Verifying Reactive Systems Using Predicate Diagrams*

Dominique Cansell[1], Dominique Méry[2], and Stephan Merz[3]

[1] Université de Metz & LORIA, cansell@loria.fr
[2] Université Henri Poincaré, Nancy & LORIA, mery@loria.fr
[3] Institut für Informatik, Universität München, merz@informatik.uni-muenchen.de

1 Motivation

Verification techniques for reactive systems are traditionally classified as either *deductive* or *algorithmic*. Deductive techniques can handle complex systems, but their implementation based on interactive proof assistants requires careful guidance by expert users. Algorithmic techniques such as model checking have found wide acceptance because they promise full automation, but they are usually restricted to finite-state models. Abstraction techniques [2, 3] promise to integrate the two paradigms. In this paper we propose a particular approach to abstraction centered around the notion of *predicate diagrams*, finite-state transition systems whose nodes are labelled with state predicates and represent the set of system states that satisfy these predicates. Diagrams serve as intermediaries between specifications and properties: on the one hand, non-temporal proof obligations ensure that the diagram represents the specification; these must in general be discharged using theorem proving. On the other hand, properties can be established from the diagram using model checking, obviating the need for tedious temporal-logic reasoning that is typical of liveness proofs. We also show that techniques from abstract interpretation can be used to generate diagrams that can at least serve as a starting point for verification.

Predicate diagrams can be shown to be as powerful as traditional deductive verification. Pragmatically, the use of predicate diagrams may be preferable because of the potential for substantial automation. They may also be attractive because similar notations are used in semi-formal design methods [1] and can aid documentation. Similar diagrammatic techniques have been proposed by the STeP group [8]; they can also be used as the basis for deductive model checking [11]. Our definition is somewhat different in the treatment of fairness conditions. It is also slightly more general in that several actions can be represented within a single diagram, which is useful for the proof of refinement relations, a topic that we do not consider in this paper.

2 Predicate Diagrams

We express system specifications and properties in a variant of linear-time temporal logic whose formulas are built from *state predicates* and *actions*, which may contain primed state variables. For example, $x > 3$ is a state predicate, and $x \leq y' + 1$ is an action. For an action A, we denote by $\textsc{enabled}\ A$ the state predicate obtained from A by existential quantification over the primed state variables. For a state predicate P, we denote by P' the action obtained from P by replacing all flexible variables v by v'. Temporal formulas are then built using boolean connectives, the *always* operator $\Box$, and quantification over rigid (state-independent) variables.

The semantics of state formulas is defined with respect to a *state*, i.e. an assignment of values to state variables, and a valuation of the rigid variables. Actions are interpreted relative to a pair (s, t) of states, where s and t interpret respectively the unprimed and primed state variables. Temporal formulas are interpreted over *behaviors*, which are ω-sequences $\sigma = s_0 s_1 \ldots$ of states [6, 9]. Derived

* This work has been partly supported by a PROCOPE grant from A.P.A.P.E. and DAAD.

connectives include the *eventually* operator defined by $\Diamond F \equiv \neg\Box\neg F$ and, for an action A, the formulas

$$\mathrm{WF}(A) \equiv \Diamond\Box\,\text{ENABLED } A \Rightarrow \Box\Diamond A \qquad \mathrm{SF}(A) \equiv \Box\Diamond\,\text{ENABLED } A \Rightarrow \Box\Diamond A$$

that assert weak and strong fairness conditions for A. This logic is similar to Lamport's Temporal Logic of Actions [7] except that it is not invariant under stuttering. We consider system specifications written in the form $Init \wedge \Box Next \wedge L$ where $Init$ is a state predicate characterizing the system's initial state, $Next$ is an action representing the next-state relation, and L is a conjunction of formulas $\mathrm{WF}(A)$ or $\mathrm{SF}(A)$.

We assume the underlying assertion language to contain a finite set $\mathcal{O}$ of binary relation symbols $\prec$ that are interpreted by well-founded orderings. For $\prec \in \mathcal{O}$, we denote by $\preceq$ its reflexive closure. We write $\mathcal{O}^=$ to denote the set of relation symbols $\prec$ and $\preceq$, for $\prec$ in $\mathcal{O}$.

The definition of predicate diagrams is relative to finite sets $\mathcal{P}$ and $\mathcal{A}$ that contain the state predicates and actions of interest. We denote by $\overline{\mathcal{P}}$ the set containing the predicates in $\mathcal{P}$ and their negations.

Definition 1. *A* predicate diagram $G = (N, I, \delta, o, \zeta)$ *over* $\mathcal{P}$ *and* $\mathcal{A}$ *consists of*

- *a finite set* $N \subseteq 2^{\overline{\mathcal{P}}}$ *of* nodes*,*
- *a finite set* $I \subseteq N$ *of* initial nodes*,*
- *a family* $\delta = (\delta_A)_{A\in\mathcal{A}}$ *of relations* $\delta_A \subseteq N \times N$*,*
- *a family* $o = (o_A)_{A\in\mathcal{A}}$ *of edge labellings* o_A *that associate finite sets* $\{(t_1, \prec_1), \ldots, (t_k, \prec_k)\}$ *of terms* t_i *paired with a relation* $\prec_i \in \mathcal{O}^=$ *with the edges* $(n, m) \in \delta_A$*,*
- *a mapping* $\zeta : \mathcal{A} \to \{\mathrm{NF}, \mathrm{WF}, \mathrm{SF}\}$ *that associates a fairness condition with every action in* $\mathcal{A}$ *(*NF *indicates "no fairness").*

We say that the action $A \in \mathcal{A}$ can be taken *at node* $n \in N$ *iff* $(n, m) \in \delta_A$ *holds for some* $m \in N$.

Definition 2. *Let* $G = (N, I, \delta, o, \zeta)$ *be a predicate diagram over* $\mathcal{P}$ *and* $\mathcal{A}$*. The set* $tr(G)$ *of* traces through G *consists of all* ω*-sequences* $\sigma = s_0 s_1 \ldots$ *of states such that there exist sequences* $n_0 n_1 \ldots$ *and* $A_0 A_1 \ldots$ *of nodes* $n_i \in N$ *and actions* $A_i \in \mathcal{A}$ *such that all of the following conditions hold:*

- $n_0 \in I$ *is an initial node,*
- $(n_i, n_{i+1}) \in \delta_{A_i}$ *holds for all* $i \in \mathbb{N}$*,*
- $s_i \models n_i$ *holds for all* $i \in \mathbb{N}$*,*
- $(s_i, s_{i+1}) \models A_i$ *holds for all* $i \in \mathbb{N}$*,*
- $(s_i, s_{i+1}) \models t' \prec t$ *holds for all* $i \in \mathbb{N}$ *and* $(t, \prec) \in o_{A_i}(n_i, n_{i+1})$*,*
- *for every action* $A \in \mathcal{A}$ *such that* $\zeta(A) = \mathrm{WF}$ *there are infinitely many* $i \in \mathbb{N}$ *such that either* $A_i = A$ *or* A *cannot be taken at* n_i*, and*
- *for every action* $A \in \mathcal{A}$ *such that* $\zeta(A) = \mathrm{SF}$*, either* $A_i = A$ *holds for infinitely many* $i \in \mathbb{N}$ *or there are only finitely many* $i \in \mathbb{N}$ *such that* A *can be taken at* n_i*.*

Predicate diagrams are finite labelled transition systems whose nodes are sets of state predicates that hold at the system states represented by the node. (We indifferently write n for the set and the conjunction of its elements.) Traces through a diagram are behaviors that correspond to fair runs, where enabling conditions of actions $A \in \mathcal{A}$ are identified with the existence of A-labelled edges. We define the notion of *conformance* between diagrams and temporal formulas by trace inclusion. The following theorem gives a set of non-temporal proof conditions that ensure that a diagram conforms to a system specification.

Theorem 3. *Let* $G = (N, I, \delta, o, \zeta)$ *be a predicate diagram over* $\mathcal{P}$ *and* $\mathcal{A}$*, and* $Spec \equiv Init \wedge \Box Next \wedge L$ *be a system specification. If all of the following conditions hold, then* $\sigma \in tr(G)$ *holds for all models* σ *of Spec.*

$$
\begin{aligned}
\mathit{Init} &\equiv n \in \mathit{Nat} \wedge n \neq 0 \wedge c_0 = \text{“t”} \wedge c_1 = \text{“t”} \\
\mathit{Eat}_0 &\equiv c_0 = \text{“t”} \wedge \mathrm{even}(n) \\
&\quad \wedge c_0' = \text{“e”} \wedge c_1' = c_1 \wedge n' = n \\
\mathit{Thk}_0 &\equiv c_0 = \text{“e”} \wedge c_0' = \text{“t”} \\
&\quad \wedge n' = n \text{ div } 2 \wedge c_1' = c_1 \\
\mathit{Eat}_1 &\equiv c_1 = \text{“t”} \wedge \neg\mathrm{even}(n) \\
&\quad \wedge c_1' = \text{“e”} \wedge c_0' = c_0 \wedge n' = n \\
\mathit{Thk}_1 &\equiv c_1 = \text{“e”} \wedge c_1' = \text{“t”} \\
&\quad \wedge n' = 3 * n + 1 \wedge c_0' = c_0 \\
\mathit{Next} &\equiv \mathit{Eat}_0 \vee \mathit{Thk}_0 \vee \mathit{Eat}_1 \vee \mathit{Thk}_1 \\
\mathit{DM} &\equiv \mathit{Init} \wedge \Box\mathit{Next} \wedge \mathrm{WF}(\mathit{Next})
\end{aligned}
$$

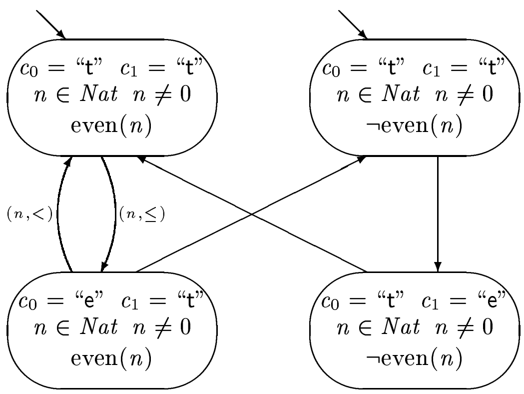

Fig. 1. The “dining mathematicians” example.

1. $\models \mathit{Init} \Rightarrow \bigvee_{n \in I} n$

2. $\models n \wedge \mathit{Next} \Rightarrow \bigvee_{\{(A,m):(n,m)\in\delta_A\}} A \wedge m'$ *holds for every node* $n \in N$.

3. $\models n \wedge A \wedge m' \Rightarrow t' \prec t$ *holds for all* $(n, m) \in \delta_A$ *and all* $(t, \prec) \in o_A(n, m)$.

4. For every action $A \in \mathcal{A}$ *such that* $\zeta(A) \neq \mathrm{NF}$*:*

(a) If $\zeta(A) = \mathrm{WF}$ *then* $\models \mathit{Spec} \Rightarrow \mathrm{WF}(A)$.

(b) If $\zeta(A) = \mathrm{SF}$ *then* $\models \mathit{Spec} \Rightarrow \mathrm{SF}(A)$.

(c) $\models n \Rightarrow \text{ENABLED } A$ *holds whenever* A *can be taken at node* n.

(d) $\models n \wedge A \Rightarrow \neg m'$ *holds for all* $n, m \in N$ *such that* $(n, m) \notin \delta_A$.

If the conditions of theorem 3 holds, we say that G *represents* the specification Spec. As an example, Fig. 1 contains the specification of the “dining mathematicians” example, a mutual-exclusion protocol taken from [3], and a predicate diagram (w.r.t. the predicates shown and the single action Next) that represents that specification. The variables c_0 and c_1 represent the control states of two processes that alternate between “thinking” and “eating”. The integer variable n is shared between the processes to ensure mutual exclusion.

As finite transition systems, predicate diagrams can easily be encoded into the input languages of standard model checkers such as Spin [5], recording the active node and the last action taken. For every predicate $p \in \mathcal{P}$ we define the atomic propositions b_p and $b_{\neg p}$ such that they are true in precisely those nodes that contain p, respectively $\neg p$. We assume the fairness conditions indicated by the diagram. For every term t and relation $\prec \in \mathcal{O}$ such that $(t, \prec)$ appears in some ordering annotation o_A, we add an assumption that ensures that whenever edges labelled by $(t, \prec)$ are taken infinitely often, then edges that are labelled by neither $(t, \prec)$ nor $(t, \preceq)$ are also taken infinitely often. This condition can be expressed by a Streett-type formula.

The encoding of a predicate diagram in a model checker can be used to verify temporal formulas built from predicates in $\mathcal{P}$, substituting b_p and $b_{\neg p}$ for occurrences of p and $\neg p$. For example, we can verify the following properties for the “dining mathematicians” specification from the diagram of Fig. 1:

$$
\begin{array}{llll}
(\mathit{Pos}) & \Box(n \in \mathit{Nat} \wedge n \neq 0) & (\mathit{Excl}) & \Box\neg(c_0 = \text{“e”} \wedge c_1 = \text{“e”}) \\
(\mathit{Live}_0) & \Box\Diamond(c_0 = \text{“e”}) & (\mathit{Live}_1) & \Box\Diamond(c_1 = \text{“e”})
\end{array}
$$

The first two properties are invariants that assert that n remains a positive natural number throughout any run of specification DM, and that mutual exclusion is ensured. The remaining properties are

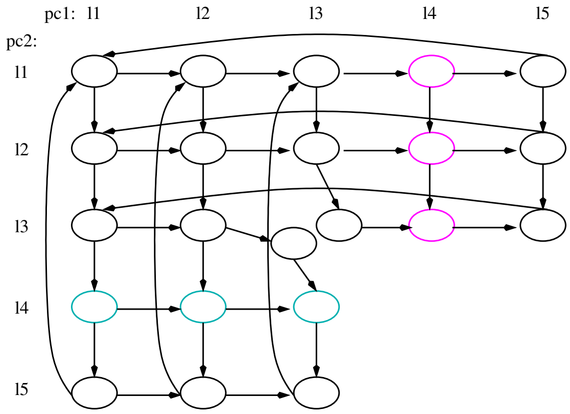

Fig. 2. Predicate diagram for the Bakery algorithm.

liveness properties that assert starvation-freedom for both processes. Note that property $(Live_1)$ can be established despite the cycle between the two leftmost states in the diagram, because the ordering annotations forbid that cycle to be followed indefinitely.

3 Generation of Diagrams by Abstract Evaluation

Instead of verifying the conditions of theorem 3 with the help of a proof assistant, we can try to generate a diagram that represents a given specification by abstract evaluation of the initial condition and the next-state relation based on user-defined (conditional) rewrite rules. Rules appropriate for the dining-mathematicians example include

$$\begin{array}{ll} \mathrm{even}(0) & \neg\mathrm{even}(1) \\ \mathrm{even}(x) \equiv \mathrm{even}(y) \Rightarrow \mathit{even}(x+y) & \mathrm{even}(x) \not\equiv \mathrm{even}(y) \Rightarrow \neg\mathit{even}(x+y) \\ x \in \mathit{Nat} \wedge \mathrm{even}(x) \wedge x \neq 0 \Rightarrow x \text{ div } 2 \neq 0 & x \in \mathit{Nat} \Rightarrow \mathrm{even}(x) \vee \neg\mathrm{even}(x) \end{array}$$

These rules are first applied to the initial condition $Init$. The resulting formula is transformed to disjunctive normal form; each disjunct gives one possible initial node. State expansion is done similarly by evaluating the action $Next$ based on the information about which predicates are true, resp. false, in the source node, and collecting information about the target nodes. The same technique can be used to evaluate other action formulas, in particular those with non-trivial fairness conditions.

It may occasionally happen during state expansion that there is insufficient information about the source node n (represented by unprimed variables) to evaluate some predicate P that appears as a guard in the action currently considered. We can then abort the construction, asking the user to supply more rewrite rules. Alternatively, we may treat P as if it were true; this respects condition (2) of theorem 3 because it can only add disjuncts on the right-hand side. On the other hand, it may invalidate condition (4c), so we are not allowed to use this approach for actions that have non-trivial fairness conditions. In any case, we mark such edges as "maybe" edges to alert the user that they could be obvious culprits when model checking fails. As a third alternative, we have found it useful to reconsider the predecessors, say $n_1, \ldots, n_k$, of n in the part of the diagram that has been constructed so far, and to check whether $n_i \wedge Next \wedge n'$ implies P' or its negation, using an automatic theorem prover. In that case, we add P or $\neg P$ to the set of predicates contained in n, splitting the node if necessary.

If the procedure does not fail, the generated diagram is guaranteed to represent the given specification. In particular, the existence of an edge (other than edges marked as "maybe") leaving a given node n implies the enabledness of the action at n. We have prototypically implemented the procedure outlined above, using the rewrite engine from Atelier B [10], the automatic prover Simplify [4], and the model checker Spin. Fig. 2 shows a predicate diagram generated for the two-process version of Lamport's "Bakery" algorithm, a standard benchmark problem for the verification of infinite-state systems. We have omitted the predicate labels, but have indicated the control locations ("l4" is the critical section). One process moves horizontally, the other one vertically. The diagram contains two nodes where both processes are at location "l3"; these are distinguished by which process holds the lower "ticket" and may therefore proceed. In producing this diagram, we have only included rules to distinguish between the "ticket" variables t_1 and t_2 being zero or not. The remaining predicates have been introduced "on the fly" via backtracking as explained above. Both mutual exclusion and liveness have been verified by Spin from the diagram.

4 Discussion

We consider predicate diagrams as an interface between specifications and properties. In this paper, we have emphasized the "bottom-up" construction of diagrams starting from the specification. Dually, deductive model checking [11] relies on a "top-down" approach where an initial diagram that ensures the desired property is interactively refined until the conditions of theorem 3 can be established. The bottom-up approach could be characterized as "abstract model checking" because it generates the full state space of the model modulo the abstraction defined by the predicates in $\mathcal{P}$.

The results reported in this paper, though encouraging, are still preliminary. It remains to consider more systematically the annotation of generated diagrams with fairness and ordering information. Another interesting option to explore is the verification of refinement relations between specifications. This is possible because we do not formally distinguish between specifications and properties, and because we allow transitions that correspond to different actions to be represented within the same diagram. We also intend to study appropriate generalizations for parametric systems.

References

1. G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modelling Language: User Guide*. Addison Wesley, 1999.
2. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.
3. Dennis Dams, Orna Grumberg, and Rob Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*.
4. D. Detlefs, G. Nelson, and J. Saxe. Simplify: the ESC theorem prover. Technical report, Systems Research Center, Digital Equipment Corporation, Palo Alto, CA, nov 1996.
5. Gerard Holzmann. The Spin model checker. *IEEE Trans. on Software Engineering*, 23(5):279–295, may 1997.
6. Fred Kröger. *Temporal Logic of Programs*, volume 8 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1987.
7. Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
8. Z. Manna, A. Browne, H.B. Sipma, and T.E. Uribe. Visual abstractions for temporal verification. In A. Haeberer, editor, *AMAST'98*, volume 1548 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, 1998.
9. Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems—Specification*. Springer-Verlag, New York, 1992.
10. STERIA Méditerrannée. *Atelier B, Manuel de Référence du Langage B*. GEC Alsthom Transport and STERIA and SNCF and INFRETS and RATP, 1997.
11. H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In *8th International Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 208–219, New Brunswick, N.J., 1996. Springer-Verlag. to appear in *Formal Methods in System Design*, 1999.