Undergraduate work

Symbolic Model Checking Using Additive Decomposition

by

Himanshu Jain

Joint work with

Supratik Chakraborty

Organization of the Talk

- Basics
- Motivation
- Related work
- Decomposition scheme
- Using the decomposition
- Future work

Basics

- Kripke structure
- CTL formula

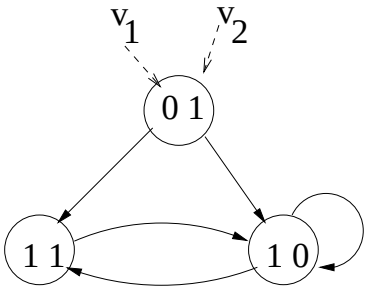


Figure 1: Kripke structure

- Consider a formula $f = (v_1 \wedge \neg v_2)$
- Specification: $\neg v_1 \wedge v_2 \models AFf$

Motivation

- Symbolic representation techniques (BDDs) have been shown to handle state transition graphs with 10^{120} states. ([3])
- For many practical circuits the problem of memory explosion exists.
- One of the causes for memory explosion is quantification over next state which arises in pre-image computation
- $N(V) = \exists V'(T(V, V') \wedge R(V'))$

Related Work

- Specialized algorithm for pre-image computation [6]. Also known as *AndAbstract* operation in BDD packages.
- Partitioned transition relations [3]
- Quantifier scheduling [5]
- Quantifier elimination in sequential circuits [8], [1]

Related Work contd.

- For sequential circuits $v_i' \Leftrightarrow f_i(V, I)$
- Transition relation $\bigwedge_{i=1}^{n}(v_i' \Leftrightarrow f_i(V, I))$
- $\mathbf{EX}Z = \exists I(Z(f_1, \ldots, f_n))$ [7]
- Replacement of quantification by substitution is a useful optimization [8]
- Works well only if number of input variables are less

Related work contd.

- Chakraborty et al. [4] came up with a new decomposition scheme.
- They decompose a Kripke structure into a set of *components*.
- For now assume we have a monolithic transition relation $T(V, V')$.

Component of a Kripke structure

- Each state has exactly one outgoing transition which is present in original Kripke struture

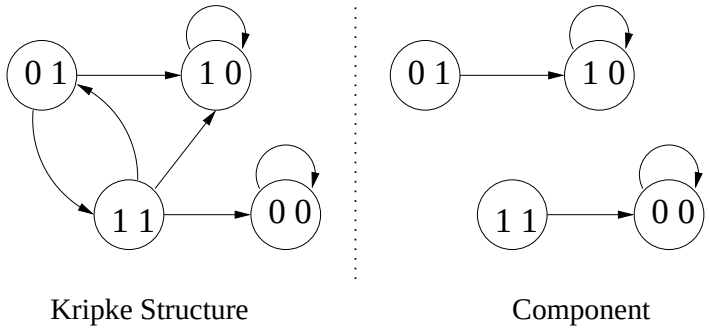


Figure 2: Component example

- For a component $v_i' = f_i(V)$
- In above example $v_1' = v_1 \oplus v_2$ and $v_2' = 0$

Properties of a component

- Pre-image of a set of states $Z(V)$ is given by
- $\mathbf{EX}Z = Z(f_1, \ldots, f_n)$
- No quantification is involved when doing pre-image computation on a component.
- Note vector composition is needed. Costly operation in a BDD based implementation.
- Path quantifiers **A** and **E** are same for a component.
- How do we generate a component?
- Is it costly?

Decomposition of a Kripke structure

- Is a set of components of a given Kripke structure.
- Complete decomposition
- Minimal decomposition
- Partial decomposition

Number of components

- Minimal decomposition requires exactly maximum outdegree number of components

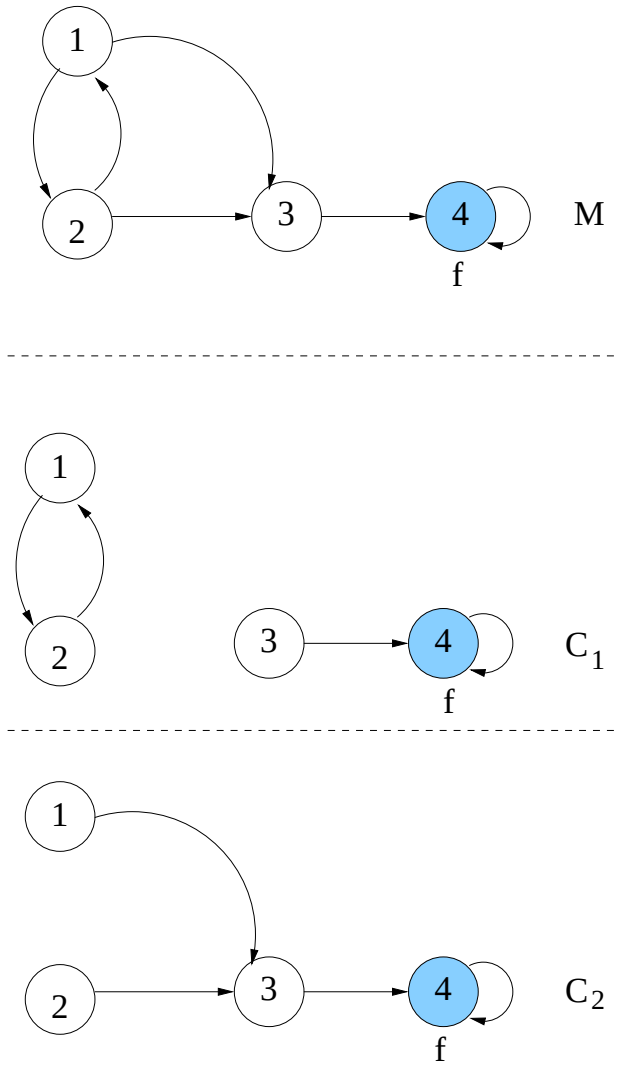


Figure 3: Minimal number of components

Related work

- Chakraborty et al. [4] showed how to do reachability analysis (**EF**) computation using partial decomposition.
- We show how to do complete **CTL** model checking using partial decomposition.
- Our algorithm will generate a minimal decomposition only in the worst case.
- For sequential circuits (ISCAS89) number of components generated were $\leq$ than 4.
- From now on we will concentrate on calculation of $\mathbf{AX}Z$ operator.

Example 1

- $\mathbf{AX}Z = \{3, 4\}$
- $\mathbf{AX}^1 Z = \{3, 4\} \wedge \mathbf{AX}^2 Z = \{1, 3, 4\}$

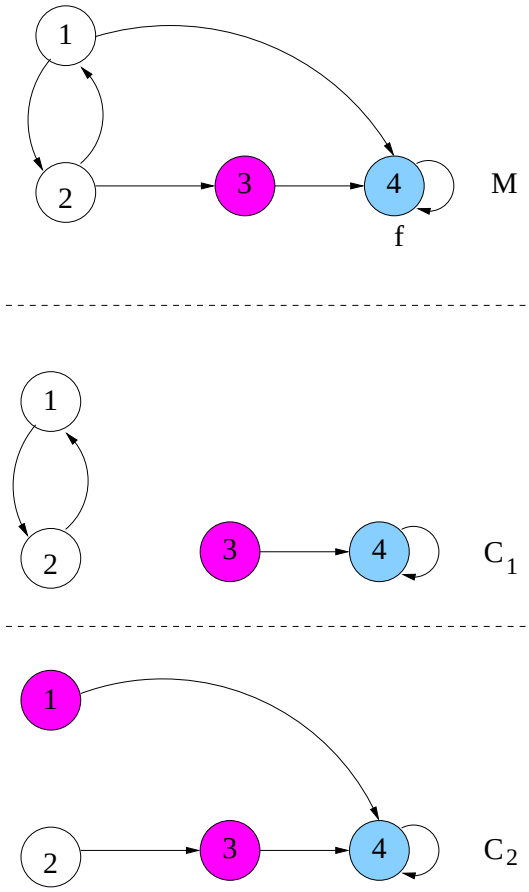


Figure 4: **AX**Z calculation using minimal decomposition

Important results

- Given a Kripke structure $M = (S, T, L)$.
- Let a minimal decomposition of M be $\{C_1, \ldots, C_k\}$.
- $\mathbf{EX}Z = \bigvee_{i=1}^{k} \mathbf{EX}^i Z$
- $\mathbf{AX}Z = \bigwedge_{i=1}^{k} \mathbf{AX}^i Z = \bigwedge_{i=1}^{k} \mathbf{EX}^i Z$
- Do these results seem familiar?
- Disjunctive partitioning [3]

Using minimal decomposition (Pro)

- We can do **CTL** model checking.
- Number of components needed is exactly equal to maximum outdegree.
- Only one component needs to remain in memory at a time.

Using a minimal decomposition (Cons)

- We need to generate a minimal decomposition in the begining itself.
- It is costly if the maximum outdegree is large.
- We will give an algorithm that generates a minimal decomposition in the worst case.

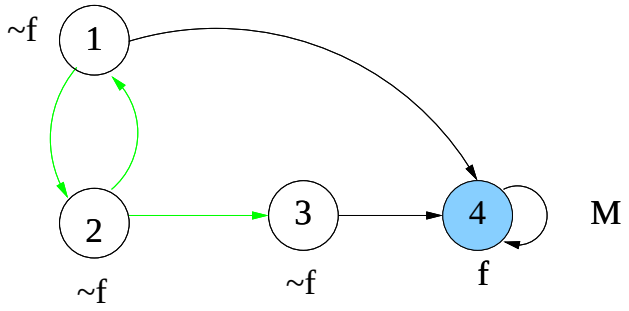
Basic intuition

- Do we need to generate both C_1 and C_2 for finding $\mathbf{AX}Z$?
- Can we generate C_1 directly?

Important result

- Given a partial decomposition D and a set of states Z.
- We wan't to calculate set of states satisfying $\mathbf{AX}Z$.
- Say D has m components. Note m can be 0.
- In worst case we will need to generate one more component to compute $\mathbf{AX}Z$.

Example 2



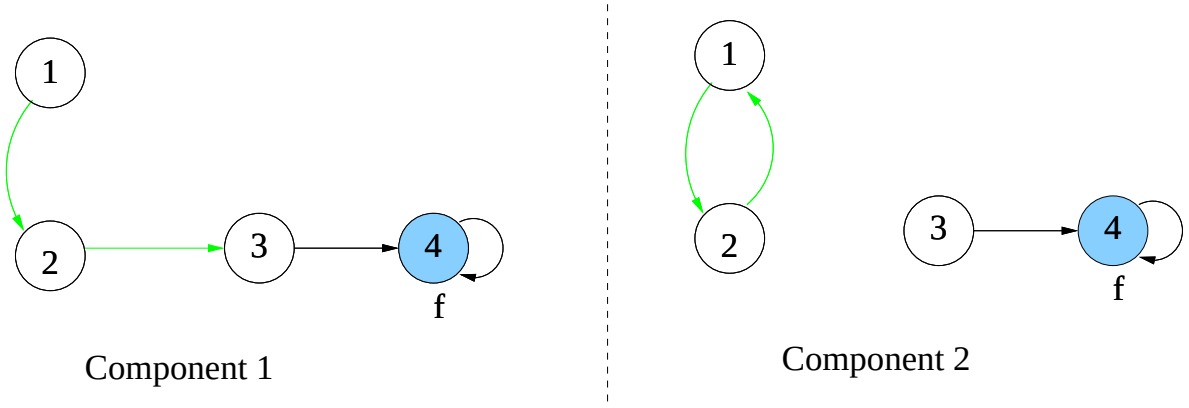


Figure 5: Preferred transitions

- $\mathbf{AX}Z = \{3, 4\}$
- $\mathbf{AX}^1 Z = \{3, 4\}$ and $\mathbf{AX}^2 Z = \{3, 4\}$

Handling $m = 0$ case

- Given M and a set of states Z and an empty decomposition of M.
- How do we generate the first component C_1 such that $\mathbf{AX}Z = \mathbf{AX}^1 Z$?
- Calculate a set of preferred transitions.
- $P(V, V') = T(V, V') \wedge \neg Z(V')$
- When generating C_1 try to pick transitions from preferred set of transitions.

CTL model checking algorithm

- Every **CTL** operator can be expressed as a least fixed point or greatest fixed point of $\mathbf{AX}Z$.
- For each $\mathbf{AX}Z$ calculation create a new component.
- Bound on number of components?
- We have better way to utilize previously generated components.
- We stop when a minimal decomposition has been generated.

Component generation

- Given a preferred set of transitions $P(V, V')$ and transition relation $T(V, V')$.
- Generate a component C giving priority to preferred transitions.
- Basic idea express v'_n as a function of $h_n(X)$ where $X = \langle v_1, \ldots, v_n, v'_1, \ldots, v'_{n-1} \rangle$.
- $h_n(X) = \neg P(X, 0) \wedge T(X, 1)$
- Substite $h_n(X)$ in preferred set of transitions and transition relation.
- Iterate.

Future work

- Efficient algorithm for component generation.
- Efficient algorithm for vector composition.
- Implemention using Boolean Decision Diagrams [2], Reduced Boolean Circuits [1].

Any questions

- Suggestions?
- Comments?

Thank you

References

[1] P. A. Abdulla, P. Bjesse, and N. Een. Symbolic reachability analysis based on sat-solver. In *TACAS*, 2000.

[2] H. R. Andersen and H. Hulgaard. Boolean expression diagrams. In *LICS: IEEE Symposium on Logic in Computer Science*, 1997.

[3] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In A. Halaas and P.B. Denyer, editors, *International Conference on Very Large Scale Integration*, pages 49–58, Edinburgh, Scotland, 1991. North-Holland.

[4] S. Chakraborty and A. Trivedi. Symbolic reachability analysis using additive decomposition. Submitted to TACAS 2004.

[5] P. Chauhan, E. M. Clarke, S. Jha, J. Kukula, T. Shiple, H. Veith, and D. Wang. Non-linear quantification scheduling for efficient image computation. In *ICCAD*, pages 293–298, 2001.

[6] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[7] Filkorn T. Functional extension of symbolic model checking. In *Proc. Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pages 225–232, 1991.

[8] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Proc. Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, Chicago, U.S.A., July 2000. Springer-Verlag.