'(\underline{i}) \wedge \neg(a[i_1] = d \wedge Q')$$

is equivalent to $a[i_1] = c \wedge P'(\underline{i})$, in case $T_E \models c \neq d$ (this is the case e.g. when c, d are distinct program locations).

MCMT: first key problem

- According to our decision procedure, we should skolemize the $\underline{i}$ and instantiate the $\underline{j}_1, \ldots, \underline{j}_n$ to the $\underline{i}$ in all possible ways.
- Most such instances however do not contribute to inconsistency.
- To illustrate, suppose that

$$\begin{aligned} P(\underline{i}) &:= a[i_1] = c \wedge P'(\underline{i}) \\ Q_1(\underline{j}_1) &:= a[j_1] = d \wedge Q' \end{aligned}$$

Then, any instance containing $(j_1 \mapsto i_1)$ is useless, because

$$a[i_1] = c \wedge P'(\underline{i}) \wedge \neg(a[i_1] = d \wedge Q')$$

is equivalent to $a[i_1] = c \wedge P'(\underline{i})$, in case $T_E \models c \neq d$ (this is the case e.g. when c, d are distinct program locations).

MCMT: first key problem

- According to our decision procedure, we should skolemize the $\underline{i}$ and instantiate the $\underline{j}_1, \ldots, \underline{j}_n$ to the $\underline{i}$ in all possible ways.
- Most such instances however do not contribute to inconsistency.
- To illustrate, suppose that

$$\begin{aligned} P(\underline{i}) &:= a[i_1] = c \wedge P'(\underline{i}) \\ Q_1(\underline{j}_1) &:= a[j_1] = d \wedge Q' \end{aligned}$$

Then, any instance containing $(j_1 \mapsto i_1)$ is useless, because

$$a[i_1] = c \wedge P'(\underline{i}) \wedge \neg(a[i_1] = d \wedge Q')$$

is equivalent to $a[i_1] = c \wedge P'(\underline{i})$, in case $T_E \models c \neq d$ (this is the case e.g. when c, d are distinct program locations).

MCMT: solution to the first key problem

- Filtering modulo enumerated data-type theory
- Each primitive differentiated formula is decorated with the values of each array variables whose co-domain is an enumerated data-type (in case these values are cheaply available from formula reading)
- Such decorations are used to dynamically filter out useless instances in the incremental safety and the fix-point checks
- Also used to establish the executability of a transition:
  1. compute the pre-image: logically equivalent to a disjunction of primitive differentiated $\exists^I$-formulae
  2. for each of these formulae, search for a pair of literals of the form $\langle a[i] = c, a[i] = d \rangle$ and conclude non-executability
  3. if the search is fruitless, then invoke the SMT solver

MCMT: solution to the first key problem

- Filtering modulo enumerated data-type theory
- Each primitive differentiated formula is decorated with the values of each array variables whose co-domain is an enumerated data-type (in case these values are cheaply available from formula reading)
- Such decorations are used to dynamically filter out useless instances in the incremental safety and the fix-point checks
- Also used to establish the executability of a transition:
  1. compute the pre-image: logically equivalent to a disjunction of primitive differentiated $\exists^I$-formulae
  2. for each of these formulae, search for a pair of literals of the form $\langle a[i] = c, a[i] = d \rangle$ and conclude non-executability
  3. if the search is fruitless, then invoke the SMT solver

MCMT: solution to the first key problem

- Filtering modulo enumerated data-type theory
- Each primitive differentiated formula is decorated with the values of each array variables whose co-domain is an enumerated data-type (in case these values are cheaply available from formula reading)
- Such decorations are used to dynamically filter out useless instances in the incremental safety and the fix-point checks
- Also used to establish the executability of a transition:
  1. compute the pre-image: logically equivalent to a disjunction of primitive differentiated $\exists^I$-formulae
  2. for each of these formulae, search for a pair of literals of the form $\langle a[i] = c, a[i] = d \rangle$ and conclude non-executability
  3. if the search is fruitless, then invoke the SMT solver

MCMT: solution to the first key problem

- Filtering modulo enumerated data-type theory
- Each primitive differentiated formula is decorated with the values of each array variables whose co-domain is an enumerated data-type (in case these values are cheaply available from formula reading)
- Such decorations are used to dynamically filter out useless instances in the incremental safety and the fix-point checks
- Also used to establish the executability of a transition:
  1. compute the pre-image: logically equivalent to a disjunction of primitive differentiated $\exists^I$-formulae
  2. for each of these formulae, search for a pair of literals of the form $\langle a[i] = c, a[i] = d \rangle$ and conclude non-executability
  3. if the search is fruitless, then invoke the SMT solver

MCMT: solution to the first key problem

- Filtering modulo enumerated data-type theory
- Each primitive differentiated formula is decorated with the values of each array variables whose co-domain is an enumerated data-type (in case these values are cheaply available from formula reading)
- Such decorations are used to dynamically filter out useless instances in the incremental safety and the fix-point checks
- Also used to establish the executability of a transition:
  1. compute the pre-image: logically equivalent to a disjunction of primitive differentiated $\exists^I$-formulae
  2. for each of these formulae, search for a pair of literals of the form $\langle a[i] = c, a[i] = d \rangle$ and conclude non-executability
  3. if the search is fruitless, then invoke the SMT solver

MCMT: solution to the first key problem

- Filtering modulo enumerated data-type theory
- Each primitive differentiated formula is decorated with the values of each array variables whose co-domain is an enumerated data-type (in case these values are cheaply available from formula reading)
- Such decorations are used to dynamically filter out useless instances in the incremental safety and the fix-point checks
- Also used to establish the executability of a transition:
  1. compute the pre-image: logically equivalent to a disjunction of primitive differentiated $\exists^I$-formulae
  2. for each of these formulae, search for a pair of literals of the form $\langle a[i] = c, a[i] = d \rangle$ and conclude non-executability
  3. if the search is fruitless, then invoke the SMT solver

MCMT: solution to the first key problem

- Filtering modulo enumerated data-type theory
- Each primitive differentiated formula is decorated with the values of each array variables whose co-domain is an enumerated data-type (in case these values are cheaply available from formula reading)
- Such decorations are used to dynamically filter out useless instances in the incremental safety and the fix-point checks
- Also used to establish the executability of a transition:
  1. compute the pre-image: logically equivalent to a disjunction of primitive differentiated $\exists^I$-formulae
  2. for each of these formulae, search for a pair of literals of the form $\langle a[i] = c, a[i] = d \rangle$ and conclude non-executability
  3. if the search is fruitless, then invoke the SMT solver

MCMT: second key problem and its solution

- When computing the pre-image, the existential prefix grows
- To illustrate, let

$$\begin{aligned} P(a) &:= \exists \underline{k}.Q(\underline{k}, a[\underline{k}]) \\ \tau(a, a') &:= \exists \underline{i}.(G(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F(\underline{i}, a[\underline{i}], \underline{j}, a[\underline{j}])) \end{aligned}$$

  then

$$\exists \underline{i}, \underline{k}.(G(\underline{i}, a[\underline{i}]) \wedge Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])))$$

- Under suitable hypotheses, it is possible to statically pre-process the transition τ so as to lower the number of (or even to avoid adding) new existentially quantified variables (see our AVOCS'08 paper for details)
- (Notice that $Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))$ must be put into primitive differentiated form)

MCMT: second key problem and its solution

- When computing the pre-image, the existential prefix grows
- To illustrate, let

$$\begin{aligned} P(a) &:= \exists \underline{k}.Q(\underline{k}, a[\underline{k}]) \\ \tau(a, a') &:= \exists \underline{i}.(G(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F(\underline{i}, a[\underline{i}], \underline{j}, a[\underline{j}])) \end{aligned}$$

then

$$\exists \underline{i}, \underline{k}.(G(\underline{i}, a[\underline{i}]) \wedge Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])))$$

- Under suitable hypotheses, it is possible to statically pre-process the transition τ so as to lower the number of (or even to avoid adding) new existentially quantified variables (see our AVOCS'08 paper for details)
- (Notice that $Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))$ must be put into primitive differentiated form)

MCMT: second key problem and its solution

- When computing the pre-image, the existential prefix grows
- To illustrate, let

$$\begin{aligned} P(a) &:= \exists \underline{k}.Q(\underline{k}, a[\underline{k}]) \\ \tau(a, a') &:= \exists \underline{i}.(G(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F(\underline{i}, a[\underline{i}], \underline{j}, a[\underline{j}])) \end{aligned}$$

  then

$$\exists \underline{i}, \underline{k}.(G(\underline{i}, a[\underline{i}]) \wedge Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])))$$

- Under suitable hypotheses, it is possible to statically pre-process the transition τ so as to lower the number of (or even to avoid adding) new existentially quantified variables (see our AVOCS'08 paper for details)
- (Notice that $Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))$ must be put into primitive differentiated form)

MCMT: second key problem and its solution

- When computing the pre-image, the existential prefix grows
- To illustrate, let

$$\begin{aligned} P(a) &:= \exists \underline{k}.Q(\underline{k}, a[\underline{k}]) \\ \tau(a, a') &:= \exists \underline{i}.(G(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F(\underline{i}, a[\underline{i}], \underline{j}, a[\underline{j}])) \end{aligned}$$

  then

$$\exists \underline{i}, \underline{k}.(G(\underline{i}, a[\underline{i}]) \wedge Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}])))$$

- Under suitable hypotheses, it is possible to statically pre-process the transition τ so as to lower the number of (or even to avoid adding) new existentially quantified variables (see our AVOCS'08 paper for details)
- (Notice that $Q(\underline{k}, F(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))$ must be put into primitive differentiated form)

MCMT: additional features

- Current backward reachable states are maintained in two formulae lists: AV and TBV. Formulae from TBV are extracted for preimage computation and then moved into AV
- Forward redundancy elimination: each newly computed pre-image is tested for fix-point wrt the set of states in AV and TBV. Instances of negations of formulae in AV and TBV are incrementally added and the result is checked for satisfiability; we start from instances of more recent formulae (chronological fix-point checking)
- Backward redundancy elimination: formulae extracted from TBV are checked for fix-point before computing their pre-images: if the test is positive, they are deleted
- Invariant synthesis: candidate invariants (with only one index variable) are extracted from the set of reachable states and limited resource backward reachability attempts to show they are indeed invariants

MCMT: additional features

- Current backward reachable states are maintained in two formulae lists: AV and TBV. Formulae from TBV are extracted for preimage computation and then moved into AV
- Forward redundancy elimination: each newly computed pre-image is tested for fix-point wrt the set of states in AV and TBV. Instances of negations of formulae in AV and TBV are incrementally added and the result is checked for satisfiability; we start from instances of more recent formulae (chronological fix-point checking)
- Backward redundancy elimination: formulae extracted from TBV are checked for fix-point before computing their pre-images: if the test is positive, they are deleted
- Invariant synthesis: candidate invariants (with only one index variable) are extracted from the set of reachable states and limited resource backward reachability attempts to show they are indeed invariants

MCMT: additional features

- Current backward reachable states are maintained in two formulae lists: AV and TBV. Formulae from TBV are extracted for preimage computation and then moved into AV
- Forward redundancy elimination: each newly computed pre-image is tested for fix-point wrt the set of states in AV and TBV. Instances of negations of formulae in AV and TBV are incrementally added and the result is checked for satisfiability; we start from instances of more recent formulae (chronological fix-point checking)
- Backward redundancy elimination: formulae extracted from TBV are checked for fix-point before computing their pre-images: if the test is positive, they are deleted
- Invariant synthesis: candidate invariants (with only one index variable) are extracted from the set of reachable states and limited resource backward reachability attempts to show they are indeed invariants

MCMT: additional features

- Current backward reachable states are maintained in two formulae lists: AV and TBV. Formulae from TBV are extracted for preimage computation and then moved into AV
- Forward redundancy elimination: each newly computed pre-image is tested for fix-point wrt the set of states in AV and TBV. Instances of negations of formulae in AV and TBV are incrementally added and the result is checked for satisfiability; we start from instances of more recent formulae (chronological fix-point checking)
- Backward redundancy elimination: formulae extracted from TBV are checked for fix-point before computing their pre-images: if the test is positive, they are deleted
- Invariant synthesis: candidate invariants (with only one index variable) are extracted from the set of reachable states and limited resource backward reachability attempts to show they are indeed invariants

MCMT: experiments

	depth	#nodes	#calls	time (sec.)
Bakery	9	28	252	0.124
Bakery (buggy)	8	90	1419	0.844
Burns	14	57	429	0.188
Java M-lock	9	23	343	0.152
Dijkstra	13	38	256	0.096
Dijkstra (rv)	14	127	4451	3.324
Szymanski	17	136	3164	2.036
Szymanski (a)	23	1745	475505	11m19s

Table: Mutual exclusion algorithms

	depth	#nodes	#calls	time (sec.)
Berkeley	2	1	103	0.020
Mesi	3	2	177	0.028
Moesi	3	2	306	0.048
Dec Firefly	4	4	167	0.052
Xerox P.D.	7	13	621	0.320
Illinois	4	8	1006	0.216
Futurebus	4	19	1355	0.476
German (original)	26	2442	126060	3m51s
German (buggy)	16	1631	43952	1m33s
German (pfs-2/3)	38	15556	1837228	83m3s
German (pfs-3/3)	42	22144	1353118	121m27s

Table: Cache coherence protocols

MCMT: experiments

	depth	#nodes	#calls	time (sec.)
Mesi	3	2	18	0.008
Moesi	3	2	21	0.016
Illinois	3	4	82	0.036
German (ca)	9	13	46	0.028

Table: Cache coherence protocols: counting abstraction

	# MCMT nodes	MCMT time	PFS time
Burns	57	0.188	**0.064**
Java M-lock	23	0.152	**0.008**
Dijkstra	38	**0.096**	0.644
Szymanski	136	**2.036**	2.192
Mesi	2	0.008	**0.000**
Moesi	2	0.016	**0.000**
Illinois	4	0.036	**0.024**
German (pfs-2/3)	15556	**83m3s**	124m13s

Table: MCMT vs PFS[6]

[6] Due to the lack of an established standard, comparison might be of relative value.

MCMT: availability

- Free download at

  http://homes.dsi.unimi.it/~ghilardi/mcmt

- Executables for Linux (Ubuntu 8.04) and Mac OSX
- Sets of benchmarks about distributed systems (mutual exclusion algorithms, cache coherence protocols) but also insertion-sort, ...
- Papers about the theoretical framework, invariant synthesis, and various techniques implemented in the system

MCMT: sample of input format (from Illinois protocol)

```
:transition
:var x
:var j
:guard (= a[x] 1)
:numcases 2
:case (= x j)
:val 2
:case (not (= x j))
:val 1
```

```
:transition
:var x
:var j
:guard (= a[x] 2)
:numcases 2
:case (= x j)
:val 1
:case (not (= x j))
:val a[j]
```

```
:unsafe
:var x
:var y
:cnj (= a[x] 2) (> a[y] 1) (< a[y] 4)
```

```
:initial
:var x
:cnj (= a[x] 1)
```

References

P. A. Abdulla, K. Cerans, B. Jonsson, Y. K. Tsay

General Decidability Theorems for Infinite State Systems,
In *Proc. of LICS 1996*, pp. 313–321, 1996, IEEE Computer Society.

P. A. Abdulla, N. Ben Henda, G. Delzanno, A. Rezine.

Regular model checking without transducers (on efficient verification of parameterized systems).
In *Proc. of TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pp. 721–736, 2007, Springer.

P. A. Abdulla, G. Delzanno, A. Rezine.

Parameterized verification of infinite-state processes with global conditions.
In *Proc, of CAV 2007*, volume 4590 of *Lecture Notes in Computer Science*, pages 145–157, 2007, Springer.

P. A. Abdulla, N. Ben Henda, G. Delzanno, A. Rezine.

Handling parameterized systems with non-atomic global conditions.
In *Proc. of VMCAI 2008*, volume 4905 of *Lecture Notes in Computer Science*, pages 22–36, 2008, Springer.

P. A. Abdulla, N. Ben Henda, G. Delzanno, F. Haziza, A. Rezine.

Parameterized Tree Systems.
In *Proc. of FORTE 2008*, *Lecture Notes in Computer Science*, 2008, Springer.

References

P. A. Abdulla, B. Jonsson.

Verifying programs with unreliable channels.
Information and Computation, 127(2):91–101, 1996.

G. Delzanno, J. Esparza, A. Podelski.

Constraint-based analysis of broadcast protocols.
In *Proc. of CSL 1999*, volume 1683 of *Lecture Notes in Computer Science*, pages 50–66, 1999, Springer.

A. Bouajjani, P. Habermehl, Y. Yurski, and M. Sighireanu.

Rewriting systems with data.
In *Proc. of Symp. on Fund. of Comp. Th. (FCT 07)*, pages 1–22, 2007.

J. Esparza, A. Finkel, R. Mayr.

On the verification of broadcast protocols.
In *Proc. of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS 1999)*, pages 352–359, 1999, IEEE Computer Society.

S. Ghilardi, E. Nicolini, S. Ranise, D. Zucchelli.

Combination methods for satisfiability and model-checking of infinite-state systems.
In *Proc. of CADE 2007*, volume 4603 of *Lecture Notes in Computer Science*, pages 362–378, 2007, Springer.

References

S. Ghilardi, S. Ranise, and T. Valsecchi.

Light-Weight SMT-based Model-Checking.
In *Proc. of AVOCS 07-08*, ENTCS, 2008.

S. Ghilardi and S. Ranise.

Goal-Directed Invariant Synthesis in Model Checking Modulo Thoeries.
In *Proc. of TABLEAUX 09*, LNCS, 2009.

S. Ghilardi and S. Ranise.

Model Checking Modulo Theory at work: the intergration of Yices in MCMT.
In *Proc. of AFM 09*, ACM Digital Library, 2009.

T. Rybina, A. Voronkov.

A logical reconstruction of reachability.
In M. Broy and A. V. Zamulin, editors, *Revised Papers of the 5th International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics (PSI 2003)*, volume 2890 of *Lecture Notes in Computer Science*, pages 222–237, Springer, 2003.

MCMT,

The Model Checker Modulo Theories,
http://homes.dsi.unimi.it/~ghilardi/mcmt/.