Verifying Featured Transition Systems using Variability Parity Games

Sjef van Loo

May 14, 2019

1 Introduction

Model verification techniques can be used to improve the quality of software. These techniques require the behaviour of the software to be modelled, we then want to check if the model behaves conforming to some requirement. Different languages are proposed and well studied to express these requirements, examples are LTL, CTL, CTL* and μ-calculus. Once the behaviour is modelled and the requirement is expressed in some language we can use modal checking techniques to determine if the model satisfies the requirement.

These techniques are well suited to model and verify the behaviour of a single software product. However software systems can be designed to have certain parts enabled or disabled. This gives rise to many software products that all behave very similar but not identical, such a collection is often called a *product family*. The differences between the products in a product family is called the *variability* of the family. A family can be verified by using the above mentioned techniques to verify every single product independently. However this approach does not use the similarities in behaviour of these different products, an approach that would make use of the similarities could potentially be a lot more efficient.

Labelled transition systems (LTSs) are often used to model the behaviour of a system, while it can model behaviour well it can't model variability. Efforts to also model variability include I/O automata, modal transition systems and *featured transition systems* (FTSs). Specifically the latter is well suited to model all the different behaviours of the software products as well as the variability of the entire system in a single model.

Efforts have been made to verify requirements for entire FTSs, as well as to be able to reason about features. Notable contributions are fLTL, fCTL and fNuSMV. However, as far as we know, there is no technique to verify an FTS against a μ-calculus formula. The modal μ-calculus is very expressive and subsumes other temporal logics like LTL, CTL and CTL*. In this thesis we will introduce a technique to do this. We first look at LTSs, the modal μ-calculus and FTSs. Next we will look at an existing technique to verify an LTS, namely solving *parity games*, as well as show how this technique can be used to verify an FTS by verifying every software product it describes independently. An extension to this technique is than proposed, namely solving *variability parity games*. We will formally define variability parity games and prove that solving a them can be used to verify FTSs.

2 Verifying transition systems

We first look at labelled transition systems (LTSs) and the modal μ-calculus and what it means to verify an LTS. The definition below are derived from [1].

Definition 2.1. *A labelled transition system (LTS) is a tuple $M = (S, Act, trans, s_0)$, where:*

- *S is a set of states,*
- *Act a set of actions,*
- *$trans \subseteq S \times Act \times S$ is the transition relation with $(s, a, s') \in trans$ denoted by $s \xrightarrow{a} s'$,*
- *$s_0 \in S$ is the initial state.*

Consider the example in figure 1 (directly taken from [2]) of a coffee machine where we have two actions: ins (insert coin) and std (get standard sized coffee).

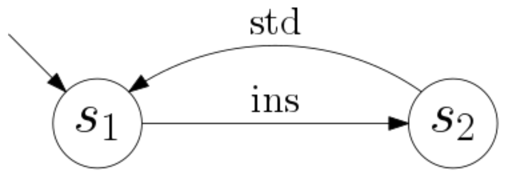


Figure 1: Coffee machine LTS C

Definition 2.2. *A modal μ-calculus formula over the set of actions Act and a set of variables $\mathcal{X}$ is defined by*

$$\varphi = \top \mid \bot \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

with $a \in Act$ and $X \in \mathcal{X}$.

No negations in the language because negations can be pushed inside to the propositions, ie. the $\top$ and $\bot$ elements.

The modal μ-calculus contains boolean constants $\top$ and $\bot$, propositional operators $\vee$ and $\wedge$, modal operators $\langle\,\rangle$ and $[\,]$ and fixpoint operators μ and ν. A formula is closed when variables only occur in the scope of a fixpoint operator for that variable.

A modal μ-calculus formula can be interpreted with an LTS, this results in a set of states for which the formula holds.

Definition 2.3. *For LTS $(S, Act, trans, s_0)$ we inductively define the interpretation of a modal μ-calculus formula φ, notation $[\![\varphi]\!]^\eta$, where $\eta : \mathcal{X} \to \mathcal{P}(S)$ is a logical variable valuation, as a set of* *states where φ is valid, by:*

$$
\begin{aligned}
[\![\top]\!]^\eta &= S\\
[\![\bot]\!]^\eta &= \emptyset\\
[\![\varphi_1 \wedge \varphi_2]\!]^\eta &= [\![\varphi_1]\!]^\eta \cap [\![\varphi_2]\!]^\eta\\
[\![\varphi_1 \vee \varphi_2]\!]^\eta &= [\![\varphi_1]\!]^\eta \cup [\![\varphi_2]\!]^\eta\\
[\![\langle a\rangle\varphi]\!]^\eta &= \{s \in S | \exists_{s' \in S} s \xrightarrow{a} s' \wedge s' \in [\![\varphi]\!]^\eta\}\\
[\![[a]\varphi]\!]^\eta &= \{s \in S | \forall_{s' \in S} s \xrightarrow{a} s' \implies s' \in [\![\varphi]\!]^\eta\}\\
[\![\mu X.\varphi]\!]^\eta &= \bigcap_{f \subseteq S} \{f | f = [\![\varphi]\!]^{\eta[X:=f]}\}\\
[\![\nu X.\varphi]\!]^\eta &= \bigcup_{f \subseteq S} \{f | f = [\![\varphi]\!]^{\eta[X:=f]}\}\\
[\![X]\!]^\eta &= \eta(X)
\end{aligned}
$$

Given closed formula φ, LTS $M = (S, Act, trans, s_0)$ and $s \in S$ we write $(M, s) \models \varphi$ iff $s \in [\![\varphi]\!]^\eta$ for M, we say that formula φ holds for M in state s. If formula φ holds for M in the initial state we say that formula φ holds for M and write $M \models \varphi$.

Again consider the coffee machine example (figure 1) and formula $\varphi = \nu X.\mu Y.([ins]Y \wedge [std]X)$ (taken from [2]) which states that action std must occur infinitely often over all runs. Obviously this holds for the coffee machine, therefore we have $C \models \varphi$.

2.1 Featured transition systems

A featured transition system extends the LTS definition to express variability. It does so by introducing *features* and *products* into the definition. Features are options that can be enabled or disabled for the system. A product is a feature assignments, ie. a set of features that is enabled for that product. Not a products are valid, some features might be mutually exclusive and some features might always be required. To express products one can use feature diagrams as explained in [3]. Feature diagrams offer a nice way of expressing which feature assignments are valid, but since they offer just that we simply represent the valid products with a set of feature assignments. Finally FTSs guards every transition with a boolean expression over the set of features. We have the following definition, based on [3]:

Definition 2.4. *A featured transition system (FTS) is a tuple $M = (S, Act, trans, s_0, N, P, \gamma)$, where:*

- *$S, Act, trans, s_0$ are defined as in an LTS,*
- *N is a non-empty set of features,*
- *$P \subseteq \mathcal{P}(N)$ is a set of products, ie. feature assignments, that are valid,*
- *$\gamma : trans \to \mathbb{B}(N)$ is a total function, labelling each transition with a boolean expression over the features. A product $p \in \mathcal{P}(N)$ satisfying the boolean expression of transition t is denoted by $p \models \gamma(t)$, $\gamma(t)(p) = 1$ or $p \in [\![\gamma(t)]\!]$. The boolean expression that is satisfied by any feature assignment is denoted by $\top$, ie $p \models \top$ for any p.*

A transition $s \xrightarrow{a} s'$ and $\gamma((s,a,s')) = f$ is denoted by $s \xrightarrow{a|f} s'$.

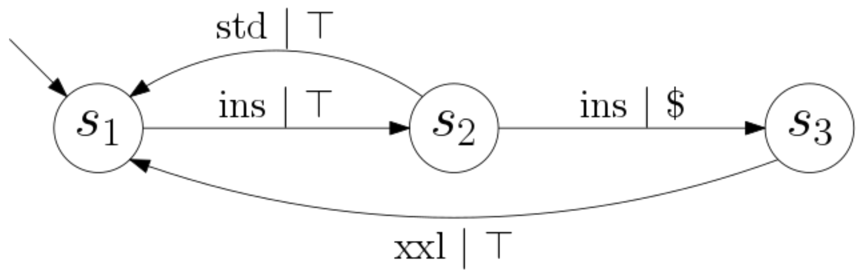


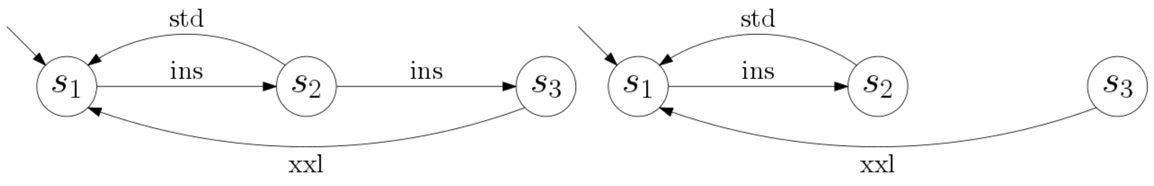
Figure 2: Coffee machine FTS C

Consider the example in figure 2 (directly taken from [2]) which shows an FTS for a coffee machine For this example we have two features $N = \{\$, €\}$ and two valid products $P = \{\{\$\}, \{€\}\}$.

An FTS expresses they behaviour of multiple products, we can derive the behaviour of a single product by simply removing all the transitions from the FTS for which the product doesn't satisfy the feature expression guarding the transition. We call this a *projection* ([3]).

Definition 2.5. *The projection of an FTS $M = (S, Act, trans, s_0, N, P, \gamma)$ to a product $p \in P$, noted $M_{|p}$, is the LTS $M' = (S, Act, trans', s_0)$, where $trans' = \{t \in trans \mid p \models \gamma(t)\}$.*

The coffee machine example can be projected to its two products, which results in the LTSs in figure 3.



(a) C projected to the dollar product: $C_{|\{\$\}}$ (b) C projected to the euro product: $C_{|\{€\}}$

Figure 3: Projections of the coffee machine FTS

We want to verify the FTS against a modal μ-calculus formula φ. That is, we want to find out for which products in the FTS its projection satisfies φ. Formally, given FTS $M = (S, Act, trans, s_0, N, P, \gamma)$ and modal μ-calculus formula φ we want to find $P_s \subseteq P$ such that:

- for every $p \in P_s$ we have $M_{|p} \models \varphi$ and
- for every $p \in P \backslash P_s$ we have $M_{|p} \not\models \varphi$.

Furthermore a counterexample for every $p \in P \backslash P_s$ is preferred.

3 Verification using parity games

LTSs can be verified using parity games.

$$\begin{tikzcd}
\text{LTS } M \arrow[dr] & & \text{Modal } \mu\text{-calculus formula } \varphi \arrow[dl] \\
& \text{PG} \arrow[d] & \\
& M \overset{?}{\models} \varphi &
\end{tikzcd}$$

Figure 4: LTS verification using PG

4 Featured parity games

5 Variability parity games

$$\begin{tikzcd}
\text{FTS } M \arrow[dr] & & \text{Modal } \mu\text{-calculus formula } \varphi \arrow[dl] \\
& \text{VPG} \arrow[d] & \\
& M_{|p} \overset{?}{\models} \varphi & \\
& \text{for every valid product } p \text{ in M} &
\end{tikzcd}$$

Figure 5: FTS verification using VPG

6 Definitions

6.1 Transition systems

Similar to [3].

Definition 6.1. *The projection of an FTS M to a product $p \in P$, noted $M_{|p}$, is the LTS $M' = (S, Act, trans', s_0)$, where $trans' = \{t \in trans \mid p \models \gamma(t)\}$.*

A fixed point formula $\sigma X.\varphi$, with $\sigma \in \{\mu, \nu\}$, can be unfolded which results in the formula φ where very X is replaced by $\sigma X.\varphi$, ie. $\varphi[X := \sigma X.\varphi]$. A fixed point formula is equivalent to its unfolding, ie. $\sigma X.\varphi$ is equivalent to $\varphi[X := \sigma X.\varphi]$. [4]

Definition 6.2. *Given LTS $M = (S, Act, trans, s_0)$, state $s \in S$ and mu-calculus formula φ we write $M, s \models \varphi$ if and only if φ is satisfied in state s for LTS M. If $M, s_0 \models \varphi$ we write $M \models \varphi$.*

7 Goal

Similar to [5].

Given an FTS $M = (S, Act, trans, s_0, N, P, \gamma)$ and a modal μ-calculus formula φ we want to find the set $P_s \subseteq P$ such that:

- for every $p \in P_s$ we have $M_{|p} \models \varphi$,
- for every $p \in P \backslash P_s$ we have $M_{|p} \not\models \varphi$.

A counterexample for every $p \in P \backslash P_s$ is preferred.

If $P_s = P$, ie. all products satisfy φ, we write $M \models \varphi$.

8 Parity Games

8.1 Parity games

Definition 8.1. *[4] A parity game (PG) is a tuple (V, V_0, V_1, E, ρ), where:*

- *$V = V_0 \cup V_1$ and $V_0 \cap V_1 = \emptyset$,*
- *V_0 is the set of vertices owned by player 0,*
- *V_1 is the set of vertices owned by player 1,*
- *$E \subseteq V \times V$ is the edge relation,*
- *$\rho : V \to \mathbb{N}$ is a priority assignment.*

We write $\alpha \in \{0, 1\}$ to denote an arbitrary player. We write $\overline{\alpha}$ to denote α's opponent, ie. $\overline{0} = 1$ and $\overline{1} = 0$.

A parity game is played by players 0 and 1. A play starts with placing a token on vertex $v \in V$. Player α moves the token if the token is on a vertex owned by α, ie. $v \in V_\alpha$. The token can be moved to $w \in V$, with $(v, w) \in E$. A series of moves results in a sequence of vertices, called path. For path π we write π_i to denote the i^{th} vertex in path π. A play ends when the token is on vertex $v \in V_\alpha$ and α can't move the token anywhere, in this case player $\overline{\alpha}$ wins the play. If the play results in an infinite path π then we determine the highest priority that occurs infinitely often in this path, formally

$$\max\{p \mid \forall_j \exists_i j < i \land p = \rho(\pi_i)\}$$

If the highest priority is odd then player 1 wins, if it is even player 0 wins.

A strategy for player α is a function $\sigma : V^* V_\alpha \to V$ that maps a path ending in a vertex owned by player α to the next vertex. Parity games are positionally determined [4], therefore a strategy $\sigma : V_\alpha \to V$ that maps the current vertex to the next vertex is sufficient.

A strategy σ for player α is winning from vertex v if and only if any play that results from following σ results in a win for player α. The graph can be divided in two partitions $W_0 \subseteq V$ and

$W_1 \subseteq V$, called winning sets. If and only if $v \in W_\alpha$ then player α has a winnings strategy from v. Every vertex in the graph is either in W_0 or W_1 [4]. Furthermore finite parity games are decidable [4].

8.2 Featured parity games

Definition 8.2. *A featured parity game (FPG) is a tuple $(V, V_0, V_1, E, \rho, N, P, \gamma)$, where:*

- $V = V_0 \cup V_1$ *and* $V_0 \cap V_1 = \emptyset$,
- V_0 *is the set of vertices owned by player* 0,
- V_1 *is the set of vertices owned by player* 1,
- $E \subseteq V \times V$ *is the edge relation,*
- $\rho : V \to \mathbb{N}$ *is a priority assignment,*
- N *is a set of features,*
- $P \subseteq \mathcal{P}(N)$ *is a set of products, ie. feature assignments, for which the game can be played,*
- $\gamma : E \to \mathbb{B}(N)$ *is a total function, labelling each edge with a Boolean expression over the features.*

An FPG is played similarly to a PG, however the game is played for a specific product $p \in P$. Player α can only move the token from $v \in V_\alpha$ to $w \in V$ if $(v, w) \in E$ and $p \models \gamma(v, w)$.

A game played for product $p \in P$ results in winnings sets W_0^p and W_1^p, which are defined similar to the W_0 and W_1 winning sets for parity games.

Definition 8.3. *The projection from FPG $G = (V, V_0, V_1, E, \rho, N, P, \gamma)$ to a product $p \in P$, noted $G_{|p}$, is the parity game (V, V_0, V_1, E', ρ) where $E' = \{e \in E \mid p \models \gamma(e)\}$.*

Playing FPG G for a specific product $p \in P$ is the same as playing the PG $G_{|p}$. Any path that is valid in G for p is also valid in $G_{|p}$ and vice versa. Therefore the strategies are also interchangeable, furthermore the winning sets W_α for $G_{|p}$ and W_α^p for G are identical. Since parity games are positionally determined so are FPGs. Similarly, since finite parity games are decidable, so are finite FPGs.

8.3 Variability parity games

Definition 8.4. *A variability parity game (VPG) is a tuple $(V, V_0, V_1, E, \rho, \mathfrak{C}, \theta)$, where:*

- $V = V_0 \cup V_1$ *and* $V_0 \cap V_1 = \emptyset$,
- V_0 *is the set of vertices owned by player* 0,
- V_1 *is the set of vertices owned by player* 1,
- $E \subseteq V \times V$ *is the edge relation; we assume that E is total, i.e. for all $v \in V$ there is some $w \in V$ such that $(v, w) \in E$,*

- *$\rho : V \to \mathbb{N}$ is a priority assignment,*
- *$\mathfrak{C}$ is a finite set of configurations,*
- *$\theta : E \to \mathcal{P}(\mathfrak{C}) \setminus \{0\}$ is the configuration mapping, satisfying for all $v \in V$, $\bigcup\{\theta(v,w) | (v,w) \in E\} = \mathfrak{C}$.*

A VPG is played similarly to a PG, however the game is played for a specific configuration $c \in \mathfrak{C}$. Player α can only move the token from $v \in V_\alpha$ to $w \in V$ if $(v,w) \in E$ and $c \in \theta(v,w)$. Furthermore VPGs don't have deadlocks, every play results in an infinite path.

A game played for configuration $c \in \mathfrak{C}$ results in winning sets W_0^c and W_1^c, which are defined similar to the W_0 and W_1 winning sets for parity games.

Definition 8.5. *The projection from VPG $G = (V, V_0, V_1, E, \rho, \mathfrak{C}, \theta)$ to a configuration $c \in \mathfrak{C}$, noted $G_{|c}$, is the parity game (V, V_0, V_1, E', ρ) where $E' = \{e \in E | c \in \theta(e)\}$.*

Playing VPG G for a specific configuration $c \in \mathfrak{C}$ is the same as playing the PG $G_{|c}$. Any path that is valid in G for c is also valid in $G_{|c}$ and vice versa. Therefore the strategies are also interchangeable, furthermore the winning sets W_α for $G_{|c}$ and W_α^c for G are identical. Since parity games are positionally determined so are VPGs. Similarly, since finite parity games are decidable, so are finite VPGs.

8.4 Creating parity games

Originating from an FTS and a modal μ-calculus we can create an FPG (from which we can create a PG by projection) and from the FPG we can create a VPG. For a specific product we can project the FTS to an LTS, from which we can create a PG. The relation between the transition systems and games is displayed in the following diagram.

$$\begin{tikzcd}
\text{FTS} \arrow[r, "\varphi"] \arrow[d, Rightarrow, "\Pi"'] & \text{FPG} \arrow[r] \arrow[d, Rightarrow, "\Pi"] & \text{VPG} \\
\text{LTS} \arrow[r, "\varphi"] & \text{PG} &
\end{tikzcd}$$

The projections are defined in the previous section. In this section we will define the horizontal arrows in the diagram. First we show how to create a PG from an LTS and a modal μ-calculus formula, this part is well studied and the approach is based on [4].

Definition 8.6. *[4] LTS2PG(M, φ) converts LTS $M = (S, Act, trans, s_0)$ and closed formula φ to a PG (V, V_0, V_1, E, ρ).*

A vertex in the parity game is represented by a pair (s, ψ) where $s \in S$ and ψ is a modal μ-calculus formula. We will create a vertex for every state with every subformula of φ except subformula's of the form X. Furthermore we create a vertex for every state with the unfolding of every fixpoint formula. Formally we define the set of vertices by first defining the set of subformula's:

$$F = \{\psi \mid \psi \text{ is a subformula of } \varphi\}, \text{ note that } \varphi \text{ is a subformula of } \varphi,$$

the set of subformula's that are of the form X:

$$F_X = \{\psi \in F \mid \psi = X\}$$

and the set of unfolded fixedpoint subformula's:

$$F_\sigma = \{\psi[X := \sigma X.\psi] \mid \sigma X.\psi \in F \text{ with } \sigma \in \{\mu, \nu\}\}$$

Having these sets we can define the set of vertices by:

$$V = S \times (F \backslash F_X \cup F_\sigma)$$

We create the parity game with the smallest set E such that:

- $V = V_0 \cup V_1$,
- $V_0 \cap V_1 = \emptyset$ *and*
- *for every $v = (s, \psi) \in V$ we have:*
  - *If $\psi = \top$ then $v \in V_1$.*
  - *If $\psi = \bot$ then $v \in V_0$.*
  - *If $\psi = \psi_1 \vee \psi_2$ then:*
    $v \in V_0$,
    $(v, (s, \psi_1)) \in E$ *and*
    $(v, (s, \psi_2)) \in E$.
  - *If $\psi = \psi_1 \wedge \psi_2$ then:*
    $v \in V_1$,
    $(v, (s, \psi_1)) \in E$ *and*
    $(v, (s, \psi_2)) \in E$.
  - *If $\psi = \langle a \rangle \psi_1$ then $v \in V_0$ and for every $s \xrightarrow{a} s'$ we have $(v, (s', \psi_1)) \in E$.*
  - *If $\psi = [a]\psi_1$ then $v \in V_1$ and for every $s \xrightarrow{a} s'$ we have $(v, (s', \psi_1)) \in E$.*
  - *If $\psi = \mu X.\psi_1$ then $(v, (s, \psi_1[X := \mu X.\psi_1])) \in E$.*
  - *If $\psi = \nu X.\psi_1$ then $(v, (s, \psi_1[X := \nu X.\psi_1])) \in E$.*

Note that since φ is closed and we use unfolding there will never be an edge $(v, (s, X)) \in E$.

Finally we have $\rho(s, \psi) = \begin{cases} 2\lfloor adepth(X)/2 \rfloor & \text{if } \psi = \nu X.\psi' \\ 2\lfloor adepth(X)/2 \rfloor + 1 & \text{if } \psi = \mu X.\psi' \\ 0 & \text{otherwise} \end{cases}$

Next we define the transformation from FTS to FPG.

Definition 8.7. *FTS2FPG(M, φ) converts FTS $M = (S, Act, trans, s_0, N, P, \gamma)$ and closed formula φ to FPG $(V, V_0, V_1, E, \rho, N, P, \gamma')$.*

We have (V, V_0, V_1, E, ρ) = LTS2PG($(S, Act, trans, s_0), \varphi$) and

$$\gamma'((s, \psi), (s', \psi')) = \begin{cases} \gamma(s, a, s') & \text{if } \psi = \langle a \rangle \psi' \text{ or } \psi = [a]\psi' \\ \top & \text{otherwise} \end{cases}$$

Finally we define how to create a VPG from an FPG. This transformation abstracts from the notion of products and uses configurations for a syntactically more pleasant representation. Furthermore in VPGs deadlocks are removed, this is done by creating two losing vertices l_0 and l_1 such that player α loses when the token is in vertex l_α. Any vertex that can not move for a configuration will get an edge that is admissible for that configuration towards one of the losing vertices.

Definition 8.8. *FPG2VPG(G^F) converts FPG $G^F = (V^F, V_0^F, V_1^F, E^F, \rho^F, N, P, \gamma)$ to VPG $G = (V, V_0, V_1, E, \rho, \mathfrak{C}, \theta)$.*

Let P be defined as $\{p_0, p_1, \dots, p_m\}$, we define $\mathfrak{C} = \{c_0, c_1, \dots, c_m\}$.

We create vertices l_0 and l_1 and define $V_0 = V_0^F \cup \{l_0\}$, $V_1 = V_1^F \cup \{l_1\}$ and $V = V_0 \cup V_1$.

We construct E by first making $E = E^F$ and adding edges (l_0, l_0) and (l_1, l_1) to E. Simultaneously we construct θ by first making $\theta(e) = \{c_i \in \mathfrak{C} | p_i \models \gamma(e)\}$ for every $e \in E^F$. Furthermore $\theta(l_0, l_0) = \theta(l_1, l_1) = \mathfrak{C}$.

Next, for every vertex $v \in V_\alpha$ with $\alpha = \{0, 1\}$, we have $C = \mathfrak{C} \setminus \bigcup \{\theta(v, w) | (v, w) \in E\}$. If $C \neq \emptyset$ then we add (v, l_α) to E and make $\theta(v, l_\alpha) = C$. Finally we have

$$\rho(v) = \begin{cases} 1 & \textit{if } v = l_0 \\ 0 & \textit{if } v = l_1 \\ \rho^F(v) & \textit{otherwise} \end{cases}$$

8.5 Correctness

Theorem 8.1. *Given:*

- *FTS $M = (S, Act, trans, s_0, N, P, \gamma)$,*
- *a closed modal mu-calculus formula φ,*
- *a product $p \in P$*

it holds that he parity games LTS2PG($M_{|p}, \varphi$) and FTS2FPG($M, \varphi)_{|p}$ are identical.

Proof. Let G^F =FTS2FPG(M, φ)= $(V^F, V_0^F, V_1^F, E^F, \rho^F, N, P, \gamma')$, using definition 8.7, and $G^F_{|p} = (V^F, V_0^F, V_1^F, E^{F'}, \rho^F)$, using definition 8.3. Furthermore we have $M_{|p} = (S, Act, trans', s_0)$ and we let $G = \text{LTS2PG}(M_{|p}, \varphi) = (V, V_0, V_1, E, \rho)$. We depict the different transition systems and games in the following diagram.

$$\begin{tikzcd}
\text{FTS } M \arrow[rr, "\varphi"] \arrow[d, Rightarrow, "\Pi"'] & & \text{FPG } G^F \arrow[d, Rightarrow, "\Pi"] \\
\text{LTS } M_{|p} \arrow[r, "\varphi"] & \text{PG } G & \text{PG } G^F_{|p}
\end{tikzcd}$$

We will prove that $G = G^F_{|p}$. We first note that game G is created by

$$(V, V_0, V_1, E, \rho) = LTS2PG((S, Act, trans', s_0), \varphi)$$

and the vertices, edges and priorities of game G^F are created by

$$(V^F, V_0^F, V_1^F, E^F, \rho^F) = LTS2PG((S, Act, trans, s_0), \varphi)$$

Using the definition of LTS2PG (8.6) we find that the vertices and the priorities only depend on the states in S and the formula φ, since these are identical in the above two statements we immediately get $V = V^F, V_0 = V_0^F, V_1 = V_1^F$ and $\rho = \rho^F$. The vertices and priorities don't change when an FTS is projected, therefore $G_{|p}^F$ has the same vertices and priorities as G^F.

Now we are left with showing that $E = E^{F'}$ in order to conclude that that $G = G_{|p}^F$. We will do this by showing $E \subseteq E^{F'}$ and $E \supseteq E^{F'}$.

First let $e \in E$. Note that a vertex in the parity game is represented by a pair of a state and a formula. So we can write $e = ((s, \psi), (s', \psi'))$. To show that $e \in E^{F'}$ we distinguish two cases:

- If $\psi = \langle a \rangle \psi_1$ or $\psi = [a]\psi_1$ then there exists an $a \in Act$ such that $(s, a, s') \in trans'$. Using definition 6.1 we get $(s, a, s') \in trans$ and $p \models \gamma(s, a, s')$. Using definition 8.7 we find that $\gamma'((s, \psi), (s', \psi')) = \gamma(s, a, s')$ and therefore $p \models \gamma'((s, \psi), (s', \psi'))$. Now using definition 8.3 we find $((s, \psi), (s', \psi')) \in E^{F'}$.
- Otherwise the existence of the edge does not depend on the *trans* parameter and therefore $((s, \psi), (s', \psi')) \in E^{F'}$ if $(s, \psi) \in V^F$, since $V^F = V$ we have $(s, \psi) \in V^F$.

We can conclude that $E \subseteq E^{F'}$, next we will show $E \supseteq E^{F'}$. Let $e = ((s, \psi), (s', \psi')) \in E^{F'}$. We distinguish two cases:

- If $\psi = \langle a \rangle \psi_1$ or $\psi = [a]\psi_1$ then there exists an $a \in Act$ such that $(s, a, s') \in trans$. Using definition 8.3 we get $p \models \gamma'(s, a, s')$. Using definition 8.7 we get $p \models \gamma(s, a, s')$. Using the projection definition 8.7 we get $(s, a, s') \in trans'$ and therefore $((s, \psi), (s', \psi')) \in E$.
- Otherwise the existence of the edge does not depend on the *trans* parameter and therefore $((s, \psi), (s', \psi')) \in E$ if $(s, \psi) \in V$, since $V^F = V$ we have $(s, \psi) \in V$.

$\square$

Theorem 8.2. *Given:*

- *FTS $M = (S, Act, trans, s_0, N, P, \gamma)$,*
- *closed modal mu-calculus formula φ,*
- *product $p \in P$ and*
- *state $s \in S$*

it holds that $M_{|p}, s \models \varphi$ if and only if $(s, \varphi) \in W_0^p$ in FTS2FPG(M, φ).

Proof. The winning set W_α^p is equal to winning set W_α in FTS2FPG$(M, \varphi)_{|p}$ using definition 8.2. Using theorem 8.1 we find that the game FTS2FPG$(M, \varphi)_{|p}$ is equal to the game LTS2PG$(M_{|p}, \varphi)$, obviously their winning sets are also equal. Using the modal verification proof from [4] we know that $M_{|p}, s \models \varphi$ if and only if $(s, \varphi) \in W_0$. Winning set W_α^p is equal to W_α, therefore the theorem holds. $\square$

Theorem 8.3. *Given:*

- *FPG* $G^F = (V^F, V_0^F, V_1^F, E^F, \rho^F, N, \{p_0, p_1, \ldots, p_m\}, \gamma)$*,*
- *product* p_i*,*
- *player* $\alpha \in \{0, 1\}$

we have for winning sets $W_\alpha^{p_i}$ *in* G *and* $W_\alpha^{c_i}$ *in FPG2VPG(*G^F*) that* $W_\alpha^{p_i} \subseteq W_\alpha^{c_i}$*.*

Proof. Let $G = (V, V_0, V_1, E, \rho, \mathfrak{C}, \theta) = \text{FPG2VPG}(G^F)$. Consider finite play π that is valid in game G^F for product p_i. We have for every (π_i, π_{i+1}) in π that $(\pi_i, \pi_{i+1}) \in E^F$ and $p_i \models \gamma((\pi_i, \pi_{i+1}))$. From definition 8.8 it follows that $(\pi_i, \pi_{i+1}) \in E$ and $c_i \in \theta(\pi_i, \pi_{i+1})$. So we can conclude that path π is also valid in game G for configuration c_i. Since the play is finite the winner is determined by the last vertex v in π, player α wins such that $v \in V_{\overline{\alpha}}$. Furthermore we know, because the play is finite, that there exists no $(v, w) \in E^F$ with $p \models \gamma(v, w)$. From this we can conclude that $(v, l_{\overline{\alpha}}) \in E$ and $c_i \in \theta(v, l_{\overline{\alpha}})$. Vertex $l_{\overline{\alpha}}$ has one outgoing edge, namely to itself. So finite play π will in game G^F results in an infinite play $\pi(l_{\overline{\alpha}})^\omega$. Vertex $l_{\overline{\alpha}}$ has a priority with the same parity as player α, so player α wins the infinite play in G for configuration c_i.

Consider infinite play π that is valid in game G^F for product p_i. As shown above this play is also valid in game G for configuration c_i. Since the win conditions of both games are the same the play will result in the same winner.

Consider infinite play π that is valid in game G for configuration c_i. We distinguish two cases:

- If l_α doesn't occur in π then the path is also valid for game G^F with product p_i and has the same winner.
- If $\pi = \pi'(l_\alpha)^\omega$ then the winner is player $\overline{\alpha}$. The path π' is valid for game G^F with product p_i. Let vertex v be the last vertex of π'. Since $(v, l_\alpha) \in E$ and $c_i \in \theta(v, l_\alpha)$ we know that there is no $(v, w) \in E^F$ with $p_i \models \gamma(v, w)$ and that vertex v is owned by player α. So in game G^F player α can't move at vertex v and therefore loses the game (in which case the winner is also $\overline{\alpha}$.

We have shown that every path (finite or infinite) in game G^F with product p_i can be played in game G with configuration c_i and that they have the same winner. Furthermore every infinite path in game G with configuration c_i can be either played as an infinite path or the first part of the path can be played in G^F with product p_i and they have the same winner. From this we can conclude that the theorem holds. □

References

[1] J. F. Groote and M. R. Mousavi, *Modeling and Analysis of Communicating Systems.* The MIT Press, 2014.

[2] M. ter Beek, E. de Vink, and T. Willemse, "Family-based model checking with mcrl2," in *Fundamental Approaches to Software Engineering* (M. Huisman and J. Rubin, eds.), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), (Germany), pp. 387–405, Springer, 2017.

[3] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin, “Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking,” *IEEE Transactions on Software Engineering*, vol. 39, pp. 1069–1089, 2013.

[4] J. Bradfield and I. Walukiewicz, *The mu-calculus and Model Checking*, pp. 871–919. Cham: Springer International Publishing, 2018.

[5] A. Classen, P. Heymans, P. Y. Schobbens, A. Legay, and J.-P. Raskin, “Model checking lots of systems: Efficient verification of temporal properties in software product lines,” vol. 1, 01 2010.