Branching Bisimulation Games

David De Frutos Escrig, Jeroen A. Keiren, Tim C. Willemse

Branching Bisimulation Games

David de Frutos Escrig[1], Jeroen J.A. Keiren[2,3], and Tim A.C. Willemse[4]

[1] defrutos@sip.ucm.es
Dpto. Sistemas Informáticos y Computación - Facultad CC. Matemáticas
Universidad Complutense de Madrid, Spain.
[2] Jeroen.Keiren@ou.nl
Open University in the Netherlands
[3] Radboud University, Nijmegen, The Netherlands
[4] t.a.c.willemse@tue.nl
Eindhoven University of Technology, The Netherlands

Abstract. Branching bisimilarity and branching bisimilarity with explicit divergences are typically used in process algebras with silent steps when relating implementations to specifications. When an implementation fails to conform to its specification, *i.e.*, when both are not related by branching bisimilarity [with explicit divergence], pinpointing the root causes can be challenging. In this paper, we provide characterisations of branching bisimilarity [with explicit divergence] as games between Spoiler and Duplicator, offering an operational understanding of both relations. Moreover, we show how such games can be used to assist in diagnosing non-conformance between implementation and specification.

1 Introduction

Abstraction is a powerful, fundamental concept in process theories. It facilitates reasoning about the conformance between implementations and specifications of a (software) system, described by a transition system. Essentially, it allows one to ignore (*i.e.*, abstract from) implementation details that are unimportant from the viewpoint of the specification. While there is a wealth of behavioural equivalences (and preorders), each treating abstraction in slightly different manners, there are a few prototypical equivalences that have been incorporated in contemporary tool sets that implement verification technology for (dis)proving the correctness of software systems. These equivalences include branching bisimulation [19] and branching bisimulation with explicit divergence [18], which are both used in tool sets such as CADP [5], μCRL [2], and mCRL2 [4].

From a practical perspective, branching bisimulation and branching bisimulation with explicit divergence have pleasant properties. For instance, both relations are essentially *compositional*, permitting one to stepwise replace subcomponents in a specification with their implementations. Moreover, both types of branching bisimulation can be computed efficiently in $\mathcal{O}(n \cdot m)$, where n is the number of states in a transition system and m is the number of transitions [8]. A recently published algorithm improves this to $\mathcal{O}(m \log n)$ [9].

The key idea behind both kinds of branching bisimulation is that they abstract from 'internal' events (events that are *invisible* to the outside observer of a system) while, at the same time, they remain sensitive to the branching structure of the transition system. This means that these relations preserve both the essential, externally visible, computations and the potential future computations of all states. At the same time, this can make it difficult to explain why a particular pair of states is not branching bisimilar, as one must somehow capture the loss of potential future computations in the presence of internal actions. While (theoretical) tools such as distinguishing formulae can help to understand why two states are distinguishable, these are not very accessible and, to date, the idea of integrating such formulae in tool sets seems not to have caught on.

We address the above concern by providing game-based views on branching bisimulation and branching bisimulation with explicit divergence. More specifically, we show that both branching bisimulation and branching bisimulation with explicit divergence can be characterised by Ehrenfeucht-Fraïssé games [17]. This provides an alternative point of view on the traditional coinductive definitions of branching bisimulation and branching bisimulation with explicit divergence. Moreover, we argue, using some examples, how such games can be used to give an operational explanation of the inequivalence of states following the ideas in [15], thereby remaining closer to the realm of transition systems.

Related work. Providing explanations of the inequivalence of states for a given equivalence relation has a long tradition, going back to at least Hennessy and Milner's seminal 1980 work [10] on the use of modal logics for characterising behavioural equivalences. Modal characterisations (and by extension, distinguishing formulae) for branching bisimulation appeared first in [11] and for branching bisimulation with explicit divergence in [18]. An alternative line of research has led to game-based characterisations of behavioural equivalences. For instance, in [16], Stirling provides a game-based definition of Milner and Park's strong bisimulation [13]. More recently, Yin *et al.* describe a branching bisimulation game in the context of normed process algebra [20], but their game uses moves that consist of sequences of silent steps, rather than single steps. As argued convincingly by Namjoshi [12], local reasoning using single steps often leads to simpler arguments. A game-based characterisation of divergence-blind stuttering bisimulation (a relation for Kripke structures that in essence is the same as branching bisimulation), provided by Bulychev *et al.* in [3] comes closer to our work for branching bisimulation. However, their game-based definition is sound only for transition systems that are essentially free of divergences, so that in order to deal with transition systems containing divergences they need an additional step that precomputes and eliminates these divergences. Such a preprocessing step is a bit artificial, and makes it hard to present the user with proper diagnostics. As far as we are aware, ours is the first work that tightly integrates dealing with divergences in a game-based characterisation of a behavioural equivalence.

Structure of the paper. Section 2 introduces the necessary preliminaries. In Section 3, we present our game-based definitions of branching bisimulation and

branching bisimulation with explicit divergence and prove these coincide with their traditional, coinductive definitions. We illustrate their application in Section 4, while Section 5 shows how our results can be easily extended to the case of branching simulation. We conclude in Section 6.

2 Preliminaries

In this paper we are concerned with relations on labelled transition systems that include both *observable* transitions, and *internal* transitions labelled by the special action τ.

Definition 1. *A* Labelled Transition System *(LTS) is a structure* $L = \langle S, A, \rightarrow \rangle$ *where:*
- S *is a set of states,*
- A *is a set of actions containing a special action* τ*,*
- $\rightarrow \subseteq S \times A \times S$ *is the transition relation.*

As usual, we write $s \xrightarrow{a} t$ to stand for $(s, a, t) \in \rightarrow$. The reflexive-transitive closure of the $\xrightarrow{\tau}$ relation is denoted by $\twoheadrightarrow$. Given a relation $R \subseteq S \times S$ on states, we simply write $s\ R\ t$ to represent $(s, t) \in R$. We say that s is a *divergent* state if there is an infinite sequence $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \cdots$.

Branching bisimulation was introduced by van Glabbeek and Weijland in [19].

Definition 2 ([19]). *A symmetric relation* $R \subseteq S \times S$ *is said to be a* branching bisimulation *whenever for all* $s\ R\ t$*, if* $s \xrightarrow{a} s'$*, then there exist states* t', t'' *such that* $t \twoheadrightarrow t''$*, with* $s\ R\ t''$ *and* $s'\ R\ t'$*; and either* $t'' \xrightarrow{a} t'$*, or both* $a = \tau$ *and* $t' = t''$*. We write* $s \leftrightarroweq_b t$ *and say that* s *and* t *are branching bisimilar, iff there is a branching bisimulation* R *such that* $s\ R\ t$*. Typically we simply write* $\leftrightarroweq_b$ *to denote* branching bisimilarity*.*

Van Glabbeek *et al.* investigated branching bisimulations with explicit divergence in [18]. We here use one of their (many) equivalent characterisations:

Definition 3 ([18, Condition D2]). *A symmetric relation* $R \subseteq S \times S$ *is called a* branching bisimulation with explicit divergence *if and only if* R *is a branching bisimulation and for all* $s\ R\ t$*, if there is an infinite sequence* $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \cdots$*, then there is a state* t' *such that* $t \xrightarrow{\tau} t'$ *and for some* k*,* $s_k\ R\ t'$*. We write* $s \leftrightarroweq_b^{ed} t$ *iff there is a branching bisimulation with explicit divergence* R *such that* $s\ R\ t$*.*

Both kinds of branching bisimulations are equivalence relations.

Theorem 1 ([1, 18]). *Both* $\leftrightarroweq_b$ *and* $\leftrightarroweq_b^{ed}$ *are equivalence relations. Moreover they are the largest branching bisimulation and branching bisimulation with explicit divergence, respectively.*

Both branching bisimulation relations and branching bisimulation with explicit divergence relations have the *stuttering property* [18, Corollary 4.4]. This will be useful in several of the proofs in this paper.

Definition 4 ([18]). *A relation* R *has the* stuttering property *if, whenever* $t_0 \xrightarrow{\tau} t_1 \cdots \xrightarrow{\tau} t_k$ *with* $s\ R\ t_0$ *and* $s\ R\ t_k$*, then* $s\ R\ t_i$*, for all* $i \leq k$*.*

3 Branching Bisimulation Games

The games we consider in this section are instances of two-player infinite-duration games with ω-regular winning conditions, played on game arenas that can be represented by graphs. In these games each vertex is assigned to one of two players, here called SPOILER and DUPLICATOR. The players move a token over the vertices as follows. The player that 'owns' the vertex where the token is pushes it to an adjacent vertex, and this continues as long as possible, possibly forever. The winner of the play is decided from the resulting sequence of vertices visited by the token, depending on the predetermined winning criterion. We say that a player *can win from a given vertex* if she has a strategy such that any play with the token initially at that vertex will be won by her. The games that we consider here are *memoryless* and *determined*: every vertex is won by (exactly) one player, and the winning player has a *positional* winning strategy, so that she can decide her winning moves based only on the vertex where the token currently resides, without inspecting the previous moves of the play. These winning strategies can be efficiently computed while solving the game. We refer to [7] for a more in-depth treatment of the underlying theory.

3.1 The Branching Bisimulation Games

We start by presenting our game-based characterisation of branching bisimilarity. This will be extended to capture branching bisimulation with explicit divergence in Section 3.2.

Definition 5. *A* branching bisimulation (bb) game *on an LTS L is played by players* SPOILER *and* DUPLICATOR *on an arena of* SPOILER*-owned configurations $[\,(s,t),c,r\,]$ and* DUPLICATOR*-owned configurations $\langle\,(s,t),c,r\,\rangle$, where $((s,t),c,r) \in \mathit{Position} \times \mathit{Challenge} \times \mathit{Reward}$. Here $\mathit{Position} = S \times S$ is the set of* positions*, $\mathit{Challenge} = (A \times S) \cup \{\dagger\}$ is the set of* pending challenges*, and $\mathit{Reward} = \{*, \checkmark\}$ the set of* rewards*. By convention, we write $((s,t),c,r)$ if we do not care about the owner of the configuration.*

- SPOILER *moves from a configuration $[\,(s_0,s_1),c,r\,]$ by:*
  1. *selecting $s_0 \xrightarrow{a} s_0'$ and moving to $\langle\,(s_0,s_1),(a,s_0'),*\,\rangle$ if $c = (a,s_0')$, and to $\langle\,(s_0,s_1),(a,s_0'),\checkmark\,\rangle$, otherwise; or*
  2. *picking some $s_1 \xrightarrow{a} s_1'$ and moving to $\langle\,(s_1,s_0),(a,s_1'),\checkmark\,\rangle$.*
- DUPLICATOR *responds from a configuration $\langle\,(s_0,s_1),c,r\,\rangle$ by:*
  1. *not moving if $c = (\tau,s_0')$ and propose configuration $[\,(s_0',s_1),\dagger,\checkmark\,]$, or,*
  2. *if $c = (a,s_0')$, moving $s_1 \xrightarrow{a} s_1'$ if available and continue in configuration $[\,(s_0',s_1'),\dagger,\checkmark\,]$, or*
  3. *if $c \neq \dagger$, moving $s_1 \xrightarrow{\tau} s_1'$ if possible and continue in configuration $[\,(s_0,s_1'),c,*\,]$.*

DUPLICATOR *wins a finite play starting in a configuration $((s,t),c,r)$ if* SPOILER *gets stuck, and she wins an infinite play if the play yields infinitely many $\checkmark$ rewards. All other plays are won by* SPOILER*. We say that a configuration is*

won by a player when she has a strategy that wins all plays starting in it. Full *plays of the game start in a configuration* $[\,(s,t),\dagger,*\,]$*; we say that* DUPLICATOR *wins the bb game for a position* (s,t)*, if the configuration* $[\,(s,t),\dagger,*\,]$ *is won by it; in this case, we write* $s \equiv_b t$*. Otherwise, we say that* SPOILER *wins that game.*

Note that by definition both players strictly alternate their moves along plays.

Remark 1. Our branching bisimulation game definition resembles the divergence-blind stuttering bisimulation (dbsb) game definition [3] of Bulychev *et al.* Apart from the different computational models, there are two fundamental differences: we maintain SPOILER's pending challenges and DUPLICATOR's earned rewards, whereas the dbsb game does not, *and* our winning condition for DUPLICATOR requires an infinite number of ✓ rewards on infinite plays, whereas the dbsb game only requires DUPLICATOR not to get stuck. However, both games are equivalent when played on LTSs in which there are no divergences. Instead, there are transition systems with divergent states that show that, unlike our bb game, the rules of [3] fail to capture branching bisimulation, see Example 1.

Let us explain how our game works intuitively: by keeping track of pending challenges and earned rewards, we can distinguish between DUPLICATOR 'facilitating' *progress* (when choosing her first or second option) and DUPLICATOR *procrastinating* (when choosing her third option) when facing challenges presented by SPOILER. Procrastination is penalised by a $*$ reward, but progress is rewarded by a ✓ reward. On her account, SPOILER can either maintain a previously presented challenge, or change it if the challenge is still not totally *solved* by DUPLICATOR. In the latter case, SPOILER is penalised by rewarding DUPLICATOR with a ✓. This notion of pending challenge will be essential when extending the game so that it coincides with branching bisimulation with explicit divergence, as we will do in the next section. Omitting the concepts of pending challenges and earned rewards is what prevented extending the dbsb game to properly deal with divergent transition systems, and to (divergence sensitive) stuttering equivalence, in [3].

Before we prove that our bb game coincides with the classical co-inductive definition of branching bisimulation, we illustrate our game definition and a few of the subtleties we discussed above.

Example 1. Consider the LTS depicted in Figure 1. Observe that s_0 and t_0 are branching bisimilar. Suppose SPOILER tries (in vain) to disprove that s_0 and t_0 are branching bisimilar and challenges DUPLICATOR by playing $s_0 \xrightarrow{a} c_1$. DUPLICATOR may respond with an infinite sequence of τ-steps, moving between t_0 and t_1, so long as SPOILER sticks to her challenge. In this way she would win the play following the rules in [3], but such *procrastinating* behaviour of DUPLICATOR is not rewarded in our game. Instead, DUPLICATOR has to eventually move to c_1, matching the challenge, if she wants to win the play.

Now suppose SPOILER tries to disprove (again in vain) that s_0 and t_0 are branching bisimilar, and challenges DUPLICATOR by playing $t_0 \xrightarrow{\tau} t_1$. The only response for DUPLICATOR is not to move at all, which completes the pending challenge, turning it into †, thus generating the new configuration $[\,(s_0,t_1),\dagger,\checkmark\,]$.

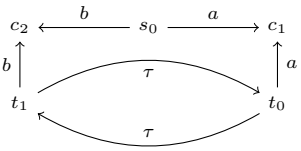


Fig. 1. LTS illustrating some consequences and subtleties of using challenges.

Spoiler may then challenge Duplicator by playing $t_1 \xrightarrow{\tau} t_0$, and Duplicator can again respond by not moving. The infinite play that is produced is winning for Duplicator, even if an infinite sequence of τ-steps proving the divergence of t_0 has been matched by Duplicator by staying totally idle, since Duplicator got infinitely many ✓s. Of course, things will be different when divergences will be taken into account in Section 3.2, since t_0 is divergent, whereas s_0 is not.

Before proving our first main theorem stating that two states are branching bisimilar just whenever Duplicator wins the associated game, we present two auxiliary results relating the winning configurations for this player.

Proposition 1. *Configurations* $[\,(s,t),c,*\,]$ *and* $[\,(s,t),c,\checkmark\,]$ *are both won by the same player. Likewise, configurations* $\langle\,(s,t),c,*\,\rangle$ *and* $\langle\,(s,t),c,\checkmark\,\rangle$ *are both won by the same player.*

Proof. This follows immediately from the Büchi winning condition: any player that wins some suffix of an infinite play also wins the infinite play itself. Furthermore, note that neither Spoiler nor Duplicator can get stuck playing a game by changing a reward from $*$ to ✓ or *vice versa*. □

Definition 6. *We say that a configuration* $((s,t),c,r)$ *is consistent when either* $c = \dagger$*, or* $c = (a,s')$ *for some* a, s' *such that* $s \xrightarrow{a} s'$ *in the given LTS.*

Proposition 2. *If* Duplicator *wins a consistent configuration* $[\,(s,t),c,r\,]$*, then* Duplicator *wins all consistent configurations* $[\,(s,t),c',r'\,]$*.*

Proof. Let $[\,(s,t),c,r\,]$ be a Spoiler-owned consistent configuration that is won by Duplicator. Towards a contradiction, assume Spoiler wins some consistent configuration $[\,(s,t),c',r'\,]$. Suppose Spoiler's winning strategy involves playing to configuration $\langle\,(s,t),c'',r''\,\rangle$. Then from $[\,(s,t),c,r\,]$, Spoiler can force play to configuration $\langle\,(s,t),c'',*\,\rangle$ or $\langle\,(s,t),c'',\checkmark\,\rangle$: if $c = \dagger$, then she can simply choose challenge c'' while, if $c = (a,s')$, she can change her challenge to c''. But this leads to a contradiction: by Proposition 1, both configurations are won by Spoiler, once $\langle\,(s,t),c'',r''\,\rangle$ is won by Spoiler. So Duplicator wins any Spoiler-owned consistent configuration $[\,(s,t),c',r'\,]$. □

We next prove that the bb game captures branching bisimilarity. We split the proof obligations and prove both implications separately. First, we show that branching bisimilar states induce positions that are won by Duplicator in the bb game.

Lemma 1. *If $s \leftrightarrows_b t$ then $s \equiv_b t$.*

Proof. We have to design a winning strategy for DUPLICATOR for the game that starts in $[\,(s,t),\dagger,*\,]$. We will call the consistent configurations $((s',t'),c,r)$ corresponding to a position (s',t'), with $s' \leftrightarrows_b t'$, *good* configurations (for player DUPLICATOR). Let us first see that whenever SPOILER makes a move from a good configuration $[\,(s',t'),c',r'\,]$, then DUPLICATOR can reply with a move to another good configuration. We distinguish cases according to the move selected by SPOILER.

Assume SPOILER plays according to her first option and chooses a transition $s' \xrightarrow{a} s''$. We distinguish cases depending on the nature of the executed action:

1. if $a = \tau$ and $s'' \leftrightarrows_b t'$, then DUPLICATOR can play choosing her first option getting the configuration $[\,(s'',t'),\dagger,\checkmark\,]$, which clearly is good for her.
2. if $a \neq \tau$ or $s'' \not\leftrightarrows_b t'$, then there exist states t'_k, t'' such that $t' \twoheadrightarrow t'_k$, $s' \leftrightarrows_b t'_k$, $s'' \leftrightarrows_b t''$ and $t'_k \xrightarrow{a} t''$. Next we consider the length of the sequence of transitions that generates $t' \twoheadrightarrow t'_k$. If this length is zero, then DUPLICATOR can directly use her second option selecting the transition $t' \xrightarrow{a} t''$ that generates $[\,(s'',t''),\dagger,\checkmark\,]$, which is clearly good for her. If instead the sequence is not empty, then she can select the first transition $t' \xrightarrow{\tau} t'_1$ of this sequence, and applying the stuttering property we have $s' \leftrightarrows_b t'_1$. Therefore, when DUPLICATOR moves according to her third option, this produces configuration $[\,(s',t'_1),(a,s''),*\,]$, which is also good.

If SPOILER plays her second option, then the strategy DUPLICATOR uses is the same that she would have used if SPOILER had played her first option from configuration $[\,(t',s'),c',r'\,]$.

When playing in this way, DUPLICATOR will never get stuck, so that next it suffices to argue that she can select her moves as above in such a way that the generated play will contain an infinite number of ✓ rewards. It is clear that the contrary could only happen if 1) SPOILER sticks to some fixed challenge (a,s'') forever, as changing challenges is penalised with a ✓; and 2) DUPLICATOR replies generating a divergent sequence, *i.e.* choosing her third option, never earning a ✓. But DUPLICATOR can simply avoid generating such a sequence if the first time that the challenge is presented to her she selects any sequence $t' \twoheadrightarrow t'_k$ as stated above, and then she plays by executing one by one the transitions in it, finally concluding by executing $t'_k \xrightarrow{a} t''$, that will produce a new ✓, thus generating the desired play with infinitely many ✓ challenges. □

Lemma 2. *The relation $\equiv_b$ is a branching bisimulation.*

Proof. First, observe that $\equiv_b$ is obviously symmetric, since starting from configuration $[\,(s,t),\dagger,*\,]$, SPOILER can propose exactly the same challenges as when starting from $[\,(t,s),\dagger,*\,]$, and the infinite suffixes of the resulting plays will therefore be identical, leading to the same winners.

Pick arbitrary s,t such that $s \equiv_b t$ and assume $s \xrightarrow{a} s'$. Let us see that $\equiv_b$ meets the transfer condition. Since DUPLICATOR has a winning strategy from

$[\,(s,t),\dagger,*\,]$, she has a winning move when Spoiler proposes the move $s \xrightarrow{a} s'$ and configuration $\langle\,(s,t),(a,s'),*\,\rangle$. We distinguish cases based on Duplicator's response in this winning strategy:

- Duplicator replies according to her first option, by not making a move, producing the configuration $[\,(s',t),\dagger,\checkmark\,]$. Then we have $s' \equiv_b t$, and the transfer condition can be satisfied by choosing $t'' = t' = t$.
- Duplicator replies following her second option, thus selecting $t \xrightarrow{a} t'$ to continue from the configuration $[\,(s',t'),\dagger,\checkmark\,]$. This means that $s' \equiv_b t'$, so that the transfer condition is satisfied by taking $t'' = t$, since obviously $s \equiv_b t''$ and $s' \equiv_b t'$.
- Duplicator replies following her third option, thus selecting $t \xrightarrow{\tau} t'_1$ to continue from configuration $[\,(s,t'_1),(a,s'),*\,]$. This configuration is again won by Duplicator, and then applying Proposition 2 we also have $s \equiv_b t'_1$. Now, Spoiler could maintain the challenge (a,s'), and then the procedure can be repeated with Duplicator responding with her third move, until she can eventually play the second move, in order to get the reward that she eventually must be able to get, since she is playing a winning strategy. This final move by Duplicator will correspond to a transition $t'_k \xrightarrow{a} t'$, and will produce the configuration $[\,(s',t'),\dagger,\checkmark\,]$. Moreover, we had $s \equiv_b t'_k$, so that taking $t'' = t'_k$ the transfer condition is again satisfied.

So R is a branching bisimulation relation. □

From the above lemmata, the following theorem follows immediately.

Theorem 2. *We have* $\leftrightarrows_b = \equiv_b$.

3.2 The Branching Bisimulation with Explicit Divergence Games

The results in the previous section demonstrate that maintaining pending challenges and earned rewards in the game play, and properly dealing with these in the winning condition, leads to an equivalence relation on states that coincides with branching bisimulation. It does not yet give rise to an equivalence that is sensitive to divergences. In fact, in Example 1 we already saw a pair of states s_0 and t_0 for which we have $s_0 \leftrightarrows_b t_0$, and therefore $s_0 \equiv_b t_0$, while instead $s_0 \not\leftrightarrows^{ed}_b t_0$.

As we argued in the previous section, by including challenges and rewards, our winning condition is able to reject plays in which Duplicator procrastinates forever. This addresses a part of the divergence problem: Duplicator cannot try to 'prove' two states equivalent modulo branching bisimulation simply by diverging when Spoiler does not ask for a divergence. However, Duplicator is still capable of matching a challenge of Spoiler that consists of a divergence by *not* diverging. Capturing explicit divergences can therefore only be achieved by clearly indicating *when* Duplicator replied to an internal move with another one, instead of just remaining idle. In the game definition we present below, we essentially do so by rewarding Duplicator in a new way only whenever she just

properly responded with a matching move. Note that the changes required are subtle: assigning rewards differently would probably lead to different relations.

Definition 7. *A* branching bisimulation with explicit divergence (bbed) game *on an LTS L is played by players* Spoiler *and* Duplicator *on an arena of* Spoiler*-owned configurations $[\,(s,t),c,r\,]$ and* Duplicator*-owned configurations $\langle\,(s,t),c,r\,\rangle$, where $((s,t),c,r) \in \mathit{Position} \times \mathit{Challenge} \times \mathit{Reward}$. Here $\mathit{Position} = S \times S$ is the set of* positions*, $\mathit{Challenge} = (A \times S) \cup \{\dagger\}$ is the set of* pending challenges*, and $\mathit{Reward} = \{*, \checkmark\}$ the set of* rewards*. We again use the convention to write $((s,t),c,r)$ if we do not care about the owner of the configuration.*

- Spoiler *moves from a configuration $[\,(s_0,s_1),c,r\,]$ by:*
  1. *selecting $s_0 \xrightarrow{a} s_0'$ and moving to $\langle\,(s_0,s_1),(a,s_0'),*\,\rangle$ if $c = (a,s_0')$ and $\langle\,(s_0,s_1),(a,s_0'),\checkmark\,\rangle$ otherwise; or*
  2. *picking some $s_1 \xrightarrow{a} s_1'$ and moving to $\langle\,(s_1,s_0),(a,s_1'),\checkmark\,\rangle$.*
- Duplicator *responds from a configuration $\langle\,(s_0,s_1),c,r\,\rangle$ by:*
  1. *not moving if $c = (\tau,s_0')$ and propose configuration $[\,(s_0',s_1),\dagger,*\,]$, or,*
  2. *if $c = (a,s_0')$, moving $s_1 \xrightarrow{a} s_1'$ if available and continue in configuration $[\,(s_0',s_1'),\dagger,\checkmark\,]$, or*
  3. *if $c \neq \dagger$, moving $s_1 \xrightarrow{\tau} s_1'$ if possible and continue in configuration $[\,(s_0,s_1'),c,*\,]$.*

Duplicator *wins a finite play starting in a configuration $((s,t),c,r)$ if* Spoiler *gets stuck, and she wins an infinite play if the play yields infinitely many ✓ rewards. All other plays are won by* Spoiler*. We say that a configuration is won by a player when she has a strategy that wins all plays starting in it.* Full *plays of the game start in a configuration $[\,(s,t),\dagger,*\,]$; we say that* Duplicator *wins the bbed game for a position (s,t), if the configuration $[\,(s,t),\dagger,*\,]$ is won by it; in this case, we write $s \equiv_b^{ed} t$. Otherwise, we say that* Spoiler *wins that game.*

In order to understand how the new game works, note first that it is a (quite subtle!) *refinement* of the bb game. To be exact, only the first option in the description of Duplicator's moves is changed, simply turning the previously obtain reward ✓ into $*$, thus reducing the set of plays that are won by this player. As a consequence, any play Duplicator wins in the bbed game is also won by her in the bb game. Moreover, the original bb game can be recovered from the bbed game by weakening the winning condition of the latter as follows: an infinite play is won by Duplicator if the play yields infinitely many ✓ rewards or † challenges.

In contrast to the bb game, Duplicator now only earns a ✓ reward when she fully satisfies a pending challenge (choosing her second option): she is now punished for choosing to not move (*i.e.* whenever she chooses her first option). As a result, whenever Duplicator is confronted with an infinite sequence of τ-challenges produced by Spoiler, effectively creating a divergent computation, Duplicator can no longer win such a play by choosing to stay put. Instead, Duplicator will need to collect a ✓ mark from time to time, so that in the end she will be able to exhibit an infinite number of such marks.

Example 2. Reconsider the LTS in Figure 1. In Example 1, we argued that Spoiler was not able to win the bb game starting in position (s_0, t_0). Now reconsider Spoiler's strategy to challenge Duplicator, by playing $t_0 \xrightarrow{\tau} t_1$ in the bbed game. As before, Duplicator's only option is not to move. However, by not moving, Duplicator discharges Spoiler's (local) challenge, but she does not earn any ✓ reward. Clearly, Spoiler can then challenge Duplicator by playing $t_1 \xrightarrow{\tau} t_0$ in the bbed game, thereby forcing Duplicator to engage in an infinite play in which she earns no ✓ reward, thus losing the game.

The above example suggests that, indeed, the reconsideration of challenges and rewards leads to a game in which Spoiler can explicitly check divergences. We next prove that the relation induced by the bbed game exactly captures branching bisimilarity with explicit divergence. We split the proof obligations into three separate lemmata.

Lemma 3. *If* $s \leftrightarrows_b^{ed} t$ *then* $s \equiv_b^{ed} t$.

Proof. We again need to design the winning strategy for Duplicator for the bbed game that starts in $[(s,t), \dagger, *]$. Since $s \leftrightarrows_b^{ed} t$ implies $s \leftrightarrows_b t$, she could use the strategy defined in the proof of Lemma 1 to win the corresponding bb game. However, if we do not change anything in this strategy, it could be the case that Spoiler now wins the bbed game, since the strategy does not take divergences into account. Let us see which changes are needed to guarantee that Duplicator will also win the bbed game.

First, note that all the *positions* along any play consistent with that winning strategy for Duplicator contain two $\leftrightarrows_b$ equivalent states, as we proved in Lemma 2. Second, observe that we start from a configuration $[(s,t), \dagger, *]$ containing two $\leftrightarrows_b^{ed}$ equivalent states, and in order to be able to repeat our arguments after any move of Duplicator, we need to preserve that relation, and not just $\leftrightarrows_b$, as in the proof of Lemma 1.

Concerning this new requirement, note that Duplicator's winning strategy designed to prove that lemma was based on $\leftrightarrows_b$, but it is easy to see that now we can base it on $\leftrightarrows_b^{ed}$, so that the new winning strategy will preserve $\leftrightarrows_b$ along the plays of that game that are consistent with that strategy.

If we apply this strategy to the bbed game, the only case in which player Duplicator loses the game is that in which she is generating infinitely many † challenges, but only finitely many ✓ rewards. In particular, there would be some suffix of a play in which Duplicator generates infinitely many † challenges, and earns no ✓ reward. Next we consider that suffix as a full play and make a few observations about the moves played by both players along it:

- Spoiler never plays her second move;
- Duplicator never plays her second move;
- Duplicator never plays her third move,

since in the first two cases, Duplicator would *immediately* earn a ✓ reward, while in the third case, Duplicator will, by definition of the strategy used in the proof of Lemma 1, *eventually* earn a ✓ reward after a finite sequence of τ-moves.

Since Duplicator is always playing her first move, all challenges involved in the infinite suffix concern τ actions; moreover, all rewards on this suffix are $*$ rewards. Now observe that this infinite sequence of τ successors of s_0 consists of states that are all $\leftrightarrows_b^{ed}$-related to the state t_0 Duplicator chooses to stay put in. But then, by definition of $\leftrightarrows_b^{ed}$, there must be some transition $t_0 \xrightarrow{\tau} t'$ such that for some k, $s_k \leftrightarrows_b^{ed} t'$, and then Duplicator can reply playing $t_0 \xrightarrow{\tau} t_1$, instead of choosing her first option, thus collecting the needed ✓ reward, and the play will continue from $[\,(s_k, t'), \dagger, \checkmark\,]$.

Then, we will change the choice selected by Duplicator whenever the situation above appears, and in this way we get a revised strategy that will allow her to win the bbed game that starts in $[\,(s,t), \dagger, *\,]$, thus proving $s \equiv_b^{ed} t$. □

Lemma 4. *The relation $\equiv_b^{ed}$ is a branching bisimulation.*

Proof. As stated above, the bbed game is a refinement of the bb game: any configuration that is won in the bbed game is also won in the bb game. Hence, we can repeat the reasoning in the proof of Lemma 2 substituting the ✓ reward by a $*$ reward whenever Duplicator resorts to choosing her first option, to obtain the proof that $\equiv_b^{ed}$ is a branching bisimulation. □

The lemma below confirms that the relation induced by a bbed game is indeed sensitive to divergences.

Lemma 5. *Let $s \equiv_b^{ed} t$, and assume that we have a divergent sequence $s = s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots$. Then $t \xrightarrow{\tau} t'$ for some t' such that for some k, $s_k \equiv_b^{ed} t'$.*

Proof. Let us suppose that for all $t \xrightarrow{\tau} t'$, and for all k, we have $s_k \not\equiv_b^{ed} t'$. Consider Spoiler's strategy that starts the game from $[\,(s_0, t), \dagger, *\,]$ by making the move $s_0 \xrightarrow{\tau} s_1$. Then Duplicator cannot reply moving to a τ-successor of t, so that she has to play choosing her first option, which produces the configuration $[\,(s_i, t), \dagger, *\,]$. Next Spoiler will play each of the moves $s_i \xrightarrow{\tau} s_{i+1}$ in a row, and in all the cases Duplicator needs to stay idle, producing the configurations $[\,(s_i, t), \dagger, *\,]$, that generate an infinite play without ✓ rewards. Hence, Spoiler's strategy is winning for this bbed game, which contradicts the assumption that $s \equiv_b^{ed} t$. □

Theorem 3. *We have $\leftrightarrows_b^{ed} = \equiv_b^{ed}$.*

Proof. The inclusion $\leftrightarrows_b^{ed} \subseteq \equiv_b^{ed}$ follows from Lemma 3. For the reverse, observe that $\equiv_b^{ed}$ is a branching bisimulation with explicit divergence relation, since by Lemma 4 it is a branching bisimulation, that also fulfils the added obligation concerning divergences, as proved by Lemma 5. □

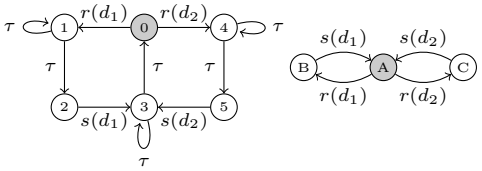
4 Some Small Applications

4.1 A Simple Application

The game-based definitions of branching bisimulation and branching bisimulation with explicit divergence provide an alternative, more dynamic view, on the

standard coinductive definitions of these relations. A major benefit of any game-based characterisation of an equivalence relation is that it lends itself to explain, in a natural way, why two states in an LTS are not equivalent, when that is the case. Such an explanation is drawn directly from Spoiler's winning strategy in the branching bisimulation game. We illustrate this by showing how one can prove that an abstraction of a communication protocol over unreliable channels differs from a simple one-place buffer.

Example 3. Consider two LTSs below. The leftmost LTS models the abstraction of an implementation of a simple communication protocol for exchanging two types of messages (d_1 and d_2), using a system of acknowledgements to cater for the unreliability introduced by a lossy/corrupting channel between sending and receiving parties. The LTS depicted below on the right models a simple specification of a one-place buffer for exchanging these two types of messages.



These LTSs are not branching bisimilar with explicit divergence. Since both *are* branching bisimilar, the difference between them must be in the lack of divergence in the specification. This is captured by Spoiler's winning strategy when playing the bbed game starting on $[\,(A,0),\dagger,*\,]$, Spoiler can play against the designer of the implementation in a way similar to that in [15], allowing the designer to better understand the mistake. Such a play could proceed as is shown below:

```
Spoiler moves A --r(d1)--> B
You respond with 0 --r(d1)--> 1
Spoiler switches positions and moves 1 --tau--> 1
You respond by not moving
...
Spoiler moves 1 --tau--> t
You respond by not moving
You explored all options. You lose.
```

Likewise, one can check that states B and 2 are not branching bisimilar with explicit divergence.

An alternative to illustrating the inequivalence between two states is through the use of a distinguishing formula. However, in many cases the nature of these formulae is rather 'descriptive' and requires a thorough understanding of modal logics, in order to understand its meaning. Instead, the game-based approach stays closer to the operational nature of LTSs. Moreover, the distinguishing formulae can become rather unwieldy, easily spanning several lines for states that are inequivalent for non-trivial reasons. The complexity of this approach is already illustrated by the following example, taken from [11].

Our game-based approach to distinguishing states 0 and A (in this case also under plain branching bisimulation equivalence) would start by SPOILER challenging by moving $0 \xrightarrow{a} 1$, to which DUPLICATOR can only respond by moving $A \xrightarrow{\tau} B$. Now, continuing from $[\,(0, B), (a, 1), *\,]$ SPOILER plays her second option and challenges DUPLICATOR to mimic move $0 \xrightarrow{b} 4$, something that DUPLICATOR cannot match.

The distinguishing formula given in [11] is $\neg((tt\langle b\rangle tt)\langle a\rangle tt)$, which holds at state A, but not at state 0. It explains that states 0 and A are inequivalent because state 0 may "engage in an a-step, while in all intermediate states (state 0 in this case) a b-step is available" [11], whereas this is not true of state A.

4.2 A More Elaborate Application

We illustrate how one can prove/argue interactively that the Alternating Bit Protocol with two messages differs (modulo branching bisimulation with explicit divergence) from a simple one-place buffer.

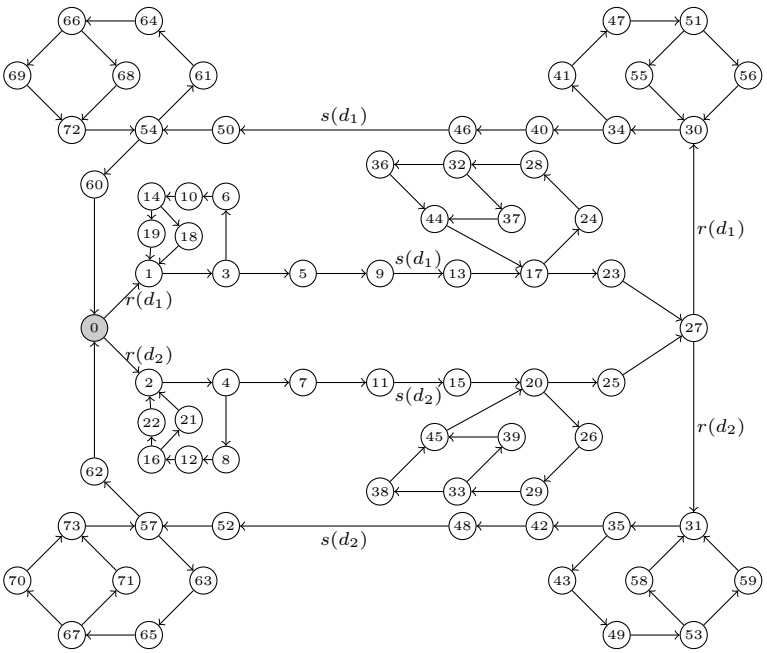


Fig. 2. The ABP with two messages; unlabelled transitions are τ transitions.

Example 4. Reconsider the one-place buffer for exchanging two different types of messages (d_1 and d_2), as specified in Example 3. Suppose one tries to implement this one-place buffer using the Alternating Bit Protocol (see Figure 2), only to find out that states A and 0 are not branching bisimilar with explicit divergences. In this case, SPOILER's winning strategy can be used to 'play' against the designer of the implementation in a way similar to that of [15], allowing the designer to better understand the reason why this implementation is not satisfactory. By solving the automatically generated game we obtain the following winning strategy for player SPOILER, that proceeds as follows:

```
Spoiler moves A --r(d1)--> B
You respond with 0 --r(d1)--> 1
Spoiler switches positions and moves 1 --tau--> 3
You respond by not moving
...
Spoiler moves 19 --tau--> 1
You respond by not moving
You explored all options. You lose.
```

In a similar vein, one can check also that states B and 9 are not branching bisimilar with explicit divergence.

5 Branching Simulation Games

In this paper we have considered branching bisimulation [with explicit divergence]. Both relations are equivalence relations. When checking an implementation relation, sometimes it is desirable to drop this symmetry requirement, and use simulation relations, rather than bisimulation relations.

Whereas branching similarity has been studied before, see, *e.g.* [6], we are not aware of an exact simulation variant of branching bisimulation with explicit divergence, although the notion of divergence preserving branching simulation defined in [14] comes quite close.

A branching simulation game can be obtained from Definition 5 by disallowing SPOILER to choose her second option. The proof of the fact that the resulting preorder coincides with branching similarity proceeds along the same lines of that of Theorem 2. If we reconsider the example we took from [11] in Section 4.1, we note that state 0 is not branching simulated by state A, which can be proved following the same arguments as used in that section. Instead, state A is branching simulated by state 0, as the last can copy any move from the former, eventually arriving at states that are trivially equivalent.

A game characterisation of branching simulation equivalence can equally straightforwardly be obtained from our definitions, by only allowing SPOILER to choose her second option for her moves during the first round of the game, and disallowing this option in any subsequent rounds. Of course, the corresponding simulation equivalence relation that one obtains in this way is coarser than the corresponding bisimulation: SPOILER has a much bigger power if she can switch the board at any round. Similarly, from Definition 7 we could obtain

games for branching simulation with explicit divergence and the corresponding simulation equivalence by restricting SPOILER's options.

6 Discussion & Future Work

In this paper we introduced game-theoretic definitions of branching bisimulation [with explicit divergence]. Compared to previous work, no transitive closure of τ-transitions is needed in the game definition, so that we obtain a much more "local" assessment when two states are declared to be not equivalent. Additionally, divergence is dealt with as a first-class citizen: no precomputation of divergences, and subsequent modification of the game, is needed. The combination of these aspects leads to a game characterisation that enables diagnostics that apply directly to the original labelled transition systems.

Future work. We have experimented with a prototype of the game-theoretic definitions of branching bisimulation (also with explicit divergence); we intend to make a proper implementation available in the mCRL2 tool set [4]. We leave further evaluating the effectiveness of the counterexamples described in this paper to future work. Furthermore, it can be investigated whether our approach of dealing with internal transitions extends to other behavioural equivalences, such as weak (bi)simulation.

References

1. T. Basten. Branching bisimilarity is an equivalence indeed! *Inform. Process. Lett.*, 58(3):141–147, May 1996.
2. S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *Proc. CAV'01*, volume 2102 of *LNCS*, pages 250–254, 2001.
3. P.E. Bulychev, I.V. Konnov, and V.A. Zakharov. Computing (bi)simulation relations preserving CTL*-X for ordinary and fair Kripke structures. *Inst. for Syst. Progr., Russian Acad. of Sci., Math. Meth. and Algor.*, 12, 2007.
4. S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, J.W. Wesselink, and T.A.C. Willemse. An overview of the mCRL2 toolset and its recent advances. In *Proc. TACAS'13*, volume 7795 of *LNCS*, pages 199–213, 2013.
5. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *Int. Journ. on Softw. Tools for Techn. Transfer*, 15(2):89–107, April 2013.
6. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Inform. and Comput.*, 150(2):132–152, 1999.
7. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.
8. J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching and stuttering equivalence. In *ICALP'90*, volume 443 of *LNCS*, pages 626–638. Springer, 1990.

9. J.F. Groote and A. Wijs. An O(m log n) algorithm for stuttering equivalence and branching bisimulation. In *Proc. TACAS'16*, volume 9636 of *LNCS*, pages 607–624. Springer, 2016.
10. M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *Proc. ICALP'80*, volume 85, pages 299–309. Springer, 1980.
11. H. Korver. Computing distinguishing formulas for branching bisimulation. In *Proc. CAV'92*, volume 575, pages 13–23. Springer, 1992.
12. K.S. Namjoshi. A simple characterization of stuttering bisimulation. In *Proc. FSTTCS'97*, volume 1346 of *LNCS*, pages 284–296. Springer, 1997.
13. D. Park. Concurrency and automata on infinite sequences. In *Proc. TCS'81*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.
14. M.A. Reniers, R. Schoren, and T.A.C. Willemse. Results on embeddings between state-based and event-based systems. *Comput. J.*, 57(1):73–92, 2014.
15. P. Stevens and C. Stirling. Practical model-checking using games. In *Proc. TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.
16. C. Stirling. Modal and temporal logics for processes. *Logics for concurrency: structure versus automata*, 1043:149–237, 1996.
17. W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In *Proc. TAPSOFT'93*, volume 668 of *LNCS*, pages 559–568. Springer, 1993.
18. R.J. van Glabbeek, S.P. Luttik, and N. Trçka. Branching bisimilarity with explicit divergence. *Fundam. Inform.*, 93(4):371–392, 2009.
19. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, May 1996.
20. Q. Yin, Y. Fu, C. He, M. Huang, and X. Tao. Branching bisimilarity checking for PRS. In *Proc. ICALP'14*, volume 8573 of *LNCS*, pages 363–374. Springer, 2014.