Exploring Infinite State Spaces with Finite Automata

Pierre Wolper

Université de Liège

Verification as State Space Exploration

- Consider programs that can be given semantics in terms of *state-transition systems*, i.e. structures

$$K = (S, R, I),$$

  where

  - S is a finite or infinite set of states,
  - $R \subseteq S \times S$ is a transition relation,
  - $I \subseteq S$ is a set of initial states.

- A program P is an implicit *finite* description of a structure $K_P = (S, R, I)$.
- Verifying a program amounts to checking properties of K_P, most commonly of checking properties of its set of reachable states

$$S_{\mathit{reach}} = \mu X.I \cup R(X).$$

Computing and Representing the Reachable States

To compute the reachable states S_{reach}, the obvious approach is to repeatedly apply $\rho \equiv I \cup R(X)$ to the empty set until stabilization. For doing this, one needs a representation for subsets of S.

If S is finite, this can be done

- *By explicit enumeration*, in which case applying R is simply done by doing a program computation step;
- *Symbolically*, in which case elements of S are coded by fixed length bit vectors, and subsets of S as well as the relation R by Boolean formulas; to ease the required computation, it is common to represent the Boolean formulas in a normal form (BDDs).

If S is infinite, the only choice is a symbolic representation. To be usable, such a representation has to be sufficiently

- *Expressive*, for coding I, R and S_{reach}; as well as sufficiently
- *Decidable*, for convergence and properties of S_{reach} to be checkable.

Usual choices are formulas in a restricted logical theory, often written in a normal form in order to ease the computation.

Note. Having a suitable representation formalism does not guarantee that the fixpoint computation terminates, though this can be the case for restricted classes of programs.

Theme of this talk: *finite automata are an interesting and versatile symbolic representation formalism*

Data-Oriented Infinite State Spaces: A Simple Framework

Let us consider systems for which the state space is infinite due to the nature of the data that is manipulated. Precisely, consider programs defined by a tuple $(C, c_0, M, m_0, \mathit{Op}, \Delta)$, where

- C is a finite set of *control locations*,
- M is a (possibly infinite) *memory domain* (often given as the cross product of the domains of a finite number of variables),
- $\mathit{Op} \subseteq M \to M$ is a set of *memory operations*,
- $\Delta \subseteq C \times \mathit{Op} \times C$ is a finite set of *transitions*,
- c_0 is an *initial control location*, and m_0 is an *initial memory content*.

A state is thus an element of $C \times M$

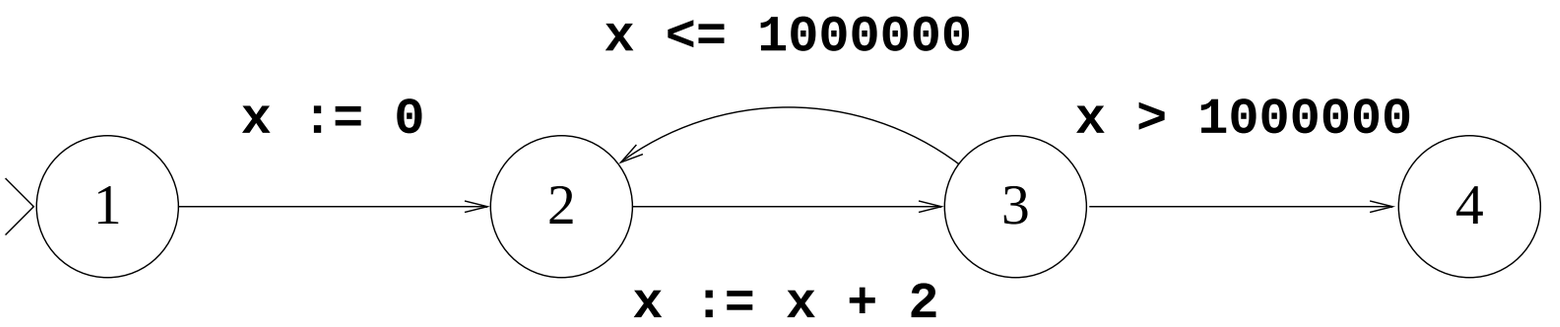
Generating Infinite Sets of States: The Need to Accelerate

- In most cases, applying the relation ρ to a finite set of states will also yield a finite set.
- Thus if the set of reachable states is infinite and the set of initial states is finite, repeatedly applying ρ to the set of initial states will never converge to the set of reachable states.
- To solve this problem, one needs to *accelerate* the exploration of the set of reachable states.
- Two common acceleration techniques are
  - *widening*, which amounts to guessing an upper approximation of the set of reachable states.
  - *using meta-transitions*, which corresponds to precomputing the effect of applying a cyclic transition an unbounded number of times.

Generating Infinite Sets of States: Using Meta-transitions

- Identify some loops in the finite-state control of the system.
- Explore the state space as usual, but when reaching a loop, attempt to compute the effect of indefinitely iterating the sequence of operations labeling the loop.
- When this computation succeeds, introduce a corresponding *meta-transition* and use it as a computation step in the state-space exploration.
- The state-space exploration terminates when nothing can be added to the computed state space.

An example of the use of meta-transitions



(①, $\perp$)
(②, 0)
(②, $2k$) with $0 \leq k \in \mathbf{N} \leq 500000$
(③, $2k + 2$) with $0 \leq k \in \mathbf{N} \leq 500000$
(④, 1000002)

Note that a meta-transition allows one to go arbitrarily deep into a computation in one step.

The limits of meta-transitions

Using meta-transitions does not guarantee that the state space can always be computed. Indeed,

- the search might not terminate in spite of the meta-transitions, or
- the meta-transitions corresponding to some cycles might not be computable and representable.

Programs with Integer Variables: Linear Integer Systems

In a *Linear Integer System*, the memory is a set of unbounded integer variables. Formally, we have the following.

- The memory domain M is $\mathbf{Z}^n$, where $n > 0$ represents the *number of variables*.
- The set of memory operations *Op* contains all functions $M \to M$ of the form

  $$P\vec{x} \leq \vec{q} \to \vec{x} := T\vec{x} + \vec{b}$$

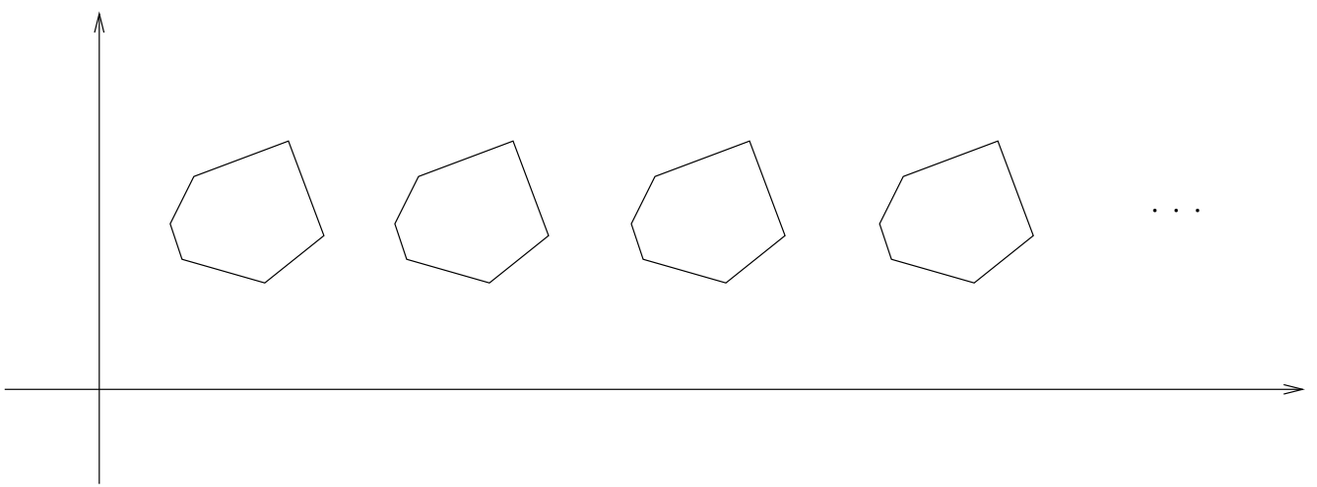
  where $P \in \mathbf{Z}^{m \times n}, \vec{q} \in \mathbf{Z}^m, m \in \mathbf{N}, T \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$.

  The system $P\vec{x} \leq \vec{q}$ is the *guard* of the operation and the transformation $\vec{x} := T\vec{x} + \vec{b}$ is the *assignment* of the operation.

Representing sets of integer values

First idea : linear constrained sets.

- Yes, but iterating a simple operation like $x := x + 3$ yields sets which are periodic unions of linear constrained sets



- One needs means to represent periodicity !

Representing sets of integers II

- Use a logical formalism, e.g. Presburger Arithmetic (first-order arithmetic without multiplication).

$$\exists k\, x_0 (x = x_0 + 5k \wedge 1 \leq x_0 \leq 3 \\ \wedge 2 \leq y \leq 4)$$

  - Expressiveness is sufficient,
  - The problem is computing with such a logical representation.

- Alternative : use automata to represent sets of integers.

Encoding Integers by Strings

Principles :

- Binary representation,
- Unbounded numbers,
- Most significant bit first.
- 2's complement for negative numbers (at least p bits for a number x such that $-2^{p-1} \leq x < 2^{p-1}$).

Examples :

$$\begin{aligned} 4 &: 0100, 00100, 000100, \ldots \\ -4 &: 100, 1100, 11100, \ldots \end{aligned}$$

Vectors are represented by using same length encodings of the components and reading them bit by bit.

Expressiveness of the Automaton Representation

- To simplify operations, we use automata that accept all valid encodings of a given subset of $\mathbf{Z}^n$.
- The subsets of $\mathbf{Z}^n$ representable by automata are those definable in a slight extension of Presburger arithmetic: one adds a function giving the largest power of 2 dividing its argument.
- If one requires representability by automata in all bases ≥ 2, then the representable subsets are exactly those definable in Presburger arithmetic.
- Reduced deterministic automata provide a normal form for all Presburger definable arithmetic constraints.

Building Automata for Linear Equations

Consider an equation $\vec{a}.\vec{x} = b$ with $\vec{a} \in \mathbf{Z}^n$ and $b \in \mathbf{Z}$.

The problem is to build an automaton $A = (S, 2^n, \delta, s_0, F)$ accepting the encodings of all $\vec{x} \in \mathbf{Z}^n$ satisfying the equation.

- Each state s of the automaton (except the initial state s_0) is uniquely labeled by an integer $\beta(s)$. The final state is the one labeled by b. The initial state is a special state labeled by 0.
- The idea of the construction is that the label of a state represents the value of $\vec{a}.\vec{x}$ for the bits that have been read so far.

- Therefore, for states s and s' other than s_0 to be linked by a transition labeled $\vec{d}$, the number $\beta(s)$ associated with the state s' has to be given by

$$\beta(s') = \vec{a}.\vec{x}' = 2.\vec{a}.\vec{x} + \vec{a}.\vec{d} = 2.\beta(s) + \vec{a}.\vec{d}.$$

  where $\vec{(}x)$ and $\vec{(}x')$ respectively represent the vectors read when respectively reaching s and s'.

  Note that the state s' is unique.

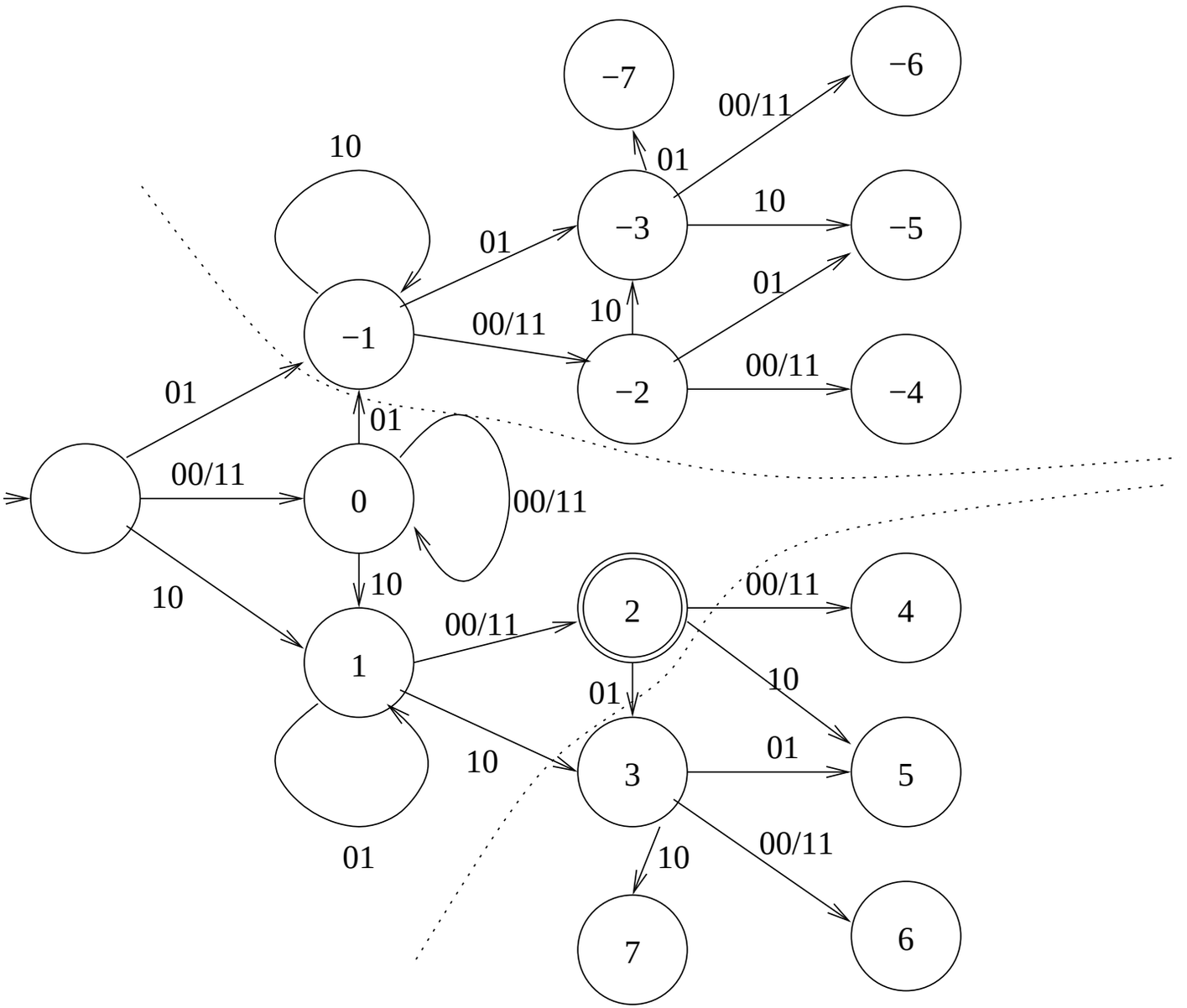
- For the initial state, the associated value is 0, but one has to take into consideration that the first bit is a sign bit: in the function given the next state, a 1 bit is interpreted as -1.

Termination of the Construction and Inequations

- If $\vec{a} = (a_1, \ldots, a_n)$, then the accepting state cannot be reached from any state s labeled by an integer $\beta(s) \geq |b/2| + \Sigma_i |a_i|$. Thus, only a finite number of states are needed in the automaton.
- In practice, it is more effective to do the construction starting with the final state and proceeding backwards.
- For an inequation $\vec{a}.\vec{x} < b$, one proceeds similarly except that all states with labels $< b$ are accepting.
- The automaton obtained is deterministic and hence can easily be minimized.

Example:

$x - y = 2$


−7
−6
00/11
10
01
−3
10
−5
01
01
10
−1
00/11
01
−2
00/11
−4
01
00/11
0
00/11
10
10
2
00/11
4
00/11
1
10
01
10
3
01
5
01
00/11
10
7
6

Handling Arbitrary Formulas

- For Boolean combinations of linear constraints, one uses the corresponding operations on automata.

- For existential quantification, one uses projection.

- Universal quantification is handled by transforming $\exists$ to $\neg\forall\neg$.

- One important advantage of this approach is that one has a normal form even for formulas that represent non convex sets and include periodicity constraints.

Iterating operations on integers

A simple case: an instruction $I \equiv T\vec{x} \leq \vec{u} \rightarrow \vec{x} := A\vec{x} + \vec{b}$, with A idempotent ($A^2 = A$) and an initial value $\vec{x}_0$

Compute the values obtained by the repeated execution of I on $\vec{x}_0$:

$$\begin{array}{lcrcl} A\vec{x}_0 & + & & + & \vec{b} \\ A\vec{x}_0 & + & A\vec{b} & + & \vec{b} \\ A\vec{x}_0 & + & 2A\vec{b} & + & \vec{b} \\ & \vdots & & \vdots & \\ A\vec{x}_0 & + & kA\vec{b} & + & \vec{b} \\ & \vdots & & \vdots & \end{array}$$

Cycle precondition : $T(A\vec{x}_0 + kA\vec{b} + \vec{b}) \leq \vec{u}$

More general results have been developed.

Programs with Integers and Reals

- The automaton-based representation for integers can be extended to reals by using automata on infinite words.

- Real numbers are encoded by their infinite binary expansion (note that some numbers have two encodings).
  Examples :

$$\begin{aligned} L(3.5) &= 0^+11\star 1(0)^\omega \cup 0^+11\star 0(1)^\omega \\ L(-4) &= 1^+00\star(0)^\omega \cup 1^+011\star(1)^\omega; \end{aligned}$$

- Implementing operations on infinite word automata is problematic (especially complementation), but using a topological argument, it has been show that all sets definable in linear arithmetic over the integers and reals has a representation that is accepted by a weak deterministic infinite word automaton.

- This allows the use of a simple algorithm for determinization and provides a canonical representation.

- Automata thus are a useful tool for handling the combined theory of the reals and integers, with applications such as analysing various classes of timed and hybrid systems.

Another Application of Automata Representations: Systems with Unbounded FIFO Queues

In a *queue system*, the memory domain is a set of unbounded queues. Formally, we have the following.

- The memory domain is of the form $\Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, where $n > 0$ represents the *number of queues*, and each Σ_i is the finite *queue alphabet* of the i-th queue q_i (we assume they are distinct).
- The set of memory operations *Op* contains the two *queue operations* $q_i!a$ and $q_i?a$ for each queue q_i and symbol $a \in \Sigma_i$.

Representing the Content of Queues: The QDD

A *Queue Decision Diagram (QDD)* is a finite automaton representation of a set of queue contents.

- A content $(w_1, \ldots, w_n)$ for a queue system with n queues is represented by the concatenation $w_1 \cdot w_2 \cdots w_n$ of the individual queue contents taken in a fixed order.
- A QDD is a finite automaton over the union of the queue $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ of the queue alphabets such that all words accepted by the automaton satisfy

  $$w = w|_{\Sigma_1} w|_{\Sigma_2} \ldots w|_{\Sigma_n}.$$

  That is, every word accepted by the automaton can be interpreted as a content for the set of queues of the system.

Operations on QDDs

A state of a queue system is a pair $(c, m) \in C \times M$. We consider sets of states with an identical control location c represented as a pair (c, A) where A is a QDD. We have to address the following problems.

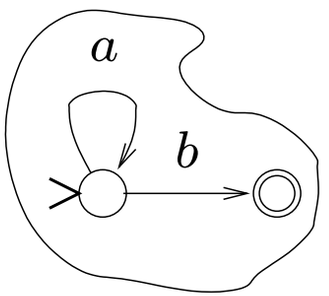
- Compute the effect of applying a transition (c, op, c') to the states represented by (c, A), i.e. the set of states $\{(c', m') | \exists m(m \in L(A) \wedge m' \in op(m)\}$.
- Compute the effect of applying a sequence of transitions to (c, A).
- Compute the effect of repeatedly applying a cyclic sequence of transitions to (c, A).

What we want to compute is (if it exists) the QDD resulting from the application of the operations.

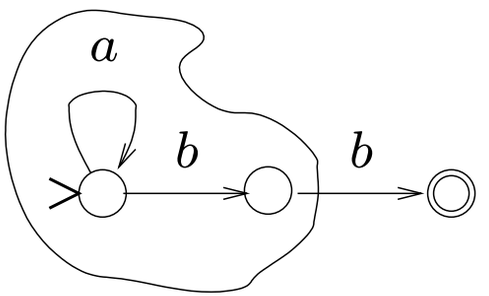
Applying Operations to QDDs
The Single Queue Case

The effect of single operations or of finite sequences of operations is easy to compute as can seen on the following example.

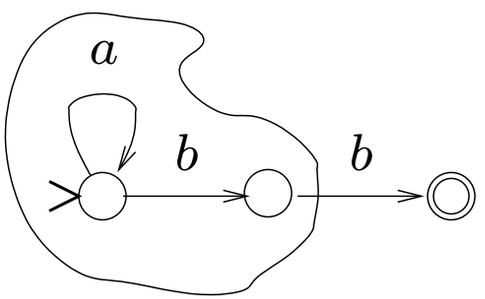
a^*b:



$(q!b)[a^*b]$:



$(q!b; q?a)[a^*b]$:



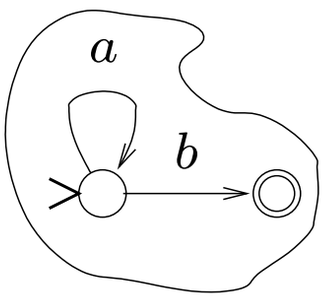
Iterating Sequences of Operations on QDDS

To compute the effect of iterating a sequence of operation σ to the set of queue contents represented by a QDD A, i.e. to compute $\sigma^*(A)$, we proceed as follows

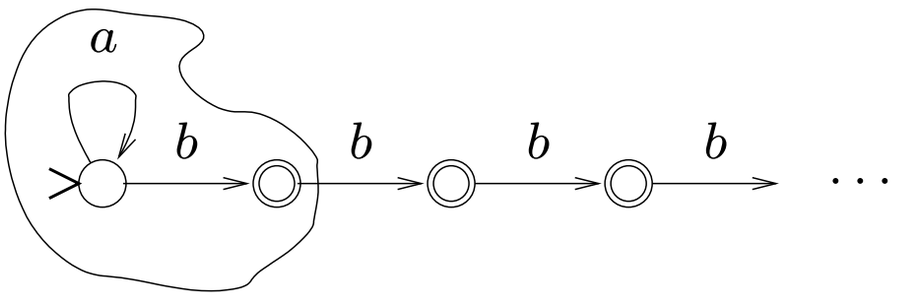
- We use $\sigma^*(A) = \bigcup_k \sigma^k(A)$
- Some periodicity will eventually occur within the $\sigma^k(A)$.
- $\sigma^*(A)$ can thus be represented by a finite union.

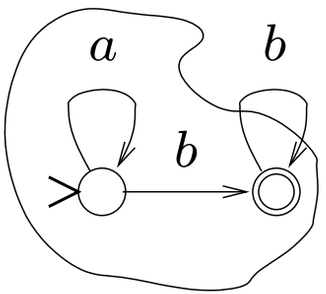
Iterating Sequences on a QDD: An Example

a^*b:



$(q!b; q?a)^*[a^*b]$:





Operations on Systems with Multiple Queues

- For single operations, one simply operates as above on the part of the QDD representing the queue on which the operation is performed.
- Sequences of operations can be similarly handled.
- The result of iterating a sequence of operations cannot always be represented as a QDD. For instance $(q_1!a; q_2!b)^*$.

  The problem comes from the ability to count the number of iterations by looking at the content of two or more queues.
- When only one queue allows to count the number of iterations, one can combine the result of handling separately the different queues.

Other Types of Systems

- **Pushdown systems**: systems with one pushdown stack. In this case the set of reachable states is regular and can always be computed.

- **Parametric Systems**: systems with an arbitrary unbounded number of processes. States are represented by words and the transition relation by a finite-state transducer. Reachable states are computed by generic techniques applied to finite-state transducers.