

# Symbolic Model Checking for LTL using SNF

**Daniel Sheridan**

dan.sheridan@contact.org.uk

**Abstract:** The Separated Normal Form (SNF) for LTL has already been successfully used for bounded model checking [2] as part of the encoding. We show how it can also be used as part of a symbolic model checking procedure for LTL by first building an automata to represent the SNF.

## SNF for LTL

SNF is defined by Fisher [1], based on the Separation Theorem of Gabbay [3]. A series of transformations based on the fixpoint characterisations of LTL operators converts an LTL expression in negation normal form into a conjunction of rules of the form:

$$\mathbf{G} \left( \bigwedge_i (P_i \Rightarrow F_i) \right)$$

Each  $P_i \Rightarrow F_i$  can be ( $l_i$  and  $l_j$  are literals):

$$\begin{aligned} \text{start} &\Rightarrow \bigvee_j l_j && \text{An initial rule} \\ \bigwedge_i l_i &\Rightarrow \mathbf{X}_1 \bigvee_j l_j && \text{A global } \mathbf{X}_1\text{-rule} \\ \bigwedge_i l_i &\Rightarrow \mathbf{X} \bigvee_j l_j && \text{A global } \mathbf{X}\text{-rule} \\ \bigwedge_i l_i &\Rightarrow \mathbf{F} \bigvee_j l_j && \text{A global } \mathbf{F}\text{-rule} \end{aligned}$$

## Example transformations

$$\begin{aligned} T_G(\{P \Rightarrow \mathbf{G} f\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow f \wedge x \\ x \Rightarrow \mathbf{X}_1(f \wedge x) \end{array} \right\} \cup \Psi \\ T_U(\{P \Rightarrow f \mathbf{U} g\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow g \vee (f \wedge x) \\ x \Rightarrow \mathbf{X}(g \vee (f \wedge x)) \\ P \Rightarrow \mathbf{F} g \end{array} \right\} \cup \Psi \\ T_{ren}(\{P \Rightarrow \mathbf{G} f(\mathbf{F} g)\} \cup \Psi) &= \left\{ \begin{array}{l} P \Rightarrow \mathbf{G} f(x) \\ x \Rightarrow \mathbf{F} g \end{array} \right\} \cup \Psi \end{aligned}$$

## The Fixpoint Form

SNF only transforms least fixpoint operators: *eventually* (**F**) is left unchanged. The advantage of SNF for bounded model checking comes from changing infinite operators to next-time operators in the transformations above. We added an extra transformation in [2] to get the Fixpoint Form:

$$T_F(\{P \Rightarrow \mathbf{F} f\} \cup \Psi) = \left\{ \begin{array}{l} P \Rightarrow f \vee x \\ x \Rightarrow \mathbf{X}(f \vee x) \\ \text{bound} \Rightarrow f \vee \neg x \end{array} \right\} \cup \Psi$$

This means that the *next time* (**X**) operator is the only temporal operator remaining. Each rule now connects two successive states — we are very close to an automaton already.

## Translation to Automata

For a given type of rule, we can write down the corresponding property of the automata.  $S$  is a set of states;  $R : S \rightarrow S$  is the transition relation;  $L : S \rightarrow \mathcal{P}(AP)$  is the labelling function; and  $I$  is the set of initial states.

- Propositional rules  $\bigwedge_i l_i \Rightarrow \bigvee_j l_j$  become invariants:

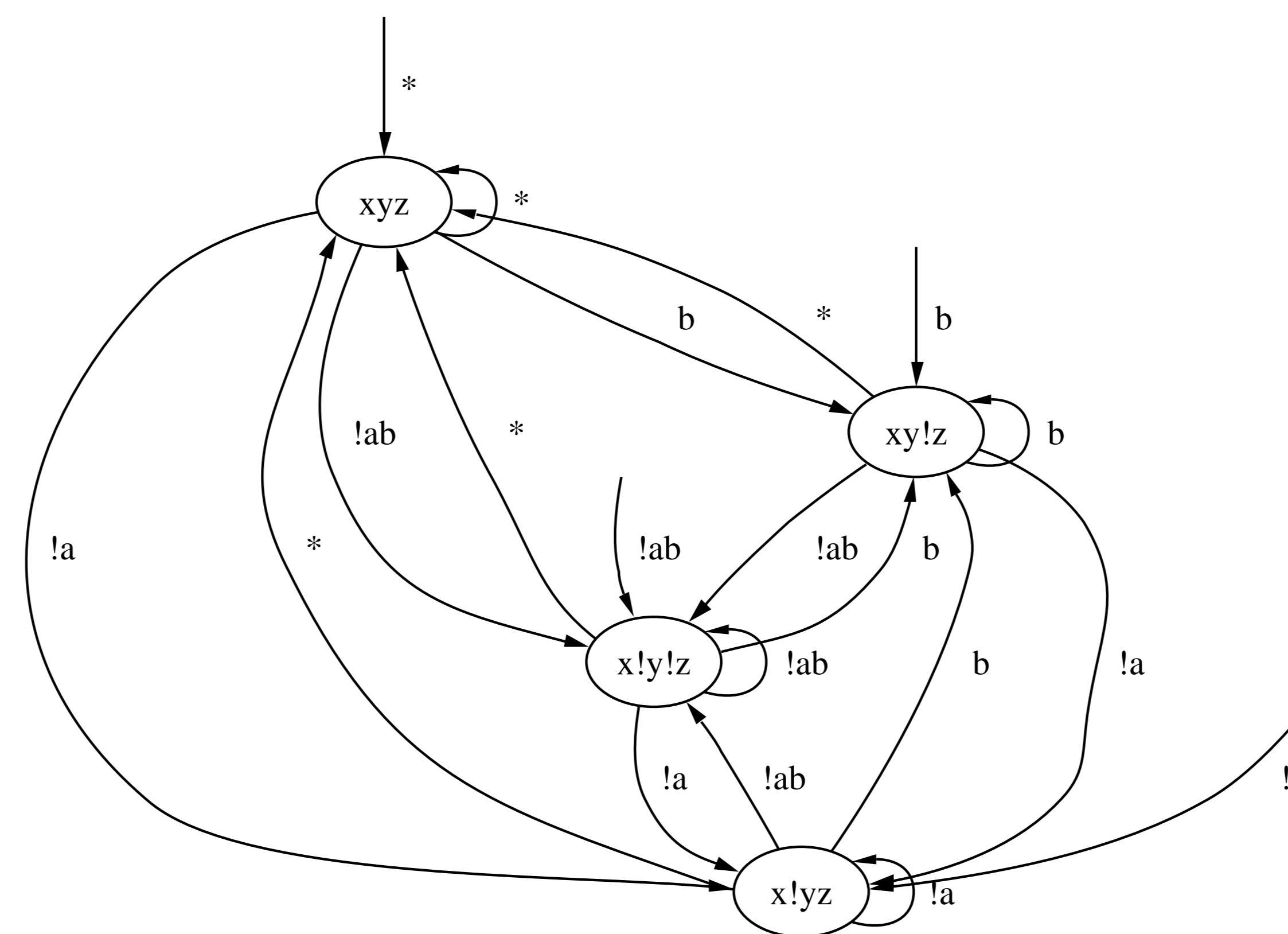
$$\forall s \in S, (\exists l \in l_i \cdot l \notin L(s)) \vee (\exists l \in l_j \cdot l \in L(s))$$

- Initial rules  $\text{start} \Rightarrow \bigvee_j l_j$  become constraints on the set of initial states:

$$I \subseteq \{s | s \in S \wedge \exists l \in l_j \cdot l \in L(s)\}$$

- Global **X** rules  $\bigwedge_i l_i \Rightarrow \mathbf{X} \bigvee_j l_j$  become constraints over transitions:

$$R \subseteq \{(s_0, s_1) | s_0, s_1 \in S \wedge (\exists l \in l_i \cdot l \notin L(s_0)) \vee (\exists l \in l_j \cdot l \in L(s_1))\}$$



**Example 1:** Automata for  $\mathbf{G}(a \rightarrow \mathbf{F}b)$ ; unreachable  $\neg x$  state are omitted. Fairness condition is  $\neg z \vee b$

## The bound Operator

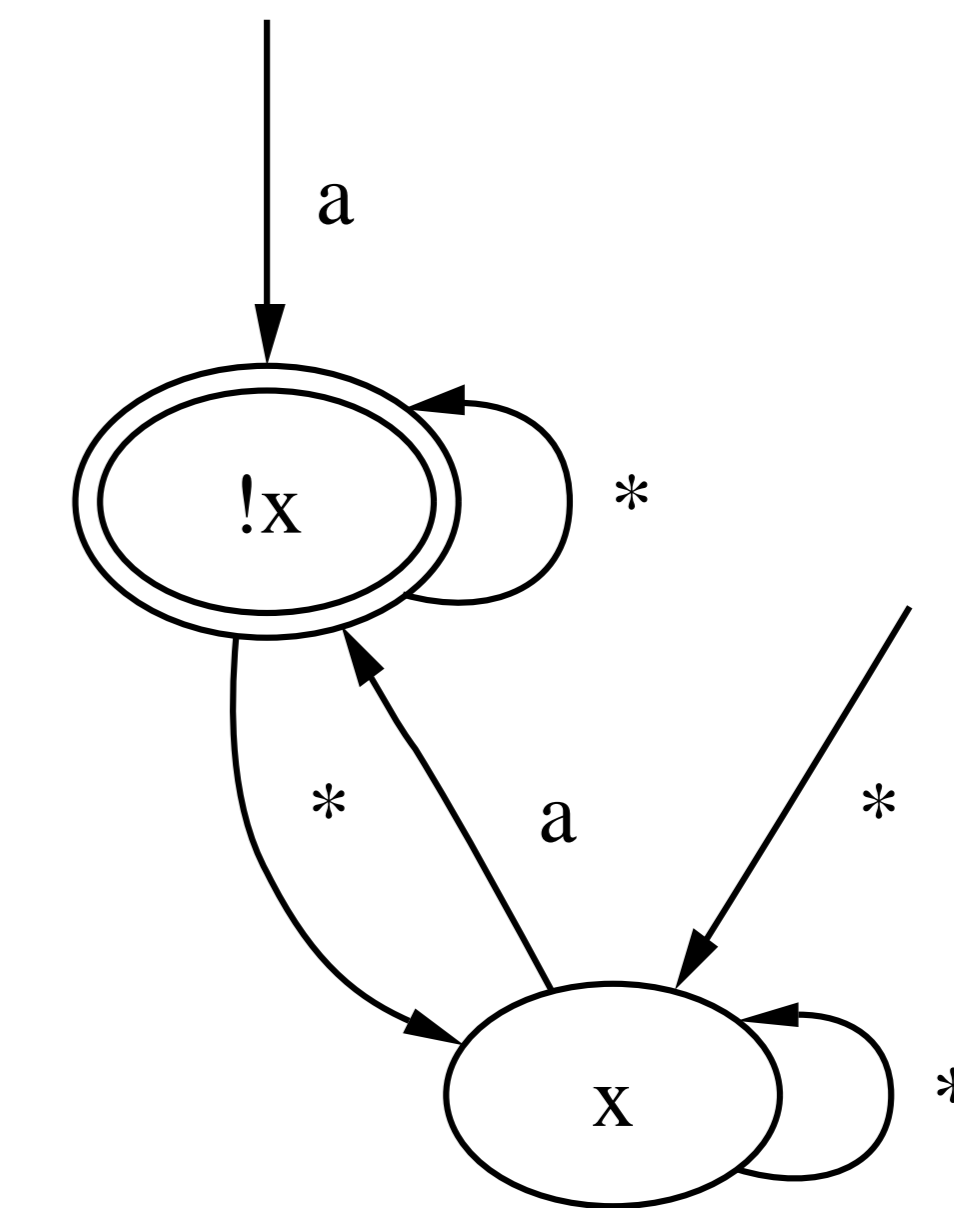
We characterise the *eventually* (**F**) operator by  $\mathbf{F}x \leftrightarrow x \vee \mathbf{X}\mathbf{F}x$ , and hence

$$y \leftrightarrow x \vee \mathbf{X}y \quad \text{and} \quad \mathbf{F}x \leftrightarrow y$$

In a loop in which  $x$  is *false*, the value of  $y$  depends only on the successive value of  $y$ : it could incorrectly take the value *true*. **bound** rules represent the

constraint that eventually formulæ do eventually become *true*. In the bounded model checking case this implicitly means “before the bound”. In the symbolic model checking case we assert that we can’t get stuck in an infinite loop unless the formula is *true*.

A rule of the form **bound**  $\Rightarrow \bigvee_j l_j$  becomes a restriction to fair paths in which  $\bigvee_j l_j$  holds.



**Example 2:** Automata for  $\mathbf{F}a$ . The fairness condition  $a \vee \neg x$  determines the acceptance state

## Examples

Example 1 :  $\mathbf{G}(a \rightarrow \mathbf{F}b)$

$$\begin{aligned} \text{start} &\Rightarrow x \wedge (\neg a \vee y) && y \Rightarrow z \vee b \\ x &\Rightarrow \mathbf{X}x \wedge (\neg a \vee y) && z \Rightarrow \mathbf{X}z \vee b \\ \text{bound} &\Rightarrow b \vee \neg z \end{aligned}$$

Example 2 :  $\mathbf{F}a$

$$\begin{aligned} \text{start} &\Rightarrow a \vee x \\ x &\Rightarrow \mathbf{X}a \vee x \\ \text{bound} &\Rightarrow a \vee \neg x \end{aligned}$$

## References

- [1] Michael Fisher. A resolution method for temporal logic. In *Proceedings of Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, August 1991.
- [2] Alan Frisch, Daniel Sheridan, and Toby Walsh. A fixpoint based encoding for bounded model checking. In M D Aagaard and J W O’Leary, editors, *Formal Methods in Computer-Aided Design; 4th International Conference, FMCAD 2002*, volume 2517 of *Lecture Notes in Computer Science*, pages 238–254, Portland, OR, USA, November 2002. Springer.
- [3] Dov Gabbay. The declarative past and imperative future. In H. Barringer, editor, *Proceedings of the Colloquium on Temporal Logic and Specifications*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer-Verlag, 1989.

