Bisimulation and Language Equivalence

Colin Stirling
Division of Informatics
University of Edinburgh

email: cps@dcs.ed.ac.uk

1 Introduction

One way to understand an interactive system is firmly rooted in language theory, that a system is its set of runs (or words). Properties of systems are described in a linear time temporal logic. Relationships between automata, language theory and logic are then utilised, such as the theory of ω-regular languages and Büchi automata.

An alternative viewpoint is that an interactive system should be understood as its capability for interacting with other systems. Language and automata theory then have less relevance because a more intensional account of system behaviour is needed than that given by sets of words. Bisimulation equivalence has a pivotal role within this approach.

Bisimulation is a rich concept which appears in various areas of theoretical computer science. Besides its origin for understanding concurrency, it was independently developed in the context of modal logic. In this paper we make some contrasts between bisimulation equivalence and language equivalence. There are two threads. First is that because bisimulation is more intensional, results in language and automata theory can be recast for bisimulation. The second thread is the contrast between definability of language equivalence and bisimulation equivalence. Bisimulation equivalence is definable as a "simple" formula in first-order logic with fixed points. Language equivalence is not definable as an unconditional projection of simple least fixed point. This should be contrasted with a known normal form result for least fixed point logic: any least fixed point definable relation is definable as a projection of a simple least fixed point under equality conditions on its components. It should be noted that undefinability of language equivalence in least fixed point logic per se would actually imply P $\neq$ NP. In section 2 we consider the two origins of bisimulation. In section 3 we describe some results which contrast bisimulation equivalence and language equivalence on automata. The final two sections discuss logics and the undefinability result.

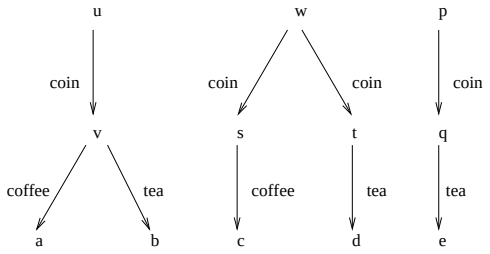


Figure 1: Simple transition graphs

2 Background

Labelled transition systems are commonly encountered in operational semantics of programs and systems. They are just labelled graphs. A transition system is a pair $G = (S, \{\stackrel{a}{\longrightarrow} : a \in A\})$ where S is a non-empty set (of states), A is a non-empty set (of labels) and for each $a \in A$, $\stackrel{a}{\longrightarrow}$ is a binary relation on S. We write $s \stackrel{a}{\longrightarrow} s'$ instead of $(s, s') \in \stackrel{a}{\longrightarrow}$. Sometimes there is extra structure in a transition system, a set of atomic colours Q, such that each colour $q \subseteq S$ (the subset of states with colour q).

Consider the transition systems pictured in Figure 1. Here u and w are simple vending machines, and p is a person who wishes to obtain tea. The language accepted by u, $\{\text{coin coffee}, \text{coin tea}\}$, is the same as that accepted by w. When p is placed in parallel with w, $p||w$, then deadlock is possible before a tea action because of the joint transition $p||w \stackrel{\text{coin}}{\longrightarrow} q||s$: we assume here that parallel composition requires both components to do the same action. In contrast $p||u \stackrel{\text{coin}}{\longrightarrow} q||v$, and tea is then the only next possible action. Therefore the language accepted by $p||w$, $\{\text{coin}, \text{coin tea}\}$, is different from the language accepted by $p||u$, $\{\text{coin tea}\}$. Consequently language equivalence is not a congruence for interacting "automata". Because of this Milner and others sought a more intensional notion of equivalence which would be preserved by communicating automata.

Bisimulations were introduced by Park [16] as a small refinement of the behavioural equivalence originally defined by Hennessy and Milner between basic CCS processes (whose behaviours are transition systems).

Definition 1 A binary relation R between states of a transition system is a *bisimulation* just in case whenever $(s, t) \in R$ and $a \in A$,

1. if $s \stackrel{a}{\longrightarrow} s'$ then $t \stackrel{a}{\longrightarrow} t'$ for some t' such that $(s', t') \in R$ and
2. if $t \stackrel{a}{\longrightarrow} t'$ then $s \stackrel{a}{\longrightarrow} s'$ for some s' such that $(s', t') \in R$.

In the case of an enriched transition system with colours there is an extra clause in the definition of a bisimulation that it preserves colours: if $(s, t) \in R$ then

$$0.\ \text{for all colours } q,\ \ s \in q \ \ \text{iff} \ \ t \in q$$

Simple examples of bisimulations are the identity relation and the empty relation. Two states of a transition system s and t are *bisimulation equivalent* (or *bisimilar*), written $s \sim t$, if there is a bisimulation relation R with $(s,t) \in R$. The machines u and w in Figure 1 are not bisimulation equivalent. The transition $u \stackrel{\text{coin}}{\longrightarrow} v$ cannot be matched either by $w \stackrel{\text{coin}}{\longrightarrow} s$ because s does not have a tea transition or by $w \stackrel{\text{coin}}{\longrightarrow} t$ because t does not have a coffee transition.

Transition systems are models for basic process calculi, such as ACP, CCS and CSP. Bisimulation equivalence is a congruence for all the operators of these calculi. By permitting more general operators, whose rules for transitions belong to a general format, bisimulation equivalence turns out to be the least congruence induced by language equivalence [9]. Models for richer process calculi capturing value passing, mobility, causality, time, probability and locations have been developed. The basic notion of bisimulation has been generalised, often in a variety of different ways, to cover these extra features. Bisimulation also has a nice categorical representation via co-algebras due to Aczel, see for example [18], which allows a very general definition. It is an interesting question whether all the different brands of bisimulation are instances of this categorical account.

It is common to identify a root of a transition system as a start state. Above we defined a bisimulation on states of the same transition graph. Equally we could have defined it between states of different transition systems. When transition systems are rooted we can then say that two systems are bisimilar if their roots are. A family Δ of rooted transition graphs is said to be *closed under bisimulation equivalence* when the following holds.

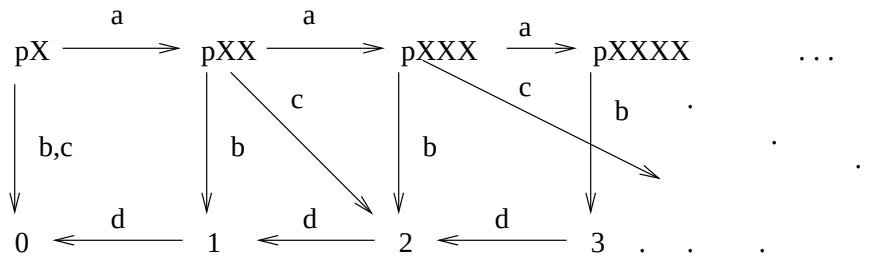
$$\text{if } G \in \Delta \text{ and } G \sim G' \text{ then } G' \in \Delta$$

Given a rooted transition system there is a "smallest" transition system which is bisimilar to it: this is its *canonical* transition graph which is the result of first removing any states which are not reachable from the root, and then identifying bisimilar states (using quotienting). For instance Figure 2 is the canonical graph of Figure 3.

An alternative perspective on bisimulation closure is from the viewpoint of properties of transition systems. Properties whose transition systems are bisimulation closed are said to be *bisimulation invariant*. Over rooted transition graphs property Φ is bisimulation invariant when the following holds.

$$\text{if } G \models \Phi \text{ and } G \sim G' \text{ then } G' \models \Phi$$

(By $G \models \Phi$ we mean that Φ is true of the transition graph G.) On the whole, "counting" properties are not bisimulation invariant, for example "has 32 states"



if $n = 2m$ then $n = \{rX^n, qX^m\}$
if $n = 2m - 1$ then $n = \{rX^n, sX^m\}$

Figure 2: Quotiented transition graph

or "has an even number of states". In contrast temporal properties are bisimulation invariant, for instance "will eventually do an a-transition" or "is never able to do a b-transition". Other properties such as "has an Hamiltonian circuit" or "is 3-colourable" are also not bisimulation invariant. Later we shall be interested in parameterised properties, that is properties of arbitrary arity. We say that an n-ary property $\Phi(x_1, \ldots, x_n)$ on transition systems is bisimulation invariant when the following is true.

if $G \models \Phi[s_1, \ldots, s_n]$ and $t_1, \ldots, t_n$ are states of G' and
$t_i \sim s_i$ for all $i : 1 \leq i \leq n$ then $G' \models \Phi[t_1, \ldots, t_n]$

(By $G \models \Phi[s_1, \ldots, s_n]$ we mean that $\Phi(x_1, \ldots, x_n)$ is true of G when x_i is interpreted as state s_i for each i.) An example of a property which is not bisimulation invariant is "$x_1, \ldots, x_n$ is a cycle", and an example of a bisimulation invariant property is "x_1 is language equivalent to x_2". The notions of bisimulation closure and invariance have appeared independently in a variety of contexts, see for instance [2, 3, 4, 5, 15].

Bisimulation was first introduced in the context of modal logic by Van Benthem [2] to give an account of which subfamily of first-order logic is definable in modal logic. Let M be the following modal logic where a ranges over A:

$$\Phi \ ::= \ \mathtt{tt} \ \mid \ \neg\, \Phi \ \mid \ \Phi_1 \vee \Phi_2 \ \mid \ \langle a \rangle \Phi$$

The inductive stipulation below defines when a state s of a transition graph G has a modal property Φ, written $s \models_G \Phi$, however we drop the index G.

$$\begin{array}{lll} s \models \mathtt{tt} & & \\ s \models \neg\, \Phi & \text{iff} & s \not\models \Phi \\ s \models \Phi \vee \Psi & \text{iff} & s \models \Phi \text{ or } s \models \Psi \\ s \models \langle a \rangle \Phi & \text{iff} & \exists t.\, s \xrightarrow{a} t \text{ and } t \models \Phi \end{array}$$

In the context of an enriched transition system one adds propositions q for each colour $q \in Q$ to the logic, with semantic clause: $s \models q$ iff $s \in q$. Modal formulas are bisimulation invariant: if $s \models \Phi$ and $s \sim t$ then $t \models \Phi$ for any modal Φ.

First-order logic, FOL, over transition systems contains binary relations E_a for each $a \in A$ (and monadic predicates $q(x)$ for each colour q if extended transition systems are under consideration). Formulas of FOL have the following form.

$$\Phi \ ::= \ xE_ay \ \mid \ x = y \ \mid \ \neg\,\Phi \ \mid \ \Phi_1 \wedge \Phi_2 \ \mid \ \forall x.\Phi$$

A formula $\Phi(x_1, \ldots, x_n)$ with at most the free variables $x_1, \ldots, x_n$ will be true or false of a transition system G and states $\widetilde{s} = s_1, \ldots, s_n$ in the usual way. For example

$$\begin{array}{lll} G \models x_iE_ax_j[\widetilde{s}] & \text{iff} & s_i \stackrel{a}{\longrightarrow} s_j \\ G \models \forall x_{n+1}.\Phi[\widetilde{s}] & \text{iff} & \forall s'.\ G \models \Phi[\widetilde{s}, s'] \end{array}$$

Not all first-order formulas are bisimulation invariant. An example is the formula $\exists y.\exists z.(x \stackrel{a}{\longrightarrow} y \wedge x \stackrel{a}{\longrightarrow} z \wedge y \neq z)$ which says "x has at least two different a-transitions".

Van Benthem introduced bisimulation to identify which formulas $\Phi(x)$ of FOL are equivalent to modal formulas (to M formulas) [3]. A formula $\Phi(x)$ is equivalent to a modal formula Φ' provided that for any G and for any state s, $G \models \Phi[s]$ iff $s \models_G \Phi'$. Van Benthem proved the following characterisation.

Proposition 1 *A* FOL *formula* $\Phi(x)$ *is equivalent to an* M *formula iff* $\Phi(x)$ *is bisimulation invariant.*

3 Caucal's hierarchy

Bisimulation equivalence is a very fine equivalence between states. An interesting line of enquiry is to re-consider classical results in automata and language theory, replacing language equivalence with bisimulation equivalence. These results concern definability, closure properties and decidability/undecidability.

Grammars can be viewed as generators of transition systems. Let Γ be a finite family of nonterminals and assume that A is a finite set (of terminals). A basic transition has the form $\alpha \stackrel{a}{\longrightarrow} \beta$ where α, $\beta \in \Gamma^*$ and $a \in A$. A state is then any member of Γ^*, and the transition relations on states are defined as the least relations containing the basic transitions and satisfying the following prefix rule.

$$\text{if } \alpha \stackrel{a}{\longrightarrow} \beta \text{ then } \alpha\delta \stackrel{a}{\longrightarrow} \beta\delta$$

Given a state α we can define its rooted transition system whose states are just the ones reachable from α. An example is a pushdown automaton over the alphabet $A = \{a, b, c, d\}$ whose basic transitions are as follows (where ϵ is the empty stack sequence).

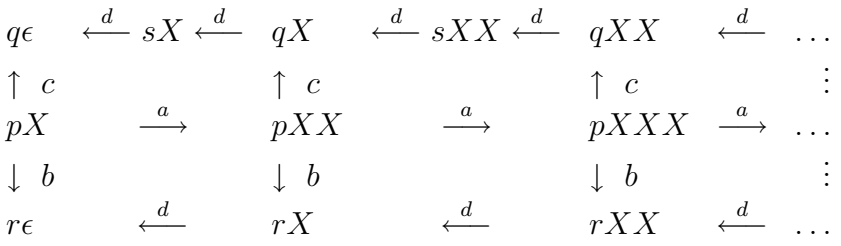


Figure 3: A pushdown automaton

$$\{pX \xrightarrow{a} pXX,\ pX \xrightarrow{c} q\epsilon,\ pX \xrightarrow{b} r\epsilon,\ qX \xrightarrow{d} sX,\ sX \xrightarrow{d} q\epsilon,\ rX \xrightarrow{d} r\epsilon\}$$

The transition graph generated by pX is pictured in Figure 3. For any $n \geq 0$ the transition $qXX^n \xrightarrow{d} sXX^n$ is derived from the basic transition $qX \xrightarrow{d} sX$ using the prefix rule when β is X^n. In this example the set of nonterminals is divided into two, states Q' and stack elements Γ. Each basic transition belongs to $(Q' \times \Gamma) \times A \times (Q' \times \Gamma^*)$.

In the table below is the "Caucal hierarchy" of transition graph descriptions, which depends on how the family of basic transitions is specified. In each case we assume a finite family of rules. Type 3 captures regular grammars (whose graphs are finite-state), Type 2 captures context-free grammars in Greibach normal form, and Type $1\frac{1}{2}$, in fact, captures pushdown automata. For Type 0 and below this means that in each case there are finitely many basic transitions. In the other two cases R_1 and R_2 are regular expressions over Γ. The idea is that each rule $R_1 \xrightarrow{a} \beta$ stands for the possibly infinite family of basic transitions $\{\alpha \xrightarrow{a} \beta : \alpha \in R_1\}$ and $R_1 \xrightarrow{a} R_2$ stands for the family $\{\alpha \xrightarrow{a} \beta : \alpha \in R_1 \text{ and } \beta \in R_2\}$. For instance a Type -1 rule of the form $X^*Y \xrightarrow{a} Y$ includes for each $n \geq 0$ the basic transition $X^nY \xrightarrow{a} Y$.

	Basic Transitions
Type -2	$R_1 \xrightarrow{a} R_2$
Type -1	$R_1 \xrightarrow{a} \beta$
Type 0	$\alpha \xrightarrow{a} \beta$
Type $1\frac{1}{2}$	$\alpha \xrightarrow{a} \beta$ where $\lvert\alpha\rvert = 2$ and $\lvert\beta\rvert > 0$
Type 2	$X \xrightarrow{a} \beta$
Type 3	$X \xrightarrow{a} Y$ or $X \xrightarrow{a} \epsilon$

This hierarchy is implicit in Caucal's work on understanding context-free graphs, and understanding when a graph has a decidable monadic second-order

theory [5, 4, 6]. With respect to language equivalence, the hierarchy collapses to just two levels, the regular and the context-free. The families between Type 2 and Type -2 are equivalent: for every G of Type -2 and root α there is a G' of Type 2 and root α' such that the language of α is the same as the language of α'.

The standard textbook transformation from pushdown automata to context-free grammars (Type $1\frac{1}{2}$ to Type 2) does not preserve bisimulation equivalence. In fact, with respect to bisimilarity pushdown automata are richer than context-free grammars. Caucal [5] shows that there is not a Type 2 transition graph and root α which is bisimulation equivalent to pX of Figure 3. He also shows that Type 0 transition systems coincide (up to isomorphism) with Type $1\frac{1}{2}$. There is a strict hierarchy between Type 0 and Type -2. Therefore, with respect to bisimulation equivalence there are five levels in the hierarchy. An interesting consideration is to what extent this hierarchy is closed under canonical transition graphs. Figure 2 is clearly not a Type 0 graph but it is the canonical graph for Figure 3, and therefore this shows that Type 0 is not closed under canonical graphs, see [4] for further details and results.

Baeten, Bergstra and Klop proved that bisimulation equivalence is decidable for a subset of Type 2 transition systems[1] [1]. The decidability result was generalised in [7] to encompass all Type 2 graphs. Groote and Hüttel proved that other standard equivalences (traces, failures, simulation, 2/3-bisimulation etc..,) on Type 2 graphs are all undecidable using reductions from the undecidability of language equivalence for these graphs [10]. The most recent result is by Sénizergues [20], who shows that bisimulation equivalence is decidable for transition systems somewhere between Type 0 and -1. This result is a small generalisation of his formidable proof of decidability of language equivalence for DPDA [19, 20]. Using ideas developed in concurrency theory (tableaux methods) we have simplified his proof of the DPDA result [22]. This leaves as an open question whether bisimulation equivalence is also decidable for Type -1 and Type -2 systems.

4 Richer logics

Modal logic M of section 2 is not very expressive. For instance it cannot express temporal properties, such as safety or liveness properties, of transition systems. Such properties have been found to be very useful when analysing the behaviour of concurrent systems. Modal mu-calculus, μM, introduced by Kozen [13], has the required extra expressive power. The new constructs over and above those of M are

$$\Phi \ ::= \ X \ \mid \ \ldots \ \mid \ \min X \Phi$$

[1] They proved it for the normed subfamily. G is normed if for every state s there is a word w such that $s \xrightarrow{w} \epsilon$.

where X ranges over a family of propositional variables, and in the case of $\min X\Phi$ there is a restriction that all free occurrences of X in Φ are within the scope of an even number of negations (to guarantee monotonicity).

The semantics of M is extended to encompass these extra constructs. The inductive definition of satisfaction stipulates when a state s of a transition system has the property $\Phi(X_1, \ldots, X_n)$ when each X_i is interpreted as the set of states S_i, written $s \models \Phi[\widetilde{S}]$, and the semantic clauses for the modal fragment are as before (except for the presence of the state sets).

$$\begin{array}{lll} s \models X_i[\widetilde{S}] & \text{iff} & s \in S_i \\ s \models \min X_{n+1}\Phi[\widetilde{S}] & \text{iff} & \forall S'.\, \text{if } (\forall t \in S'.\, t \models \Phi[\widetilde{S}, S']) \text{ then } s \in S' \end{array}$$

The stipulation for the fixed point follows directly from the Tarski-Knaster theorem, as a least fixed point is the intersection of all prefixed points. (Again we would add atomic formulas q if we are interested in extended transition systems.)

Second-order propositional modal logic, 2M, is defined as an extension of M as follows.

$$\Phi \;::=\; X \;\mid\; \ldots \;\mid\; \Box\,\Phi \;\mid\; \forall X.\Phi$$

The modality $\Box$ is the reflexive and transitive closure of $\bigcup\{[a] \,:\, a \in A\}$, and is included so that 2M includes μM. As with modal mu-calculus we define when $s \models \Phi[\widetilde{S}]$. The new clauses are:

$$\begin{array}{lll} s \models \Box\Phi[\widetilde{S}] & \text{iff} & \forall t.\, \forall w \in A^*.\, \text{if } s \xrightarrow{w} t \text{ then } t \models \Phi[\widetilde{S}] \\ s \models \forall X_{n+1}.\Phi[\widetilde{S}] & \text{iff} & \forall S'.\; s \models \Phi[\widetilde{S}, S'] \end{array}$$

There is a straightforward translation of μM into 2M. Let Tr be this translation. The important case is the fixed point: $\mathrm{Tr}(\min X\Phi) = \forall X.(\Box(\mathrm{Tr}(\Phi) \to X) \to X)$.

Formulas of M and closed formulas of μM are bisimulation invariant. This is not true in the case of 2M, for it is too rich for characterising bisimulation: for instance, a variety of "counting" properties are definable, such as "has at least two different a-transitions", expressible as $\exists X.(\langle a\rangle X \wedge \langle a\rangle\neg X)$. This means that two bisimilar states need not have the same 2M properties.

FOL over transition graphs is also not rich enough for capturing interesting properties. One extension of first-order logic is monadic second-order logic, 2OL, with the extra formulas

$$\Phi \;::=\; X(x) \;\mid\; \ldots \;\mid\; \forall X.\Phi$$

The semantic clauses are generalised as follows:

$$\begin{array}{lll} G \models X_i(x_j)[\widetilde{s}, \widetilde{S}] & \text{iff} & s_j \in S_i \\ G \models \forall X_{n+1}.\Phi[\widetilde{s}, \widetilde{S}] & \text{iff} & \forall S'.\, G \models \Phi[\widetilde{s}, \widetilde{S}, S'] \end{array}$$

Formulas of 2OL need not be bisimulation invariant. An interesting question is the relationship between μM and 2OL. Van Benthem's result was generalised by Janin and Walukiewicz [12] as follows.

Proposition 2 *A* 2OL *formula* $\Phi(x)$ *is equivalent to a closed* μM *formula iff* $\Phi(x)$ *is bisimulation invariant.*

One corollary of this result is that the bisimulation invariant closed formulas of 2M has the same expressive power as the closed formulas of μM.

A different extension of FOL is first-order logic with fixed points, μFOL, where there is the following extra formulas

$$\Phi \ ::= \ X(x_1, \ldots, x_k) \ \mid \ \ldots \ \mid \ (\mu X(x_1, \ldots, x_k).\, \Phi)(y_1, \ldots, y_k)$$

In the case of $(\mu X(\ldots).\, \Phi)(\ldots)$, there is the same restriction as in μM that all free occurrences of X in Φ lie within the scope of an even number of negations. The interpretation of a predicate X_i with arity k is a set of k-tuples, a subset of S^k. The semantic clauses are therefore as follows (where we use the notation S_i for sets of tuples).

The new semantic clauses are (where $\widetilde{s'}$ is $s_{k+1}, \ldots, s_n$)

$$\begin{array}{lll} G \models X_j(x_{i1}, \ldots, x_{ik})[\widetilde{s}, \widetilde{S}] & \text{iff} & (s_{i1}, \ldots, s_{ik}) \in S_j \\ G \models (\mu X_{n+1}(\ldots).\Phi)(x_1, \ldots, x_k)[\widetilde{s}, \widetilde{S}] & \text{iff} & \forall S'.\, \text{if } (\forall (t_1, \ldots, t_k) \in S'. \\ & & G \models \Phi[\widetilde{t}, \widetilde{s'}, \widetilde{S}, S']) \text{ then } (s_1, \ldots, s_k) \in S' \end{array}$$

When the alphabet A is finite, bisimulation equivalence is definable in μFOL as a dyadic greatest fixed point formula

$$(\nu Z(x, y).(Z(y, x) \wedge \bigwedge_{a \in \mathcal{A}} (\forall x'.\exists y'.xE_ax' \rightarrow yE_ay' \wedge Z(x', y'))))(\ldots)$$

where $(\nu Z(\ldots)\Phi)\,(\ldots)$ is $\neg\,((\mu Z(\ldots)\Phi(\neg Z))(\ldots))$. An interesting open question is how to characterise the bisimulation invariant sublogic of μFOL.

5 Finite model theory

Finite model theory is concerned with relationships between complexity classes and logics over finite structures. It is interesting to consider bisimulation invariance in the context of finite model theory. Rosen showed that Proposition 1 (in section 2) remains true with the restriction to finite transition systems [17]. It is an open question whether Proposition 2 also remains true under this restriction.

Part of the interest in relationships between μM and 2M or 2OL with respect to finite transition systems is that within 2M and 2OL one can define NP-complete problems: examples include 3-colourability on finite connected undirected graphs.

Consider such a graph. If there is an edge between two states s and t let $s \stackrel{a}{\longrightarrow} t$ and $t \stackrel{a}{\longrightarrow} s$. So in this case $A = \{a\}$, and 3-colourability is given by:

$$\exists X.\, \exists Y.\, \exists Z.\, (\Phi \wedge \Box((X \rightarrow [a]\neg X) \wedge (Y \rightarrow [a]\neg Y) \wedge (Z \rightarrow [a]\neg Z)))$$

where Φ, which says that every vertex has a unique colour, is

$$\Box((X \wedge \neg Y \wedge \neg Z) \vee (Y \wedge \neg Z \wedge \neg X) \vee (Z \wedge \neg X \wedge \neg Y))$$

In contrast, μM formulas over finite transition systems can only express PTIME properties.

An interesting open question is whether there is a logic which captures exactly the PTIME properties of transition systems. Otto has shown that there is a logic for the PTIME properties that are bisimulation invariant [15]. The right setting is μFOL over canonical transition systems (where $=$ is $\sim$, and a linear ordering on states is thereby definable).

We now consider emaciated finite transition systems whose set A is a singleton. That is now $G = (S, \longrightarrow)$ where S is finite. We write $s \stackrel{n}{\longrightarrow} t$, $n \geq 0$, if there is a sequence of transitions of length n from s to t (and by convention $s \stackrel{0}{\longrightarrow} s$). A state is terminal if it has no transitions. The language of state s is therefore the set of words $L(s) = \{i \geq 0 : s \stackrel{i}{\longrightarrow} t \text{ and } t \text{ is terminal}\}$. Consequently, s and s' are language equivalent if $L(s) = L(s')$. The property "x is language equivalent to y" as was noted earlier is bisimulation invariant. The definition in μFOL of bisimulation equivalence of the previous section remains correct when transition systems are finite. However it is unlikely that language equivalence is definable in μFOL because of the following result, proved in [23].

Proposition 3 *Language equivalence on (canonical) emaciated finite transition graphs is co-NP complete.*

Hence language equivalence over finite transition systems is definable in μFOL iff PTIME = NP. Dawar offers a different route to this observation [8].

A classical result (due to Immermann, Gurevich and Shelah) in a slightly normalised form is:

Proposition 4 *A μFOL formula $\Psi(y_1, \ldots, y_n)$ over finite transition systems is equivalent to a formula of the form $\exists u.\, ((\mu Z(x_1, \ldots, x_m).\, \Phi)(y_1, \ldots, y_n, u, \ldots, u))$ where Φ is first-order and contains at most $x_1, \ldots, x_m$ free.*

The argument places in the application $(\ldots)$ from $n+1$ to m are all filled by the same element u. This allows for the arity of the defining fixed point m to be larger than the arity of the μFOL formula n. Consequently, if one can prove that "y is language equivalent to z" is not definable by a μFOL formula in normal form, $\exists u.\, (\mu Z(x_1, \ldots, x_m).\, \Phi\,(y, z, u, \ldots, u))$, then this would show that PTIME is different from NP.

The result below has the consequence that language equivalence is not definable by a normal formula of the form $\exists u.\,(\mu Z(x_1, x_2, x_3).\,\Phi\,(y, z, u))$. We present the theorem in the most general form possible, that language equivalence is not definable as an unconditional projection of a simple fixed point.

Theorem 1 *Language equivalence is not definable by a normal formula of the form* $(\mu Z(x_1, \ldots, x_n).\,\Phi)(\ldots)$ *(over finite transition systems).*

That is, language equivalence is not definable as an unconditional projection of a simple fixed point. The rest of the paper is devoted to its proof. One popular method for showing non-definability is to use games. Here we use a variant method which introduces "proofs" of formulas. A sufficiently concrete account of when a formula is true of a transition system is given by a tableau proof. The aim is to provide a proof system $G \vdash \Psi(s_1, \ldots, s_n)$ for showing $G \models \Psi[s_1, \ldots, s_n]$ when Ψ is a formula of the form $(\mu Z(x_1, \ldots, x_n).\,\Phi)(\ldots)$ containing the single fixed point $\mu Z(\ldots)$: it is straightforward to extend the proof system to formulas with multiple fixed points. The property checker is a tableau system, a goal directed proof system. Assume that the starting formula is $(\mu Z(x_1, \ldots, x_n).\,\Phi)(\widetilde{x})$ and let $\{s_1/x_1, \ldots, s_n/x_n\}$ be the simultaneous substitution of each s_i for x_i. We assume that in Φ all negations are moved inwards in the usual way. The tableau rules are therefore as follows.

$$\frac{G \vdash (\mu Z(x_1, \ldots, x_n).\,\Phi)(\widetilde{x})(s_1, \ldots, s_n)}{G \vdash Z(s_1, \ldots, s_n)}$$

$$\frac{G \vdash Z(s_1, \ldots, s_n)}{G \vdash \Phi\{s_1/x_1, \ldots, s_n/x_n\}}$$

$$\frac{G \vdash \exists x.\Psi}{G \vdash \Psi\{s/x\}}\ s \in S$$

$$\frac{G \vdash \forall x.\Psi}{G \vdash \Psi\{s_1/x\} \quad \ldots \quad G \vdash \Psi\{s_k/x\}}\ S = \{s_1, \ldots, s_k\}$$

$$\frac{G \vdash \Psi_1 \wedge \Psi_2}{G \vdash \Psi_1 \quad G \vdash \Psi_2}$$

$$\frac{G \vdash \Psi_1 \vee \Psi_2}{G \vdash \Psi_1} \qquad \frac{G \vdash \Psi_1 \vee \Psi_2}{G \vdash \Psi_2}$$

To test if $G \models (\mu Z(x_1, \ldots, x_n).\,\Phi)(\widetilde{x})[s_1, \ldots, s_n]$ one tries to develop a proof of $G \vdash (\mu Z(x_1, \ldots, x_n).\,\Phi)(\widetilde{x})(s_1, \ldots, s_n)$ by building a tableau, a finite proof tree whose root is labelled with this initial sequent. The sequents labelling the immediate successors of a node are determined by an application of one of the rules. One keeps building a proof until we reach a terminal sequent.

A terminal sequent has one of the following forms

1. $G \vdash s = t$ or $G \vdash s \neq t$

Assume $0 < t < k < k+t < a$

$$\begin{array}{llll} e_0 \longrightarrow \ldots \longrightarrow e_k \longrightarrow \ldots \longrightarrow e_{k+t} \longrightarrow \ldots \longrightarrow e_a & \text{and} & e_0 \longrightarrow e_t \\ l_0 \longrightarrow \ldots \longrightarrow l_k \longrightarrow \ldots \longrightarrow l_{k+t} \longrightarrow \ldots \longrightarrow l_a & \text{and} & e_k \longrightarrow e_{k+t} \\ l'_1 \longrightarrow \ldots \longrightarrow l'_k \longrightarrow \ldots \longrightarrow l'_{k+t} \longrightarrow \ldots \longrightarrow l'_a & \text{and} & l'_k \longrightarrow l'_{k+t} \end{array}$$

Figure 4: Ingredients of the graphs G and G'

2. $G \vdash sEt$ or $G \vdash \neg(sEt)$

3. $G \vdash Z(s_1, \ldots, s_n)$ and in the proof tree above this sequent there is the same sequent $G \vdash Z(s_1, \ldots, s_n)$.

Terminal sequents of type 1 or 2 which are true are successful. A tableau proof is successful if all of its leaves are successful, as shown by the next result whose proof is straightforward.

Lemma 1 *$G \models (\mu Z(x_1, \ldots, x_n).\, \Phi)(\widetilde{x})[s_1, \ldots, s_n]$ iff there is a successful tableau whose root is $G \vdash (\mu Z(x_1, \ldots, x_n).\, \Phi)(\widetilde{x})(s_1, \ldots, s_n)$.*

Proof of Theorem 1: Suppose $(\mu Z(x_1, \ldots, x_n).\, \Phi)(\widetilde{x})$ defines language equivalence. That is for any transition graph G and states $s_1, \ldots, s_n$ of S

$$G \models (\mu Z(x_1, \ldots, x_n).\, \Phi)(\widetilde{x})[s_1, \ldots, s_n] \;\text{ iff }\; L(s_1) = L(s_2)$$

We assume that Φ is in prenex normal form $Q_1 x_{i1} \ldots Q_m x_{im} \Psi$ where Ψ is in DNF: each clause in Ψ contains atomic formulas of the form $x = y$, $x \neq y$, xEy, $\neg(xEy)$ and $Z(y_1, \ldots, y_n)$ where the variables x, y range over the set $X = \{x_1, \ldots, x_n, x_{i1}, \ldots, x_{im}\}$ (where x_{ij} could be the same as x_k).

Consider the transition systems in Figure 4. The e vertices have an "early" branching point whereas the l and l' vertices have a "late" branching point. Let G be the graph whose vertices are e_i and l_i, and let G' be the similar graph whose vertices are e_i and l'_i. Notice that $L(e_0) = L(l_0)$ but $L(e_i) \neq L(l_i)$ when $i : 0 < i < k+1$. Moreover $L(e_0) \neq L(l'_1)$. We assume that $k > n2^{m+1}$ where m is the number of quantifiers and n is the arity of Z, and we assume that $t < k$ and $t > 1$.

There is a tableau proof of $G \vdash (\mu Z(x_1, \ldots, x_n).\, \Phi)(\widetilde{x})(e_0, l_0, \widetilde{v})$ by Lemma 1, where $\widetilde{v}$ is any sequence of vertices $v_3, \ldots, v_n$. Consider any $\widetilde{v}$ such that there is a shortest depth tableau proof of $G \vdash Z(e_0, l_0, \widetilde{v})$. The argument proceeds by showing that there is also a proof of $G' \vdash Z(e_0, l'_1, \widetilde{v}')$. First we define the elements $\widetilde{v}'$.

Assume $\widetilde{v} = v_3 \ldots v_n$. We define $\widetilde{v}' = v'_3 \ldots v'_n$ as follows. If v_i is e_j then $v'_i = e_j$ and if $v_i = l_{k+j}$ then $v'_i = l'_{k+j}$. Otherwise $v_i = l_j$ and $j < k$. Consider all

such v_i in decreasing order, say $l_{j1} \ldots l_{jb}$ (where $b < (n-1)$). If $j1 \geq (k - 2^{m+1})$ then the corresponding element is l'_{j1} otherwise it is l'_{j1+1}: and in the second case all the other corresponding elements are l'_{js+1} for $1 < s \leq b$. Assume then that for the first element l_{j1} the index $j1 \geq (k - 2^{m+1})$. Consider now the second element l_{j2}. If the index $j2 \geq (j1 - 2^{m+1})$ then the corresponding element is l'_{j2} otherwise it is l'_{j2+1} and again in this second case the rest of the corresponding elements are l'_{js+1} for $2 < s \leq b$. Repeat this construction as long as $js \geq j(s-1) - 2^{m+1}$ where the corresponding element is l'_{js}, and otherwise it is l'_{js+1} and the rest of the corresponding elements are l'_{jz+1} for $s < z \leq b$.

Consider the sequence of elements $l_k\, l_{j1} \ldots l_{jb}\, l_0$ in G and the corresponding sequence $l'_k\, l'_{j1'} \ldots l'_{jb'}\, l'_1$ in G'. The index ji' in the second sequence is either ji or $ji+1$. Let p be the pivot point in the sequence where all indices to the left also occur in the first sequence and all indices to the right including that of p are 1 plus the index of the corresponding element in the first sequence. Note that the difference in the index between element p and the next element to the left is greater than 2^{m+1}.

We now show how a proof of $G \vdash Z(e_0, l_0, \widetilde{v})$ can be used to develop a proof of $G' \vdash Z(e_0, l'_1, \widetilde{v'})$. The goal $G \vdash Z(e_0, l_0, \widetilde{v})$ reduces to the subgoal $G \vdash (Q_1 x_{i1} \ldots Q_m x_{im} \Psi)(e_0, l_0, \widetilde{v})$. Similarly the goal $G' \vdash Z(e_0, l'_1, \widetilde{v'})$ reduces to the subgoal $G' \vdash (Q_1 x_{i1} \ldots Q_m x_{im} \Psi)(e_0, l'_1, \widetilde{v'})$.

We consider the quantifiers in turn, and at each stage the pivot point may become updated.

Suppose $Q_1 x_{i1} = \forall x_{i1}$. The goal $G \vdash (Q_1 x_{i1} \ldots Q_m x_{im} \Psi)(e_0, l_0, \widetilde{v})$ reduces to subgoals one for each $u \in G$, $G \vdash ((Q_2 x_{i2} \ldots Q_m x_{im} \Psi)(e_0, l_0, \widetilde{v}))\{u/x_{i1}\}$. For each such subgoal we associate a subgoal of the second proof which has the following form $G' \vdash ((Q_2 x_{i2} \ldots Q_m x_{im} \Psi)(e_0, l'_1, \widetilde{v'}))\{u'/x_{i1}\}$, so that all $u' \in G'$ are dealt with. If $u = e_i$ then $u' = e_i$. Let j be the index of the pivot element p. If $u = l_i$ and $i > j + 2^m$ then $u' = l'_i$ otherwise $u' = l'_{i+1}$. In the circumstance that $i \geq j$ but $j + 2^m \geq i$ then the pivot element p is updated to that of u'.

The argument is similar when $Q_1 x_{i1} = \exists x_{i1}$. Now there is only one subgoal $G \vdash ((Q_2 x_{i2} \ldots Q_m x_{im} \Psi)(e_0, l_0, \widetilde{v}))\{u/x_{i1}\}$. The corresponding subgoal $G' \vdash ((Q_2 x_{i2} \ldots Q_m x_{im} \Psi)(e_0, l'_1, \widetilde{v'}))\{u'/x_{i1}\}$ is chosen as above, and again the pivot element may be updated.

The argument continues for the remaining quantifiers. Suppose the goal is $G \vdash ((Q_c x_{ic} \ldots Q_m x_{im} \Psi)(e_0, l_0, \widetilde{v}))\{u_1/x_{i1}, \ldots, u_{c-1}/x_{i(c-1)}\}$, and the corresponding goal is $G' \vdash ((Q_c x_{ic} \ldots Q_m x_{im} \Psi)(e_0, l'_1, \widetilde{v'}))\{u'_1/x_{i1}, \ldots, u'_{c-1}/x_{i(c-1)}\}$ in the second proof. If $Q_c x_{ic}$ is $\forall x_{ic}$ then we proceed as above except if j is the index of the current pivot element p and $u_c = l_i$ and $i > j + 2^{(m+1)-c}$ then $u'_c = l'_i$ otherwise $u' = l'_{i+1}$. The pivot element is updated to u_c when $i \geq j$ but $j + 2^{(m+1)-c} \geq i$.

As quantifiers are eliminated the sequence of elements $l_k\, l_{j1} \ldots l_{jb}\, l_0$ in G and the corresponding sequence $l'_k\, l'_{j1'} \ldots l'_{jb'}\, l'_1$ in G' may be expanded, and the pivot element in the second sequence updated. However at each stage, after eliminating quantifier $Q_c x_{ic}$, the difference in the index between the pivot element p and the

next element to the left is greater than $2^{(m+1)-c}$.

Finally all the quantifiers are removed, and a subgoal of the first proof has the form $G \vdash \Psi(e_0, l_0, \widetilde{v}, \widetilde{u})$ and in the second proof has the form $G' \vdash \Psi(e_0, l'_1, \widetilde{v}', \widetilde{u}')$. The formula Ψ is in DNF. Assume that $\Psi = \bigvee \Psi_i$ where each Ψ_i is a conjunction of atomic formulas. Suppose the subgoal of $G \vdash \Psi(e_0, l_0, \widetilde{v}, \widetilde{u})$ is $G \vdash \Psi_i(e_0, l_0, \widetilde{v}, \widetilde{u})$, then the corresponding subgoal is $G' \vdash \Psi_i(e_0, l'_1, \widetilde{v}', \widetilde{u}')$. Consider any atomic formula B in Ψ_i. If B has the form $x = y$ or xEy then because of the construction of $\widetilde{v}'$ and $\widetilde{u}'$ it follows that $G \models B[e_0, l_0, \widetilde{v}, \widetilde{u}]$ iff $G' \models B[e_0, l'_1, \widetilde{v}', \widetilde{u}']$. The only other possible atomic sentences have the form $Z(y_1, \ldots, y_n)$. Suppose a subgoal of $G \vdash \Psi_i(e_0, l_0, \widetilde{v}, \widetilde{u})$ is $G \vdash Z(w_1, \ldots w_n)$. It follows that w_1 and w_2 are not e_0 and l_0 (for otherwise the proof of $G \vdash Z(e_0, l_0, \widetilde{w})$ must be shorter than that of $G \vdash Z(e_0, l_0, \widetilde{v})$ contrary to assumption). Hence either $w_1 = w_2$ or $w_1 = e_j$ and $w_2 = l_j$ where $j > k$. But then there must also be a successful proof for $G' \vdash Z(w'_1, \ldots, w'_n)$ as $L(w'_1) = L(w'_2)$. □

Acknowledgement: I would like to thank the referee for comments and wording of the main result of this paper.

References

[1] Baeten, J., Bergstra, J., and Klop, J. (1993). Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of Association of Computing Machinery*, **40**, 653-682.

[2] van Benthem, J. (1984). Correspondence theory. In *Handbook of Philosophical Logic*, Vol. II, ed. Gabbay, D. and Guenthner, F., 167-248, Reidel.

[3] van Benthem, J. (1996). Exploring Logical Dynamics. *CSLI Publications*.

[4] Burkart, O., Caucal, D., and Steffen, B. (1996). Bisimulation collapse and the process taxonomy. *Lecture Notes in Computer Science*, **1119**, 247-262.

[5] Caucal, D. (1992). On the regular structure of prefix rewriting. *Theoretical Computer Science*, **106**, 61-86.

[6] Caucal, D. (1996). On infinite transition graphs having a decidable monadic theory. *Lecture Notes in Computer Science*, **1099**, 194-205.

[7] Christensen, S., Hüttel, H., and Stirling, C. (1995). Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, **121**, 143-148.

[8] Dawar, A. (1998). A restricted second-order logic for finite structures, *Information and Computation*, **143**.

[9] Groote, J. (1993). Transition system specifications with negative premises. *Theoretical Computer Science*, 118, 263-299.

[10] Groote, J., and Hüttel, H. (1994). Undecidable equivalences for basic process algebra. *Information and Computation*, **115**, 354-371.

[11] Hennessy, M. and Milner, R. (1985). Algebraic laws for nondeterminism and concurrency. *Journal of Association of Computer Machinery*, **32**, 137-162.

[12] Janin, D. and Walukiewicz, I (1996). On the expressive completeness of the propositional mu-calculus with respect to the monadic second order logic. *Lecture Notes in Computer Science*, **1119**, 263-277.

[13] Kozen, D. (1983). Results on the propositional mu-calculus. *Theoretical Computer Science*, **27**, 333-354.

[14] Milner, R. (1989). *Communication and Concurrency*. Prentice Hall.

[15] M. Otto. Bisimulation-invariant ptime and higher-dimensional μ-calculus. *Theoretical Computer Science*, **224**, 73–113, 1999.

[16] Park, D. (1981). Concurrency and automata on infinite sequences. *Lecture Notes in Computer Science*, **154**, 561-572.

[17] Rosen, E. (1997). Modal logic over finite structures. *Journal of Logic, Language and Information*, **6**, 427-439.

[18] Rutten, J. (1995). A calculus of transition systems (towards universal coalgebra). In *Modal Logic and Process Algebra*, ed. Ponse, A., De Rijke, M. and Venema, Y. *CSLI Publications*, 187-216.

[19] Sénizergues, G. (1997). The equivalence problem for deterministic pushdown automata is decidable. *Lecture Notes in Computer Science*, **1256**, 671-681.

[20] G. Sénizergues. L(A) = L(B)? decidability results from complete formal systems. *Theoretical Computer Science* **251**, 1–166, 2001

[21] Sénizergues, G. (1998). Decidability of bisimulation equivalence for equational graphs of finite out-degree. *Procs IEEE FOCS 98*, 120-129.

[22] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, **255**, 1–31, 2001.

[23] Stockmeyer, L. and Meyer, A. (1973). Word problems requiring exponential time. *Procs. 5th ACM STOC*, 1-9.