
Symbolic synthesis of state based reactive programs

Diploma Thesis

Nico Wallmeier

18.03.03

Structure

1. Background
2. Infinite games over a finite game graph
3. Transformation to the symbolic state space
4. Applications

1. Background

Motivation

- Crash of the first Ariane-5 rocket (iX, September 1996)
- Computation error of the Intel Pentium processor

⇒ Verification is necessary

- Testing & Simulation
 - Does not supply any correctness guarantee
 - Sometimes only limited applicability

⇒ Computer aided techniques in formal verification

Model Checking

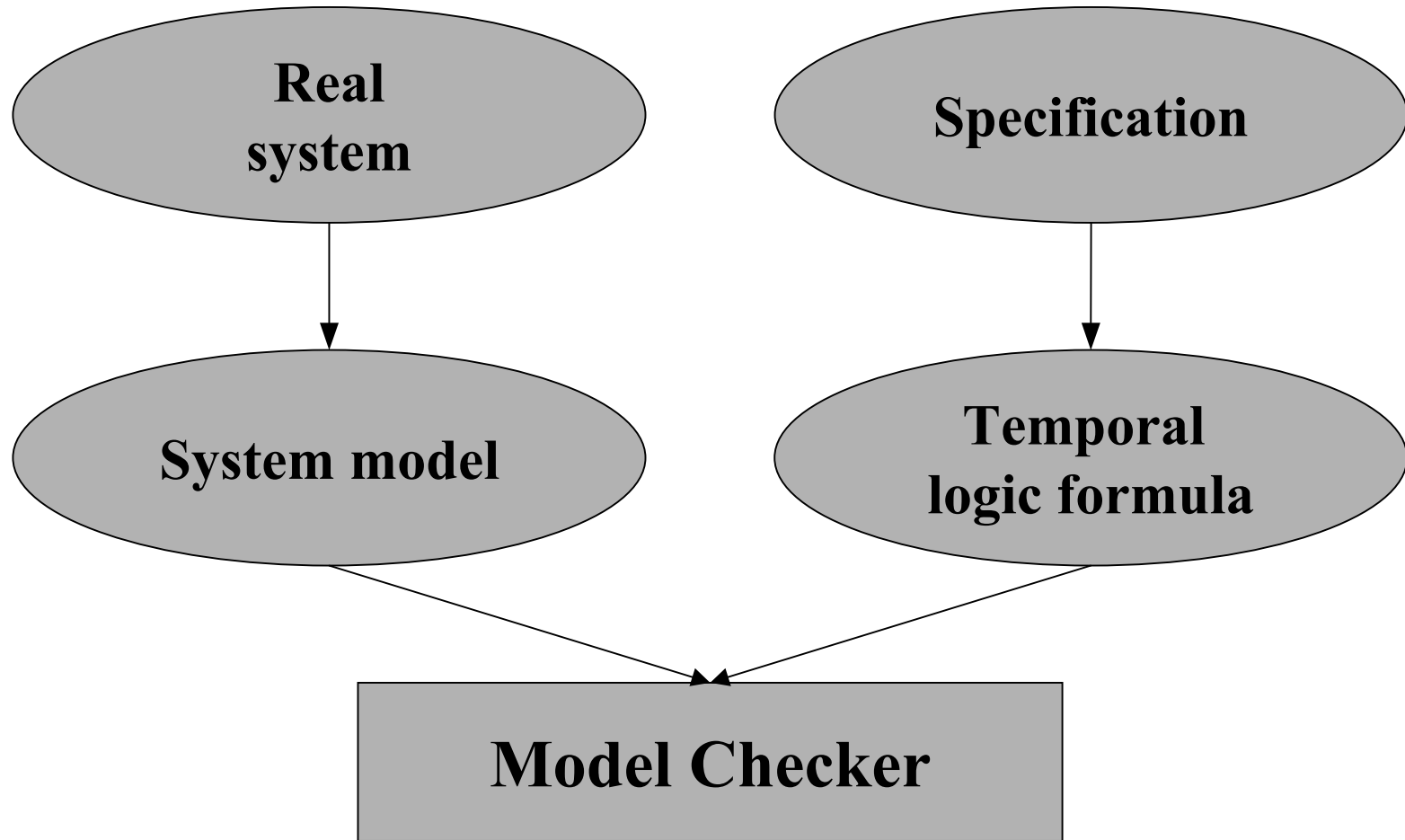
- Clarke, Emerson et al.:

Model checking is an automatic technique for verifying correctness properties of safety-critical reactive systems.

- System is tested against a specification
- If an error occurs an error scenario will be generated

Model Checking - 2

- Procedure



Model Checking - 3

- Success in practical applications with two ideas (symbolic Model Checking):
 - Specification logic CTL (polynomial time) by Clarke and Emerson at the beginning of the 1980s
 - Symbolic method to overcome the “state explosion problem” – presentation of the states is done via BDDs (Binary Decision Diagrams) (Lee, Akers, Moret and Bryant)

Infinite two person games

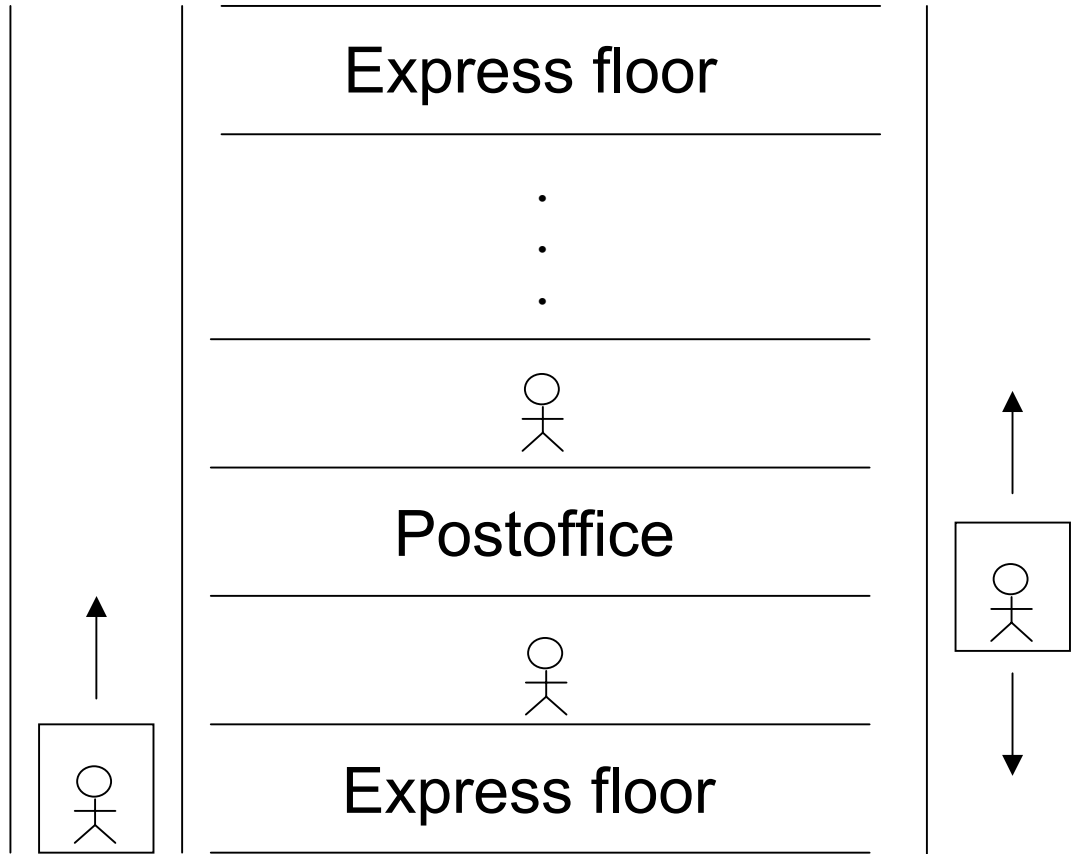
- Better system model: 2 agents
 - Controller (agent 0)
 - Environment (agent 1)
- Specification by
 - Game graph
 - Winning condition for player 0
- Play: Infinite path in the game graph

Infinite two person games - 2

- Classical theory of solving games
 - 1969 Büchi, Landweber
 - 1993 McNaughton
 - Currently: EU-project GAMES
(Aachen, Bordeaux, ..., Warsaw)
- Goal of this work:
 - Transformation to the symbolic method
 - Implementation of these algorithms

Goal

- Goal is to solve such examples:



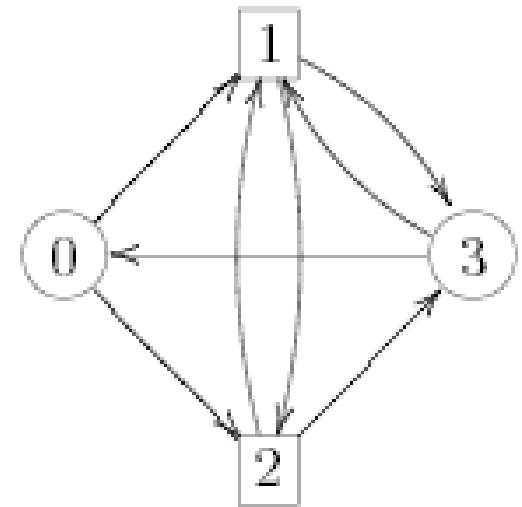
Specification

- Two lifts in a building with e floors should satisfy:
 - All requested floors will be served
 - The highest and the ground floor are served directly
 - No lift drives past a requested floor on his way
 - At most one person gets in a lift at a time
 - At least three floors are not requested
 - In the second floor is the post office. A lift needs one turn of the controller to wait there for exchanging the mail.
 - Both lifts are not at the same time in the second floor.

2. Infinite games over a finite game graph

Game graph

- *Game graph* G is defined by
 - Set of states $Q = Q_0 \dot{\cup} Q_1$
 - Transitions $E \subseteq Q \times Q$
(every state must have a successor)



- *Play* ρ is a infinite sequence of states
 $\rho = \rho(0)\rho(1)\rho(2)\dots$ with $(\rho(i), \rho(i+1)) \in E$
- $\text{Oc}(\rho) = \{ q \mid \exists i \rho(i) = q \}$ – occurrence set
- $\text{In}(\rho) = \{ q \mid \exists^\omega i \rho(i) = q \}$ – infinity set

Overview winning conditions

Name	Requirement	Winning condition
Reachability	$F \subseteq Q$	$\text{Oc}(\rho) \cap F \neq \emptyset$
Safety	$F \subseteq Q$	$\text{Oc}(\rho) \subseteq F$
Weak parity	$c:Q \rightarrow \{0, \dots, k\}$	$\max(\text{Oc}(c(\rho)))$ is even
Staiger-Wagner	$F \subseteq \text{Pot}(Q)$	$\text{Oc}(\rho) \in F$
Request-Response	$P_i, Q_i \subseteq Q \ (1 \leq i \leq r)$	$\bigwedge_{i=1}^r \forall j (\rho(j) \in P_i \Rightarrow \exists j' \geq j \rho(j') \in R_i)$ Temporal: $\bigwedge_{i=1}^r G(P_i \rightarrow F R_i)$
Büchi	$F \subseteq Q$	$\text{In}(\rho) \cap F \neq \emptyset$
Parity	$c:Q \rightarrow \{0, \dots, k\}$	$\max(\text{In}(c(\rho)))$ is even

Method for solving example

1. Capture safety conditions by restricting the game graph
2. Rest of winning conditions is conjunction of request-response conditions:
Reduce to Büchi condition
3. Solve game for Büchi condition

Reachability winning condition

- Simplest winning condition:
reachability of a set F
- player 0 wins the play $\rho \Leftrightarrow$
 ρ reaches a state in the set F sometime
- Solution with “Attractor”: Compute for
 $i=0,1,2,\dots$ the nodes, from which player 0
can reach the set F in $\leq i$ moves

Attractor

- Definition

- $\text{Attr}_0^i(F) = \{ q \in Q \mid \text{player 0 can reach the set } F \text{ from } q \text{ in } \leq i \text{ moves} \}$
- $\text{Attr}_0^0(F) = F$
- $\text{Attr}_0^{i+1}(F) = \text{Attr}_0^i(F) \cup \{ q \in Q_0 \mid \exists (q,r) \in E \text{ with } r \in \text{Attr}_0^i(F) \} \cup \{ q \in Q_1 \mid \forall (q,r) \in E \text{ holds } r \in \text{Attr}_0^i(F) \}$

- Conclusions:

- $\text{Attr}_0^i(F) \subseteq \text{Attr}_0^{i+1}(F)$
- $\text{Attr}_0^m(F) = \text{Attr}_0^{m+1}(F)$ for a $m \leq |Q|$
 $\Rightarrow \text{Attr}_0(F) = \text{Attr}_0^m(F)$ for such a m

Use of attractor computation

- Solvable games by attractor computation
 - Reachability game
 - Safety game
 - Weak parity game
 - Büchi game

3. Transformation to the symbolic state space

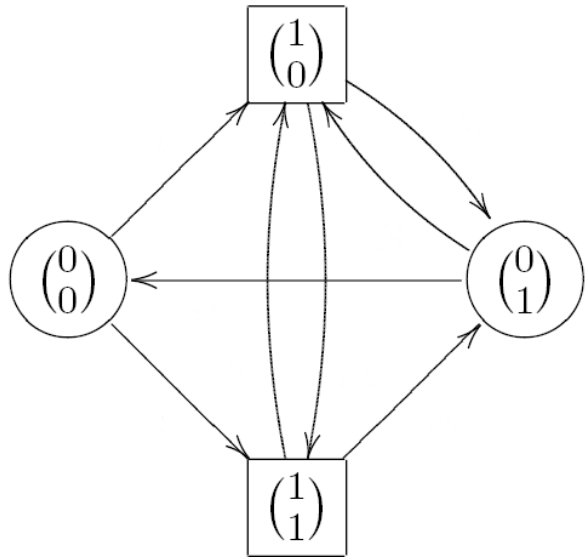
Motivation

- Abstract state space:
 - „State Explosion Problem“
 - Analogous to Model Checking
 - Often no practical application possible
- ⇒ In this work the symbolic method is introduced (as known from Model Checking)

Symbolic state space

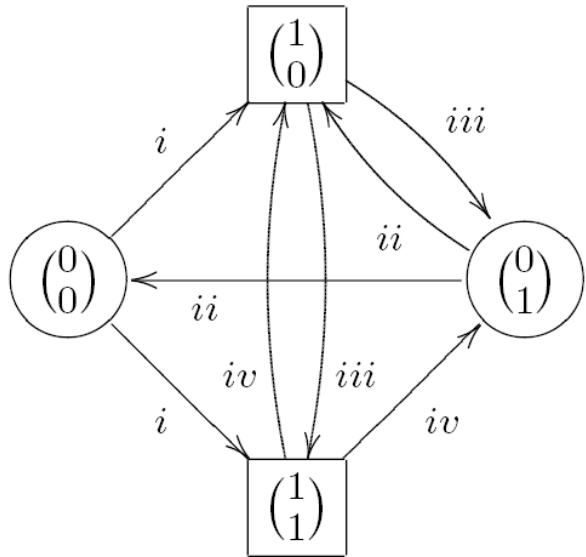
- Set of Boolean variables
- $V = \{v_0, \dots, v_n\}$ as well as $V' = \{v'_0, \dots, v'_n\}$
- Concrete state is an assignment of all variables of V
- 2^n states \rightarrow n variables

Symbolic game graph



- Is defined by formulas for
 - Nodes of player 0
 - Nodes of player 1
 - Transitions
- Nodes of player 0
 - $\varphi_0 = \neg v_0$
- Nodes of player 1
 - $\varphi_1 = v_0$

Symbolic game graph - 2



- Transition formula τ

i. $\neg V_0 \wedge \neg V_1 \wedge V_0'$

ii. $\neg V_0 \wedge V_1 \wedge \neg V_1'$

iii. $V_0 \wedge \neg V_1 \wedge V_1'$

iv. $V_0 \wedge V_1 \wedge (V_0' \Leftrightarrow \neg V_1')$

Attractor

- Definition

- $\text{Attr}_0^0(\lambda) = \lambda$

- $\text{Attr}_0^{i+1}(\lambda) = \text{Attr}_0^i(\lambda) \vee$

- $(\varphi_0 \wedge (\tau \wedge \text{Attr}_0^i(\lambda)|_{V \rightarrow V^c})|_V) \vee$

- $(\varphi_1 \wedge \neg(\tau \wedge \neg \text{Attr}_0^i(\lambda)|_{V \rightarrow V^c})|_V)$

- Strategy

- $\text{Strat}_0^0(\lambda) = \text{false}$

- $\text{Strat}_0^{i+1}(\lambda) = \text{Strat}_0^i(\lambda) \vee$

- $(\text{Attr}_0^{i+1}(\lambda) \wedge \neg \text{Attr}_0^i(\lambda) \wedge \tau \wedge$

- $(\varphi_1 \vee (\varphi_0 \wedge \text{Attr}_0^i(\lambda)|_{V \rightarrow V^c}))$

Achieved results

Game	Solution
Reachability	Attractor computation
Safety	Attractor computation
Weak parity	Attractor computation
Staiger-Wagner	Reduction to weak parity
Request-Response	Reduction to Büchi
Büchi	Attractor+ and Recur
Parity	McNaughton-algorithm

4. Applications

Input language

- $x[2], x'[2]$
- Boolean Operations such as Or, And, XOr, XAnd, Not, ...
- Existential and universal quantifier for variable indices – e.g. $\exists i \{i < 3\} x[i]$
- Arithmetic for variable indices, e.g. $x[i+3]$
- External parameters

Case study

- Request-Response game with $3 \cdot (e-2)$ RR-pairs
- $5 \cdot e + 3$ variables for e floors
 - e variables: Position first lift
 - e variables: Position second lift
 - e variables: Requests on the floors
 - e variables: Requests in the first lift
 - e variables: Requests in the second lift
 - One variable to determine the player
 - Two variables for the post office

Case study - 2

Floors	Size game graph	BDD creation	Solve game	Size Büchi game	Size winning regions	
					player 0	player 1
3	25	40.69 s	30.38 s	1,200	24	1
4	673	53.77 s	73.09 s	516,864	672	1
5	12,913	172.29 s	191.10 m	119,006,208	0	12,913

Case study - 3

Winning strategy of the environment for five floors

Force one lift to second floor, let it wait one move with no other requests and look at second lift:

E	Pos. 2. Lift	Chosen Requests
	Ground floor	1. floor + 4. floor
P	1. floor	Ground floor + 4. floor
	3. floor	Ground floor + 4. floor
E	4. floor	Ground floor + 1. floor

Further work

- Develop suitable restrictions for
 - Game graph specification
 - Winning conditions
- Hierarchical approach (SDL specification)
- Support for time conditions

Screenshots

SymProg

Infos | Eingabe | Parser | Zwischenergebnisse | Ergebnis

Spiel

Spieltyp: Paritätsspiel
kombiniert mit: kein 2. Spiel

Optionen
 Parser-Seite anzeigen
(sollte nur bei kleinen Beispielen aktiviert werden)

Spielgraph | Paritätsspiel

Externe Parameter
Anzahl: 0

#	Var	Wert

Anzahl der Variablen: 2

Knoten Spieler 0:
 $x[0] = x[1]$

Knoten Spieler 1: Rest
 $x[0] \neq x[1]$

Transitionen:
 $((x[0] = x'[0]) \& (x[1] = x'[1]) \& !(x[0] \& x[1])) \mid$
 $(x[0] \& !x[1] \& (x'[0] = x'[1])) \mid$
 $(!x[0] \& x[1] \& (((x[0] = !x'[0]) \& (x[1] = x'[1])) \mid ((x[1] = !x'[1]) \& (x[0] = x'[0])))$

Farbe: 0 1 2

```
graph TD
    N00((0,0))
    N01[0,1]
    N10[1,0]
    N11((1,1))
    N00 --> N00
    N00 --> N01
    N00 --> N10
    N01 --> N00
    N01 --> N11
    N01 --> N01
    N10 --> N00
    N10 --> N11
    N11 --> N00
    N11 --> N11
```


Screenshots - 3

The screenshot shows the SymProg application window with the following content:

SymProg (Title Bar)

Infos | Eingabe | **Parser** | Zwischenergebnisse | Ergebnis

Spielgraph:

Knoten Spieler0: $x[0] = x[1]$

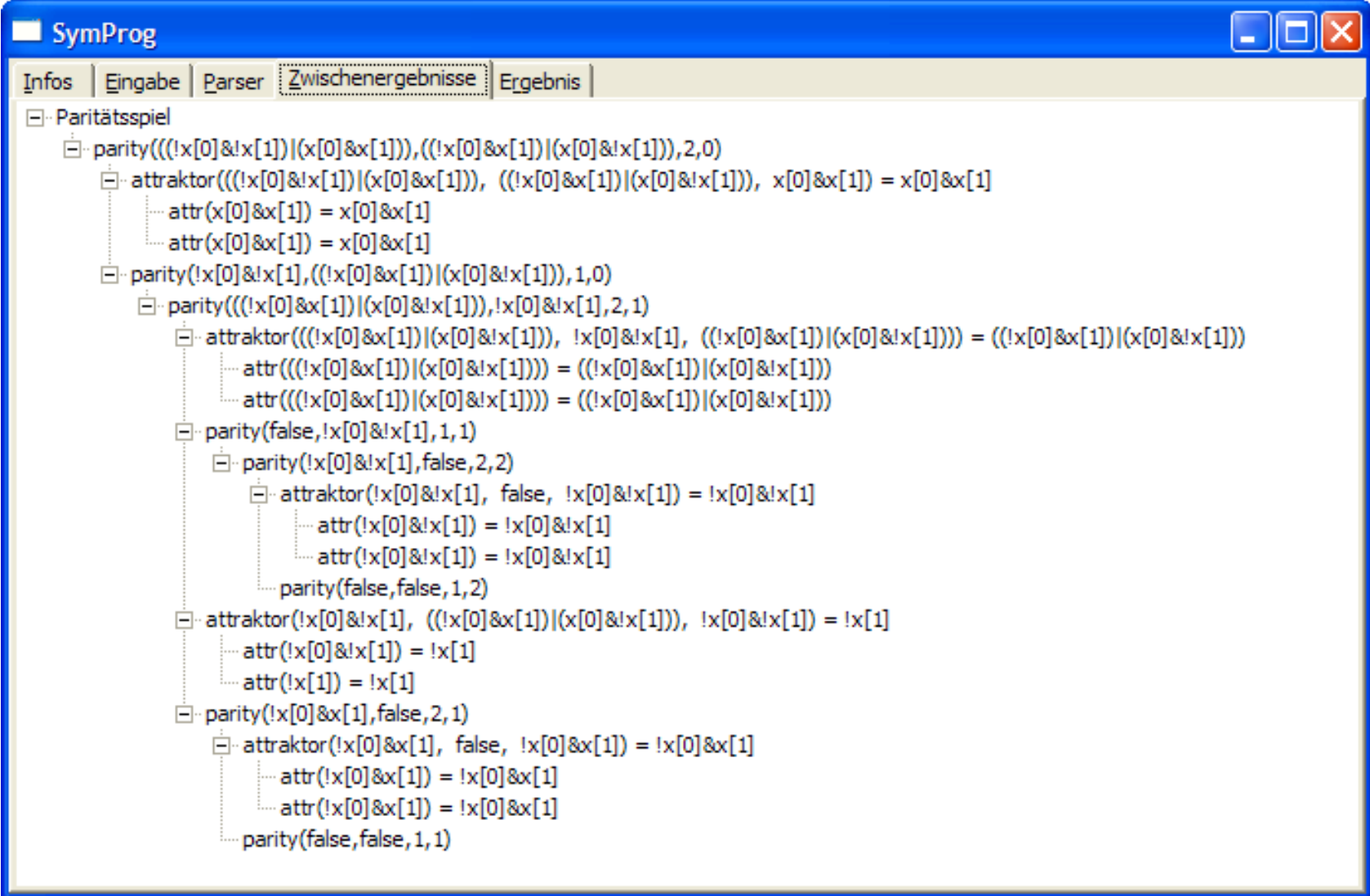
Knoten Spieler1: $x[0] = !x[1]$

Transitionen: $((x[0] = x'[0]) \& (x[1] = x'[1]) \& !(x[0] \& !x[1])) \mid (x[0] \& !x[1] \& (x'[0] = x'[1])) \mid (!x[0] \& x'[1] \& (((x[0] = !x'[0]) \& (x[1] = x'[1])) \mid ((x[1] = !x'[1]) \& (x[0] = x'[0]))))$

Paritätsspiel

#	Färbung
0	$!x[0] \& !x[1]$
1	$x[0] = !x[1]$
2	$x[0] \& x[1]$

Screenshots - 4



The screenshot shows the SymProg application window with the 'Zwischenergebnisse' (Intermediate Results) tab selected. The main content area displays a tree view of a parity game solution. The root node is 'Paritätsspiel', which branches into several nodes representing different game states and attractor computations. The nodes are as follows:

- Paritätsspiel
 - parity(((!x[0]&!x[1])|(x[0]&x[1])),((!x[0]&x[1])|(x[0]&!x[1])),2,0)
 - attraktor(((!x[0]&x[1])|(x[0]&x[1])), ((!x[0]&x[1])|(x[0]&!x[1])), x[0]&x[1] = x[0]&x[1])
 - attr(x[0]&x[1]) = x[0]&x[1]
 - attr(x[0]&x[1]) = x[0]&x[1]
 - parity(!x[0]&!x[1],((!x[0]&x[1])|(x[0]&!x[1])),1,0)
 - parity(((!x[0]&x[1])|(x[0]&!x[1])),!x[0]&!x[1],2,1)
 - attraktor(((!x[0]&x[1])|(x[0]&!x[1])), !x[0]&!x[1], ((!x[0]&x[1])|(x[0]&!x[1]))) = ((!x[0]&x[1])|(x[0]&!x[1]))
 - attr(((!x[0]&x[1])|(x[0]&!x[1]))) = ((!x[0]&x[1])|(x[0]&!x[1]))
 - attr(((!x[0]&x[1])|(x[0]&!x[1]))) = ((!x[0]&x[1])|(x[0]&!x[1]))
 - parity(false,!x[0]&!x[1],1,1)
 - parity(!x[0]&!x[1],false,2,2)
 - attraktor(!x[0]&!x[1], false, !x[0]&!x[1]) = !x[0]&!x[1]
 - attr(!x[0]&!x[1]) = !x[0]&!x[1]
 - attr(!x[0]&!x[1]) = !x[0]&!x[1]
 - parity(false,false,1,2)
 - attraktor(!x[0]&!x[1], ((!x[0]&x[1])|(x[0]&!x[1])), !x[0]&!x[1]) = !x[1]
 - attr(!x[0]&!x[1]) = !x[1]
 - attr(!x[1]) = !x[1]
 - parity(!x[0]&x[1],false,2,1)
 - attraktor(!x[0]&x[1], false, !x[0]&x[1]) = !x[0]&x[1]
 - attr(!x[0]&x[1]) = !x[0]&x[1]
 - attr(!x[0]&x[1]) = !x[0]&x[1]
 - parity(false,false,1,1)

Screenshots - 5

The screenshot shows the SymProg application window with the following content:

- Window title: SymProg
- Tabbed interface with tabs: Infos, Eingabe, Parser, Zwischenergebnisse, Ergebnis (selected).
- Section: Gewinnbereiche:
 - Gewinnbereich Spieler0: $((!x[0] \& !x[1]) | x[0])$
 - Mächtigkeit: 3
 - Gewinnbereich Spieler1: $|x[0] \& x[1]$
 - Mächtigkeit: 1
 - Benötigte Zeit (Spiel): 0,020 Sekunden
 - Benötigte Zeit (Parser): 0,951 Sekunden
- Section: Gewinnstrategie:
 - Knoten: $|x[0] \& x[1]$
 - Button: Berechnen
 - Mögliche Nachfolgerknoten gemäß Gewinnstrategie:
 - $|x[0] \& x[1]$