A Machine-Checked Proof of a State-Space Construction Algorithm

Nestor Catano[1] Radu I. Siminiceanu[2]

[1]University of Madeira, CMU-Portugal

[2]National Institute of Aerospace, Hampton, Virginia, US

NFM 2010

Outline

What is Saturation ?

A symbolic algorithm for building state spaces of discrete systems

At-a-glance

Not your "typical" reachability algorithm

- not focused on states, but on (decision diagram) nodes
- *"chaotic"* fixed points
- other *"non-standard"* data structures and conventions
  - QOMDDs, Kronecker matrices
  - storage: index-based nodes, partitioned caches/unique tables, ...
  - first implemented in the SMART tool $\Rightarrow$ Petri nets
- **much faster and leaner than Breadth-First-Search (BFS)**

The Saturation Algorithm

Brief History

Evolved from standard BDD algorithm for state-space construction

- 1999: exploit event locality, MDDs
- 2000: exploit Kronecker decomposition, chaining
- 2001: new iteration strategy (named **Saturation**)
- 2002: on-the-fly (a.k.a. "unbound") version
- 2003: build counterexamples (+ EVMDDs)
- 2004: other extensions, generalizations
  - lift Kronecker requirements (i.e. back to diagrams for tr. rel.)
  - identity reduced MDDs
  - partial reachability (with EVMDDs)
- 2009: C library

Saturation: Background and origins

BFS for Discrete state systems: $(\mathcal{S}, \mathcal{S}^0, \mathcal{N})$

- $\mathcal{S}^0 \in \mathcal{S}$: initial state(s)
- $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$, the next-state function

Standard iteration strategy: $\mathcal{S} = \mathcal{S}^0 \cup \mathcal{N}(\mathcal{S}^0) \cup \mathcal{N}^2(\mathcal{S}^0) \cup \ldots$

Weaknesses:

- function $\mathcal{N}$ is monolithic
- applied to the entire current set of states

Kronecker Consistency

- How to improve?
  - split the transition function: $\mathcal{N} = \bigcup_{e \in \mathcal{E}} \mathcal{N}_e$
  - further split events by subsystem (level): $\mathcal{N}_e = \mathcal{N}_{e,K} \times \ldots \times \mathcal{N}_{e,1}$
  - $Top(e) = l$
- What it does?
  - Speeds up algorithm
    - allows one to operate on sub-states
    - fire below, then concatenate with above: perfectly "legal"
  - Makes prover's life miserable

Kronecker Consistency

- Origin: Kronecker product of (transition probability) matrices:

$$\sum_{e \in \mathcal{E}} \bigotimes_{K \geq k \geq 1} \mathcal{N}_{e,k}$$

- What is it? independence of local effects of $\mathcal{N}_{e,k}$ (by level k)

$$\mathcal{N}_e(s) = \mathcal{N}_{e,K}(s_K) \times \mathcal{N}_{e,K-1}(s_{K-1}) \times \ldots \times \mathcal{N}_{e,1}(s_1)$$

Saturation: Description

A node p at level k is **saturated** if
it encodes a fixed point w.r.t. events e with $Top(e) \leq k$

$$Below(k,p) = \mathcal{N}^{*}_{\leq k}(Below(k,p))$$

Algorithm in a nutshell:

- build the MDD encoding of $\mathcal{S}^0$
- for $k = 1$ upto K, saturate all nodes at level k:
  $\Rightarrow$ *exhaustively* fire events e with $Top(e) = k$ (transitive closure)
- if a firing creates nodes at levels below k:
  $\Rightarrow$ saturate them immediately upon creation
- when the root node is saturated: *voilà*, the full state space

NB: mutual recursion between saturation and firing of events

The two core routines: *Fire()* and *Saturate()*

Simplified/sanitized pseudo-code:

Saturate(k,p)

1. do
2. foreach $e : Top(e) = k$ do
3. Fire(e,k,p)
4. while new states discovered

Fire(e,k,p)

1. foreach local transition $i_k \xrightarrow{e} j_k$
2. f = Fire(e, $k-1$, child(k,p,i_k))
3. if $f \neq \emptyset$
4. Saturate($k-1$,f)
5. $u = Union$(k-1, f, child(k,p,j_k))
6. set child(k,p,j_k) $\leftarrow u$

NB: Auxiliary structures (caches, unique tables) and other MDD node management routines are not captured here

The Correctness Theorem

Original statement (Simplified Version)

" Let $\langle k|p\rangle$ be a node with saturated children, and $\langle l|q\rangle$ be one of its children with $q \neq 0$ and $l = k-1$;

- let $\mathcal{U}$ stand for *Below*(l, q) before the call to *Fire*(e, l, q), for some event e with $l <$ *Top*(e)
- let $\mathcal{V}$ represent *Below*(l, f), where $f =$ *Fire*(e, l, q);

$\Rightarrow$ Then, $\mathcal{V} = \mathcal{N}^*_{\leq l}(\mathcal{N}_e(\mathcal{U}))$."

PVS Formalisation

PVS

- PVS is a system for specifying and verifying properties of software and hardware systems.
- PVS' logic is based on simply typed higher-order logic with functions, product types, records, and recursive definitions.
- PVS' type systems is extended with sub-types.

Abuse(s) of notation

- $\mathcal{N}_e(\mathcal{U})$ is an abuse : because $Top(e) = k > l$
- Semantics of $\mathcal{N}$ is actually not next-state
- PVS discovered these "inconsistencies"

Challenge

- PVS does not directly provide support to mutually recursive functions definitions.

PVS Formalisation

General Approach

- Formalize basic concepts related to Kronecker consistency.
  - Events, MDDs, states, local states, next-state functions, local next-state functions, etc.
- Use definitions to formalize *Saturate*() and *Fire*() to reflect in Logic invariant properties of Saturation algorithm
- Conduct proofs following the pencil-and-paper proof

PVS Formalization

Formalizing Basic Concepts

$$\mathtt{local\_value?(m)(n)} : \mathtt{bool} = (\mathtt{m} = 0 \wedge \mathtt{n} = 0) \vee (\mathtt{m} > 0 \wedge \mathtt{n} > 0 \wedge \mathtt{n} \leq \mathtt{nk(m)})$$

$$\mathtt{local\_value(m)} : \mathtt{type} = (\mathtt{local\_value?(m)})$$

$$\mathtt{state(k)} : \mathtt{type} = \{\mathtt{s} : \mathtt{Seq(k)} \mid \forall (\mathtt{m} : \mathtt{upto(k)}) : (\mathtt{m} = 0 \wedge \mathtt{s`sq(m)} = 0) \vee (\mathtt{m} > 0 \wedge \mathtt{s`sq(m)} > 0 \wedge \mathtt{s`sq(m)} \leq \mathtt{nk(m)})\}$$

Formalizing Basic Concepts

$$\mathtt{event} : \mathtt{type+}$$

$$\mathtt{Top} : [\mathtt{event} \rightarrow \mathtt{posnat}]$$

PVS Formalization

Formalizing Basic Concepts

$$\mathtt{next(k)} : \mathtt{type} = [\mathtt{event} \rightarrow [\mathtt{state(k)} \rightarrow \mathtt{setof[state(k)]}]]$$

$$\mathtt{Localnext(k)} : \mathtt{type} = [\mathtt{event} \rightarrow [\mathtt{local\_value(k)} \rightarrow \mathtt{setof[local\_value(k)]}]]$$

$$\begin{aligned}&\mathtt{Kronecker?(k)(N)(fs)} : \mathtt{bool} = \\ &\ \forall(\mathtt{e} : \mathtt{event}, \mathtt{x}, \mathtt{y} : \mathtt{state(k)}) : \\ &\quad \mathtt{N(e)(x)(y)} \Leftrightarrow \forall(\mathtt{m} : \mathtt{upto(k)}) : \mathtt{fs`sq(m)(e)(x`sq(m))(y`sq(m))}\end{aligned}$$

Axiomatic vs Definition Approach

Discussion

- Definitions do not introduce inconsistencies.
- The use of definitions may force PVS to generate additional proof obligations all over the theorems and lemmas using the definitions, cluttering the proofs.
- Definitions may require to reflect the implementation of the algorithm in Logic.

Axiomatic vs Definition Approach

Discussion

- Definitions do not introduce inconsistencies.
- The use of definitions may force PVS to generate additional proof obligations all over the theorems and lemmas using the definitions, cluttering the proofs.
- Definitions may require to reflect the implementation of the algorithm in Logic.

Axiomatic vs Definition Approach

Discussion

- Axioms might introduce inconsistencies.
- Axioms are more suitable than definitions when one is not interested in generating code.

Axiomatic vs Definition Approach

Discussion

- Axioms might introduce inconsistencies.
- Axioms are more suitable than definitions when one is not interested in generating code.

Axiomatic vs Definition Approach

Lessons learned from this exercise

- Non-axiomatic approach was attempted first
- Too many TCCs, mostly from subtyping conditions
- Forced to change definitions, add more (type-dependent) parameters, etc.
- This introduces more TCCs ...
- Proof becomes unmanageable

Formalizing *Saturate*()

Is node $\langle k|p\rangle$ saturated?

$$Below(k,p) = \mathcal{N}^{*}_{\leq k}(Below(k,p))$$

Is node $\langle k|p\rangle$ saturated?

$$\begin{aligned}&\mathtt{saturated?(k : upto(L),\ p : OMDD)}\\ &\qquad\mathtt{(N : next(k),\ fs : (kronecker?(k)(N)))(w : nat) : bool =}\\ &\quad\mathtt{Below(k,p) = Apply(k)(N,fs)(w)(Below(k,p))}\end{aligned}$$

Reaching the Fixed-Point

- w represents the number of iterations after which $\mathcal{N}^{*}_{\leq k}(Below(k,p))$ does not generate any new state.
- The existence of w is guaranteed by the finiteness of $Below(k,p)$ and because the firing of any $\mathcal{N}_e \in \mathcal{N}_{\leq k}$ is an increasing function (the set of reached states gets larger)

Formalizing *Fire*()

Recursion's Base Case

$\mathtt{fire}(\mathtt{l}, \mathtt{e}, \mathtt{N}, \mathtt{fs}, \mathtt{w}, \mathtt{q}) : \mathtt{OMDD}$

$\mathtt{fire\_trivial} : \mathtt{axiom}$

$\mathtt{l} < \mathtt{Bottom}(\mathtt{e}) \lor \mathtt{l} = 0 \Rightarrow \mathtt{fire}(\mathtt{l}, \mathtt{e}, \mathtt{N}, \mathtt{fs}, \mathtt{w}, \mathtt{q}) = \mathtt{q}$

Formalizing *Fire*()

Recursive Case

Fire() is recursively called on their children `child(q,i)`

Recursive Case

$$\mathtt{fire\_recursive} : \mathtt{axiom}$$
$$\mathtt{Below}(\mathtt{l}, \mathtt{fire}(\mathtt{l}, \mathtt{e}, \mathtt{N}, \mathtt{fs}, \mathtt{w}, \mathtt{q})) =$$
$$\{\mathtt{s} : \mathtt{state}(\mathtt{l}) \mid \exists(\mathtt{i} : \mathtt{local\_value}(\mathtt{l})) : \mathtt{fs}`\mathtt{sq}(\mathtt{l})(\mathtt{e})(\mathtt{i})(\mathtt{s}`\mathtt{sq}(\mathtt{l})) \wedge$$
$$\mathtt{Below}(\mathtt{l\text{-}1}, \mathtt{fire}(\mathtt{l\text{-}1}, \mathtt{e}, \mathtt{N_l}', \mathtt{fs_l}', \mathtt{w}, \mathtt{child}(\mathtt{q}, \mathtt{i})))(\mathtt{s_t}(\mathtt{l}, \mathtt{s}, \mathtt{l\text{-}1}))\}$$

Formalizing *Fire*()

Mutual Recursion *Fire*() vs *Saturate*()

Fire() is always invoked on a saturated node $\langle l|q\rangle$ with $l < Top(e)$ and *Saturate*() is invoked just before returning from *Fire*().

Mutual Recursion *Fire*() vs *Saturate*()

$\texttt{fire\_saturated} : \texttt{axiom}$
$\quad \texttt{saturated?}(\texttt{l}, \texttt{fire}(\texttt{l}, \texttt{e}, \texttt{N}, \texttt{fs}, \texttt{w}, \texttt{q}))(\texttt{N}, \texttt{fs})(\texttt{w})$

The PVS Correctness Proof

Auxiliary Result

$$\mathcal{N}^*_{\leq k-1}(\mathcal{N}_e(\mathcal{B}(k,p))) = \bigcup_{i \in S^k} \mathcal{N}^k_e(i) \times \mathcal{N}^*_{\leq k-1}(\mathcal{N}_e(\mathcal{B}(\langle k-1 | \langle k|p\rangle[i]\rangle)))$$

Auxiliary Result

$\texttt{kronecker\_apply: }$ $\textcolor{red}{\texttt{theorem}}$
$\quad \texttt{Apply(k,k-1)(N,fs)(w)(Next(k,ev)(N,fs)(Below(k,p)))(s)}$
$\qquad \Leftrightarrow$
$(\exists(\texttt{i} : \texttt{local\_value}(\texttt{k})) :$
$\quad \texttt{fs`sq(k)(ev)(i)(s`sq(k))} \ \wedge$
$\quad \texttt{Apply}(\texttt{k}-1)(\texttt{N}'_{\texttt{k}}, \texttt{fs}'_{\texttt{k}})(\texttt{w}_1)(\texttt{Next}(\texttt{k}-1, \texttt{ev})(\texttt{N}'_{\texttt{k}}, \texttt{fs}'_{\texttt{k}})($
$\qquad \texttt{Below}(\texttt{k-1}, \texttt{child}(\texttt{p}, \texttt{i}))))(\texttt{s}_{\texttt{t}}(\texttt{k}, \texttt{s}, \texttt{k-1})))$

General Idea

- Induction on `w`
- Kronecker Consistency

The PVS Correctness Proof

The PVS Correctness Proof

- let $\mathcal{U}$ stand for $Below(l, q)$ before the call to $Fire(e, l, q)$, for some event e with $l < Top(e)$
- let $\mathcal{V}$ represent $Below(l, f)$, where $f = Fire(e, l, q)$;

$\Rightarrow$ Then, $\mathcal{V} = \mathcal{N}^*_{\leq l}(\mathcal{N}_e(\mathcal{U}))$."

The PVS Correctness Proof

```
saturation_correctness: theorem
  Below(l,fire(l,e,N,fs,w,q)) =
    Apply(l)(N,fs)(w)(Next(l,e)(N,fs)(Below(l,q)))
```

General Idea

- Kronecker consistency
- Finite domains, increasing functions
- Induction on `l`

Proof Statistics

Statistics

- 10 Theories
- 145 Proofs
- 10 Lemmas
- 2 main Theorems

http://www.uma.pt/ncatano/satcorrectness/saturation-proofs.htm

Conclusion and Future Work

Code Generation

- Porting PVS theories to B Machines.
- Using refinement tools (e.g., AtelierB) to generate certified C code.
- Benchmarking (parts of) generated code with existing implementation in SMART.

Questions?