Seminar Paper

A Complete Axiomatization for Observational Congruence of Finite-state Behaviors

by **Felix Klein**

Concurrency Theory Seminar

Prof. Dr. Holger Hermanns,
Dr. Martin Neuhäußer

Saarland University
Winter Term 2011/12

This seminar paper corresponds to the paper "A Complete Axiomatization for Observational Congruence of Finite-state Behaviors" of Robin Milner, which was published in the journal *Information and Computation, Volume 81 Issue 2* in May 1989. The paper provides a complete axiomatization for observational congruence, which is proven to be both: sound and complete. In this paper, we will explicitly analyze the preliminaries, the basic definitions, the notations and the proof structure in more detail. For the proof structure, we additionally consider the basic ideas and intuitions of the separate steps. Regarding the age of the paper, we will also have a short look in what extend the paper influenced todays research in this area.

1 Introduction

Nowadays, first modeling a system in a simplified manner instead of directly building it is a common practice. The reasons therefore are: easier and cheaper testability, more modularity and the possibility to abstract from unnecessary details. If the abstraction level is high, then the observational behavior of the system is mostly completely sufficient to describe all necessary components and abstract from the implementation details. Finite-state behaviors can then be used to model such systems. For example, imagine you are an alarm clock manufacturer. Then normally, your clients will come to you with a description of the design and the behavior of a clock. They are not interested in how you will build the clock, they only want that the final clock you produce for them behaves like they have specified it. Then, with a finite-state behavior model, you can represent the given specification in an exact way and formally analyze the model for each property you need. We will now see how to specify such a model. Therefore, assume you get the following alarm clock behavior specification from one of your clients:

- Every second, the alarm clock makes a tick and the clock hands move.
- If the wake up time is reached, the alarm clock rings.
- The alarm clock measures time to work correctly.

Each of these descriptions specify an action the alarm clock is able to do. We abbreviatory call this actions: *tick*, *ring* and *measure*. A *specific behavior* of an alarm clock is then an arbitrary or specific composition of these actions, like for example: *tick*, *tick*, *ring*, *tick*, *ring*, *measure*, and so on. This composition can either stop after some time or continue forever. Then we call a set of such compositions, which describes each possible execution, a *behavior*. Finally, a *finite-state behavior* is a behavior that can be described with a finite representation. Consider, that there are behaviors that are no finite-state behaviors. There are several ways to describe finite-state behaviors, one is the class of behavior expressions $\mathscr{E}$, which is defined recursively by the rules given in Figure 1.

$$\boxed{\begin{array}{rccc} & \multicolumn{3}{c}{0 \in \mathscr{E} \wedge Var \subseteq \mathscr{E}} \\ \text{action:} & E \in \mathscr{E} \wedge a \in Act & \Rightarrow & a.E \in \mathscr{E} \\ \text{summation:} & E \in \mathscr{E} \wedge E' \in \mathscr{E} & \Rightarrow & E + E' \in \mathscr{E} \\ \text{recursion:} & E \in \mathscr{E} \wedge X \in Var & \Rightarrow & \mu X E \in \mathscr{E} \end{array}}$$

Figure 1: Recursive definition of the behavior expressions $\mathscr{E}$, where Var is the set of variables ($0 \notin Var$) and Act is the set of possible actions

Before we specify the semantics for these expressions, we introduce some further definitions. We call a variable $X \in Var$ *bounded*, if it occurs as a subexpression of $\mu X E$, otherwise we call the variable *free*. Furthermore, we use the notation $E\{F_1, ..., F_n / X_1, ..., X_n\}$ to denote, that the variables X_1 to X_n occurring in the behavior expression E are replaced by the behavior expressions F_1 to F_n simultaneously. Now, we can specify the semantics of the behavior expressions, which is given by the smallest transition relation $\to \subseteq \mathscr{E} \times Act \times \mathscr{E}$, where the elements $E \xrightarrow{a} E'$ [1] of the transition relation are also defined recursively by the rules shown in Figure 2.

$$\boxed{\begin{array}{ccc} \multicolumn{3}{c}{\forall a \in Act, E \in \mathscr{E}.\ \ aE \xrightarrow{a} E} \\ E_1 \xrightarrow{a} E \vee E_2 \xrightarrow{a} E & \Rightarrow & E_1 + E_2 \xrightarrow{a} E \\ E\{\mu X E / X\} \xrightarrow{a} E' & \Rightarrow & \mu X E \xrightarrow{a} E' \end{array}}$$

Figure 2: Recursive definition of the transition relation

[1]alternative notation for $(E, a, E') \in \to$

The intuition behind this definition is that we can now represent a behavior expression also as a graph, where the initial state is labeled with the expression itself and the other states are defined by the remaining behavior expressions that are reachable through the transition relation. We also call such graphs *transition diagrams* or *charts*. To get more comfortable with the new two representations, Figure 3 shows six different behavior expressions with their corresponding transition diagrams for six different alarm clock models.

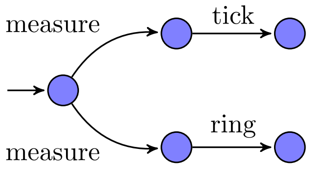
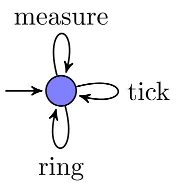
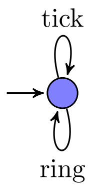
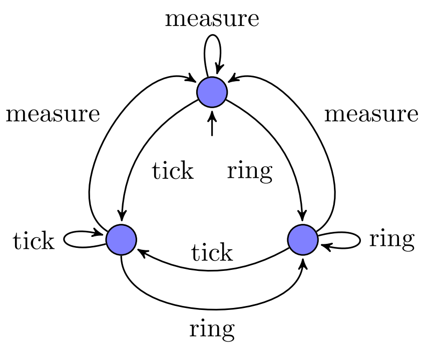
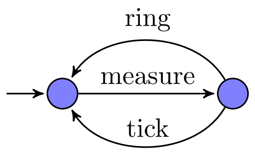
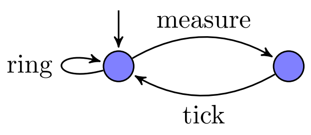
clock no.	**behavior expression**	**transition diagram**
1	$measure.\mu X(tick.Y + X) + measure.ring.0$	
2	$\mu X(measure.X + ring.X + tick.X)$	
3	$\mu X(ring.X + tick.X)$	
4	$\mu X \mu Y \mu Z(measure.X + ring.Y + tick.Z)$	 measure measure measure tick ring tick tick ring ring
5	$\mu X(measure.(ring.X + tick.X))$	
6	$\mu X(measure.tick.X + ring.X)$	

Figure 3: Some alarm clock models

By analyzing these models, it should be straightforward to see in which way these expressions describe a finite-state behavior. We say a specific behavior is specified by a finite-state expression when there exists a path on the corresponding transition diagram such that the composition of the edge labels on this path is equal to this specific behavior. Since we only cover the labels of the edges and not the labels of the states, we also call such compositions *traces* and a behavior is then a set of traces.

The first clock of our example then describes a behavior with exactly two finite traces: $measure, tick$ and $measure, ring$. But this model also shows us some other interesting derivations. From a first observation, we get that the corresponding transition diagram has two different final states[2], which correspond to the expressions 0 and Y. Generally, there are only two types of expressions which cannot have any successors. One type uses the single expression 0, which we also call the inaction, the second type uses free variables. The difference to using 0 is that whenever free variables occur as subexpressions of another expression, they could be bound again and have successors, where 0 never has a successor independently of the expression in which it occurs. Furthermore consider that if multiple 0s occur in a summation, like in the expression $0 + 0 + 0$, and these are the only summands, then the expression also represents a final state that is different to 0 and has no successors. Accordingly, we can always create arbitrary many final states, using these summations. A second observation corresponds to the μ-operator or recursion. Normally, recursion is used to introduce loops in the transition diagram, where μX indicates the start of the loop and the single occurrence of X inside the subexpression of $\mu X E$ indicates the end. The actions, that occur between μX and X, are the labels of the loop in the graph. If we have multiple loops, having the same start- and end-expression, we can use the same variable to express them, like for clock 2, otherwise we need multiple variables[3]. The difference to clock 1 is, that in clock 1, there are no actions between the start and the end point at all. Accordingly, we are also not able to introduce edges in the transition graph and therefore we have no loops in the transition diagram. For this special case, we call the variable X *unguarded*, where otherwise we call the variable *guarded*. Unguarded variables still play an important role later. Finally, we can observe that transition diagrams are in general a weaker representation than behavior expressions. For example, the expression $\mu X(measure.X + tick.X + ring.X)$ has the same transition diagram as the expression of clock 2. On the one hand, transition diagrams hide the order of the statements in an expression, on the other hand unguarded recursions vanish.

2 Bisimulations

As we have seen, sometimes we have different representations, which cover the same finite-state behavior. For example, we can have different behavior expressions, that have the same transition system for clock 2. But also comparing clock 4 with clock 2 gains to the result that somehow they are the same, because they have both the same set of traces. This analysis results in a major question in this theory: whether two behavior expressions are equal.

As already said, the main idea here is to say two behavior expressions are equal, if they induce the same set of traces. Using the definition of a *bisimulation* guarantees us this property:

Definition. A relation $\mathcal{R} \subseteq \mathscr{E} \times \mathscr{E}$ is a <u>bisimulation</u> if, whenever $(E, F) \in \mathcal{R}$, $a \in Act$

(i) $E \xrightarrow{a} E' \;\Rightarrow\; \exists\, F' \in \mathscr{E}.\ F \xrightarrow{a} F' \;\wedge\; (E', F') \in \mathcal{R}$
(ii) $F \xrightarrow{a} F' \;\Rightarrow\; \exists\, E' \in \mathscr{E}.\ E \xrightarrow{a} E' \;\wedge\; (E', F') \in \mathcal{R}$
(iii) $\forall\, X \in Var.\ E \vartriangleright X \Leftrightarrow F \vartriangleright X$

Where we use $E \vartriangleright X$ to say X is a free unguarded occurrence in the behavior expression E. The idea behind this definition is that we can denote two behavior expressions as equal by defining: if one expression is able to do an action, then also the other expression is able to do the same action and vice versa. Recursively, the same argument must hold for the

[2]states without a successor
[3]We can also use the same variable, if a recursion occurs as a subexpression of another recursion and there is no recursion variable in the subexpression, which corresponds to the outer recursion. But to improve readability, we try to avoid such expressions.

resulting subexpressions. We use the bisimulation relation here, to track all possible occurring subexpressions and to ensure that they are equal after definition too. Accordingly, we can now say that two behavior expressions E and F have the same set of traces if there exists a bisimulation $\mathcal{R}$ such that $(E, F) \in \mathcal{R}$ and we denote this by writing $E \sim F$.

Using this definition, the existence of the bisimulation

$$\begin{aligned}
\{(&\mu X(measure.X + ring.X + tick.X), \mu Z(measure.X + ring.Y + tick.Z)),\\
(&\mu X(measure.X + ring.X + tick.X), \mu Y \mu Z(measure.X + ring.Y + tick.Z)),\\
(&\mu X(measure.X + ring.X + tick.X), \mu X \mu Y \mu Z(measure.X + ring.Y + tick.Z))\}
\end{aligned}$$

indeed proves that clock 2 and clock 4 have the same observational behavior. For all other combinations of our examples, there does not exist such a bisimulation; they are different. The reason therefore is that the clocks 5 and 6 enforce a specific order on the execution, which is in both cases different, and clock 3 cannot do the *measure* action.

If we now jump back to your role as an alarm clock manufacturer, you could imagine, that identifying two expressions to be equal would gain you a lot. You can now check whether two clients specify the same clock on two different ways and simply build the same for them. But additionally, the question arises, is this all we can do? The problem, somebody normally would mention here, is that using this definition the clocks 2 and 3 are different, but if we look at the final produced clock afterwards, we normally cannot observe any difference at all. This leads to the fact that we can "see" only two actions instead of three, the actions *ring* and *tick*. The action *measure* happens inside the alarm clock and from outside, we do not observe a difference if the action happens or not. But then, our definition does not reflect the reality, because clock 2 and clock 3 could also be produced the same way and the clients would both be happy. Therefore, we have to refine our definitions. We introduce the new, so called silent action τ and replace all not observable actions with this action. Now, we can say whenever an action τ occurs something internal happens, but we are not able to say what it is in particular. Furthermore, we could ask ourself why we not simply remove the non-observable actions instead of replacing them. A simple reason is given by Figure 4.

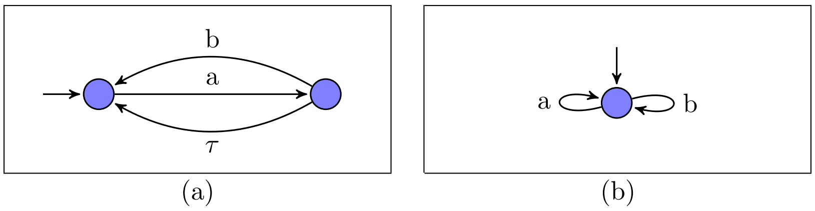


Figure 4: Examples showing why we need the silent action τ

In the left model, we can always observe an action a, because in the a-τ-loop we do not observe the τs. But if we remove the τ, then a-loops are no longer possible, because we have to do the b first. But also the modification depicted in the right model, where we introduce a single a-loop, produces the problem that now b-loops are possible without any as. Accordingly, the behavior of the left model cannot be produced with a model not using τs.

The new problem, that now occurs, is that the usage of a bisimulation is no longer sufficient to express equality, because it does not consider the silent action τ. Therefore, we introduce a new kind of bisimulation, which we will call *weak bisimulation*. We use the extended set of actions $Act_\tau = Act \cup \{\tau\}$ and the extended behavior expressions $\mathscr{E}_\tau$.

Definition. A relation $\mathcal{R} \subseteq \mathscr{E}_\tau \times \mathscr{E}_\tau$ is a <u>weak bisimulation</u> if, whenever $(E, F) \in \mathcal{R}$, $u \in Act_\tau$

(i) $E \xrightarrow{u} E' \;\Rightarrow\; \exists\, F' \in \mathscr{E}.\ F \xrightarrow{\tau}{}^{*} \xrightarrow{u} \xrightarrow{\tau}{}^{*} F' \;\wedge\; (E', F') \in \mathcal{R} \;\vee\; (E', F) \in \mathcal{R}$
(ii) $F \xrightarrow{u} F' \;\Rightarrow\; \exists\, E' \in \mathscr{E}.\ E \xrightarrow{\tau}{}^{*} \xrightarrow{u} \xrightarrow{\tau}{}^{*} E' \;\wedge\; (E', F') \in \mathcal{R} \;\vee\; (E, F') \in \mathcal{R}$
(iii) $\forall\, X \in Var.\ E \rhd X \Leftrightarrow F \rhd X$

Where $\xrightarrow{\tau}{}^{*} = \xrightarrow{\tau}_1 \xrightarrow{\tau}_2 \ldots \xrightarrow{\tau}_n$ for some $n \in \mathbb{N}$, what also allows to do no τ-transition at all.

The main idea using the definition of a weak bisimulation is, that we are now able to ignore sequences of τs when we compare two behavior expressions. Accordingly, the finite-state behavior also only consists of actions from Act. We now can define two behavior expressions E and F to be equal, if there exists a weak bisimulation $\mathcal{R}$, such that $(E, F) \in \mathcal{R}$ and write $E \approx F$. We get now new results by comparing our example alarm clock models. The modified models, where the action *measure* is replaced by the silent action τ, can be found in Figure 5.

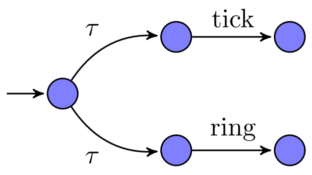
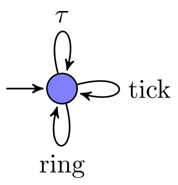
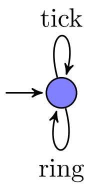
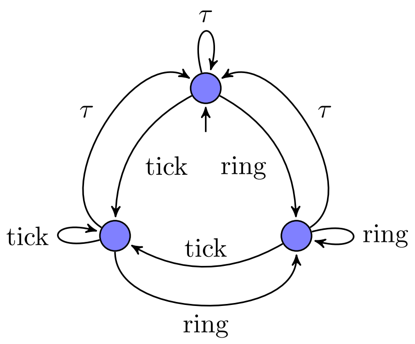
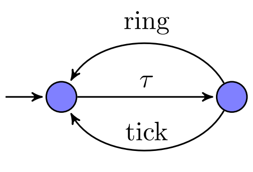
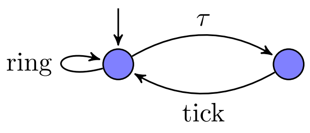
clock no.	**behavior expression**	**transition diagram**
1	$\tau.\mu X(tick.Y + X) + \tau.ring.0$	
2	$\mu X(\tau.X + ring.X + tick.X)$	
3	$\mu X(ring.X + tick.X)$	
4	$\mu X \mu Y \mu Z(\tau.X + ring.Y + tick.Z)$	
5	$\mu X(\tau.(ring.X + tick.X))$	
6	$\mu X(\tau.tick.X + ring.X)$	

Figure 5: Extended alarm clock models, where the action *measure* is replaced by τ

As before, clock 1 is incomparable to all other clocks, because it only have the finite traces *tick* and *ring*. Also clock 2 and clock 4 remain to be the same, where we can use the same relation, after replacing *measure* by τ, as a weak bisimulation as well. The difference to the old models now is that also clock 3 and clock 5 turn out to be equivalent to clock 2 and 4. This is proven by the existence of the weak bisimulations:

$$\{(\mu X(\tau.X + ring.X + tick.X), \mu X(ring.X + tick.X))\}$$

justifying clock 2 $\approx$ clock 3 and

$$\begin{aligned}\{(&\mu X(\tau.X + ring.X + tick.X),\\ &ring.(\mu X(\tau.(ring.X + tick.X))) + tick.(\mu X(\tau.(ring.X + tick.X)))),\\ (&\mu X(\tau.X + ring.X + tick.X), \mu X(\tau.(ring.X + tick.X)))\}\end{aligned}$$

justifying clock 2 $\approx$ clock 5. The other equivalences follow by transitivity. But against our intuition, clock 6 seems to step out of line. The problem is that if we compare for example clock 6 with clock 3 and let clock 6 do the silent action τ, then clock 3 must stay in the same state and does nothing. Now, clock 3 can do a *ring* and a *tick*, where clock 6 can only do a *ring*. So after definition, both processes are not equal. But if we compare the finite-state behaviors of both, they have the same traces, each arbitrary composition of *tick* and *ring*. The problem is that weak bisimulation is strictly stronger than the equivalence, defined through having the same τ-free traces. The reason why we prefer to use the definition of weak bisimulation here is that we want to argue about behavior expressions, which can influence each other using synchronization actions, in the future too. Then there are some problems with "trace equivalence", which are solved by using weak bisimulation. Since this goes beyond the scope of this paper, we simply accept this circumstance. Further informations to this problem can be found for example in [1]. Nevertheless, there still exists another problem with our definition of weak bisimulation, the problem that weak bisimulation is not a congruence relation. A congruence relation is a relation with the property, that whenever the relation proves two behavior expressions E and F to be equal, then the expression E', which we obtain by replacing a subexpression X of E by an equal subexpression X', is equal to F too. Consider for example the subexpression $tick.X$ of clock 3. It should be easy to see that this expression can be proven to be equal to the expression $\tau.tick.X$. But if we now replace the subexpression $tick.X$ with the equal subexpression $\tau.tick.X$, then we transform clock 3 into clock 6, which, as we have seen before, is not equal to clock 3. Some analyses have shown that for weak bisimulation this problem can only occur for an initial τ-transition. Accordingly, the general idea to solve the problem is to extend our definition of a weak bisimulation $\approx$ to a weak congruence $\approx^c$ such that whenever one behavior expression can do an initial τ, we enforce equal behavior expressions also to do this initial τ. We get as a definition for $\approx^c$:

Definition. $E \approx^c F$ iff for $u \in Act_\tau$

(i) $E \xrightarrow{u} E' \;\Rightarrow\; \exists\, F' \in \mathscr{E}.\ F \xrightarrow{\tau} \ldots \xrightarrow{\tau} \xrightarrow{u} \xrightarrow{\tau} \ldots \xrightarrow{\tau} F' \;\wedge\; E' \approx F'$

(ii) $F \xrightarrow{u} F' \;\Rightarrow\; \exists\, E' \in \mathscr{E}.\ E \xrightarrow{\tau} \ldots \xrightarrow{\tau} \xrightarrow{u} \xrightarrow{\tau} \ldots \xrightarrow{\tau} E' \;\wedge\; E' \approx F'$

(iii) $\forall\, X \in Var.\ E \triangleright X \Leftrightarrow F \triangleright X$

Consider that after the first transition, we are able to use the our standard weak bisimulation again and that we are really enforcing the first τ here. Using this definition, we are now able to check whether behavior expressions are equal and the definition respects our intuition. Furthermore, clock 2 to clock 5 can still be proven to be equal and the subexpression $tick.X$ is no longer equivalent to the subexpression $\tau.tick.X$.

3 Axiomatization

Using the formal definition of weak congruence seems often to be not very manageable. Like transition diagrams are a good representation for human readers, in contrast to the representation as behavior expressions, which is better for showing formal properties, we wish to have similarly an alternative formalization for proving behavior expressions to be equal. We get such a formalism if we encode our equality constraints into axioms. An axiom is a defined transformation, which is given by an equation, such that if we have a behavior expression which fits into one side of the equation, we can transform it into the corresponding other representation while preserving equality. Milner provides in his paper the list of axioms given in Figure 6. The main ambition of Milner's paper was to prove that these axioms are equivalent to the weak congruence definition. If we want to prove, that both formalisms define the same kind of equality we have to prove that the axiom system is sound and complete. Proving soundness normally counts to the easier parts. We only have to show that for each axiom the right expression and the left expression are equivalent under weak congruence. So, providing the corresponding equivalences is sufficient here. This is

	Summation axioms		**τ-axioms**
S1:	$E + F = F + E$	T1:	$u.\tau.E = u.E$
S2:	$E + (F + G) = (E + F) + G$	T2:	$E + \tau.E = \tau.E$
S3:	$E + E = E$	T3:	$u.(E + \tau.F) + u.F = u.(E + \tau.F)$
S4:	$E + 0 = E$		

	Recursion axioms
R1:	$\mu X E = E\{\mu X E/X\}$
R2:	If $F = E\{F/X\}$ then $F = \mu X E$, provided X is guarded in E
R3:	$\mu X(X + E) = \mu X E$
R4:	$\mu X(\tau.X + E) = \mu X \tau.E$
R5:	$\mu X(\tau.(X + E) + F) = \mu X(\tau.X + E + F)$

Figure 6: Axioms for weak observational congruence provided by Milner

mainly the reason why Milner skips these proofs in his paper and we follow suit. Giving a proof for completeness, which shows that for each pair of expressions, which are proven by a weak congruence to be equal, there exists also a conversion from one expression into the other using the axioms, would be more complicate. We analyze this proof more deeply in the next section.

4 The proof structure

As we have seen before, finding a conversion for two "equal" behavior expressions could be a complicate task. Therefore, we will present a formal step by step algorithm, which provides us such a conversion. The only conversion rules we use are the aforementioned axioms. Accordingly, proving the correctness of the individual steps would simultaneously prove the completeness of our axioms.

During this algorithm, we transform our behavior expressions into an alternative representation: an equation system S. This equation system S consists of a set $\widetilde{X} \cup \widetilde{W}$ of variables, where $\widetilde{X} = \{X_1, ..., X_n\}$ and $\widetilde{W} = \{W_1, W_2, ...\}$, and equations, which assign to each variable $X_i \in \widetilde{X}$ a behavior expression H_i over $\widetilde{X} \cup \widetilde{W}$:

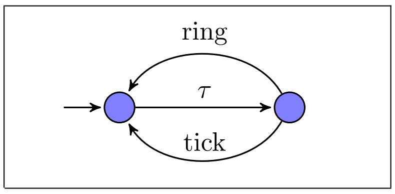
$$\begin{array}{ccc} X_1 & = & H_1 \\ \vdots & \vdots & \vdots \\ X_n & = & H_n \end{array}$$

The idea behind this representation is that if each H_i has the form

$$H_i = \sum_{X \in \widetilde{T}} u_{(X,i)}.X \; + \sum_{W \in \widetilde{R}} W$$

where $\widetilde{T} \subseteq \widetilde{X}$ and $\widetilde{R} \subseteq \widetilde{W}$, then an equation system directly corresponds to a transition diagram of a behavior expression. Each variable X_i represents a state of the transition diagram and the variables $W_i \in \widetilde{W}$ represent the unbounded variables in the corresponding behavior expression. We remark that, these variables are only visible in the transition diagram if they correspond to final states.

$$\begin{array}{ccl} X_1 & = & \tau.X_2 \\ X_2 & = & ring.X_1 + tick.X_1 \end{array}$$



equation system **transition diagram**

Figure 7: Equation system, describing the same finite-state behavior as the transition diagram of clock 5

According to this correspondence, we also call equation systems having this form *standard* equation systems. An example can also be seen in Figure 7. Unfortunately, unguarded variables are still possible within this representation, according to the existence of τ-loops. Therefore, we call an equation system *guarded* if and only if no such loops exist. Additionally, if there are not only no τ-loops, but also no unnecessary τ-paths, like if we reduce $X_1 = \tau.X_2$ and $X_2 = a.X_3$ to $X_1 = a.X_3$, then we call the equation system *saturated*.

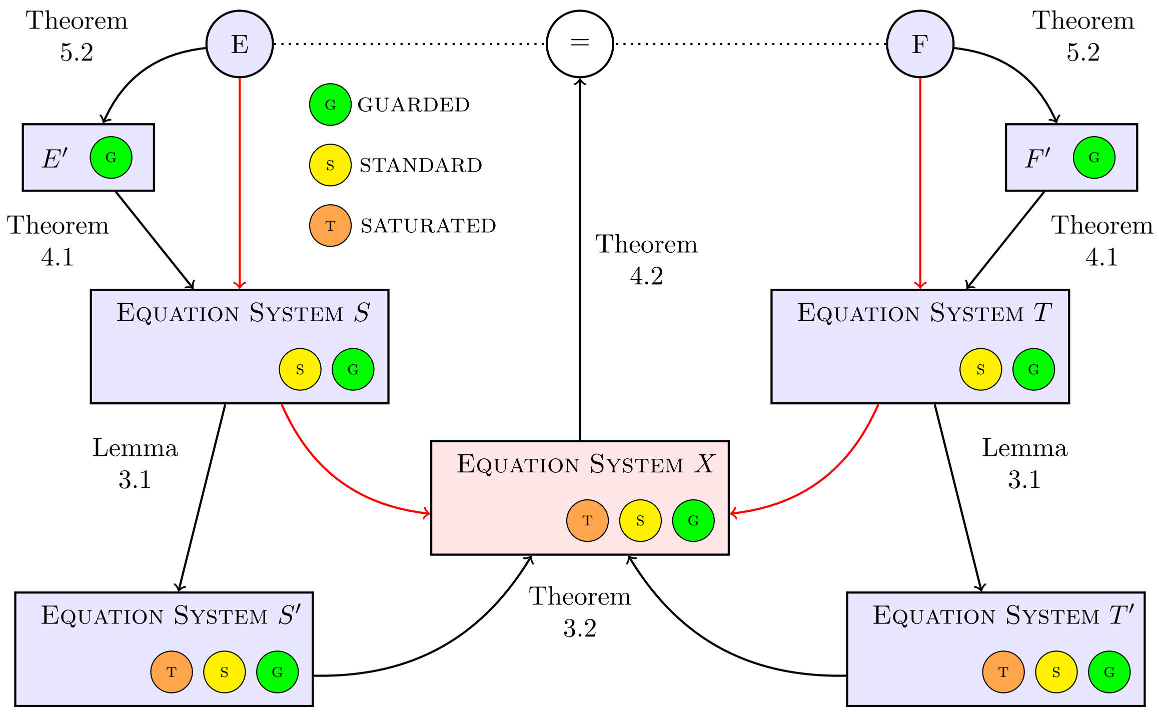


Figure 8: General prove structure

Now we are able to analyze the given transformation algorithm, which directly corresponds to the final proof structure. A graphical representation is given by Figure 8. Assume we have given two behavior expressions E and F, for which a weak congruence exists, which proves them to be equal. Then in a first step, we transform them into new behavior expressions E' and F', which only consist of guarded variables, using the axioms. Then, in the next step, we create an equation system, which represents the same finite-state behavior as these expressions. We also say, the behavior expression E' *provably satisfies* the equation system S. The equation systems we create are standard equation systems. According to the fact that we used a guarded expression for the transformation, our equation system will also be guarded. In the next step, we transform the equation systems S and T in such a way that they also have the property to be saturated. Finally, we create a common equation system X such that both E' and F' provably satisfy this system. Therefrom it follows directly that there is a conversion, described by transforming from E to X and then doing the transformations of the conversion from F to X in the backward direction. We now analyze each of these steps in more detail. The theorems labeled on the edges of Figure 8 correspond to the theorems proving these conversions in Milner's paper.

4.1 Creating guarded expressions [Theorem 5.2]

In this step we want to transform behavior expressions with unguarded variables into guarded ones. Therefore we only need the recursion axioms R3, R4, and R5, which cover all possible cases. Either the variable occurs directly unguarded without any τs, then we can simply remove it (R3), there could be a single τ-transition to the variable itself (R4), also represented by a τ-self-loop in the transition graph, or there exists a complete τ-path (R5). To clarify these transformations, have a look at Figure 9, where for simplicity we assume that $E \xrightarrow{u} E'$ and $F \xrightarrow{u'} F'$ are the only possible transitions of E and F. As we can see here, larger τ-loops can be removed by repetitive applying axiom R5 and folding the τs into the initial τ-self-loop. Consider that if there is no such self-loop, but only a larger τ-loop, we can initially create it by applying R4 once. After the larger τ-loop has vanished, we can apply R4 one more time and there is no τ-loop left. Remember that we need this τ-self-loop here and also the additional τ after applying R4 to take care of the initial τ, which is forced by the weak congruence in the initial step. Accordingly, we have to take care that this τ is still possible.

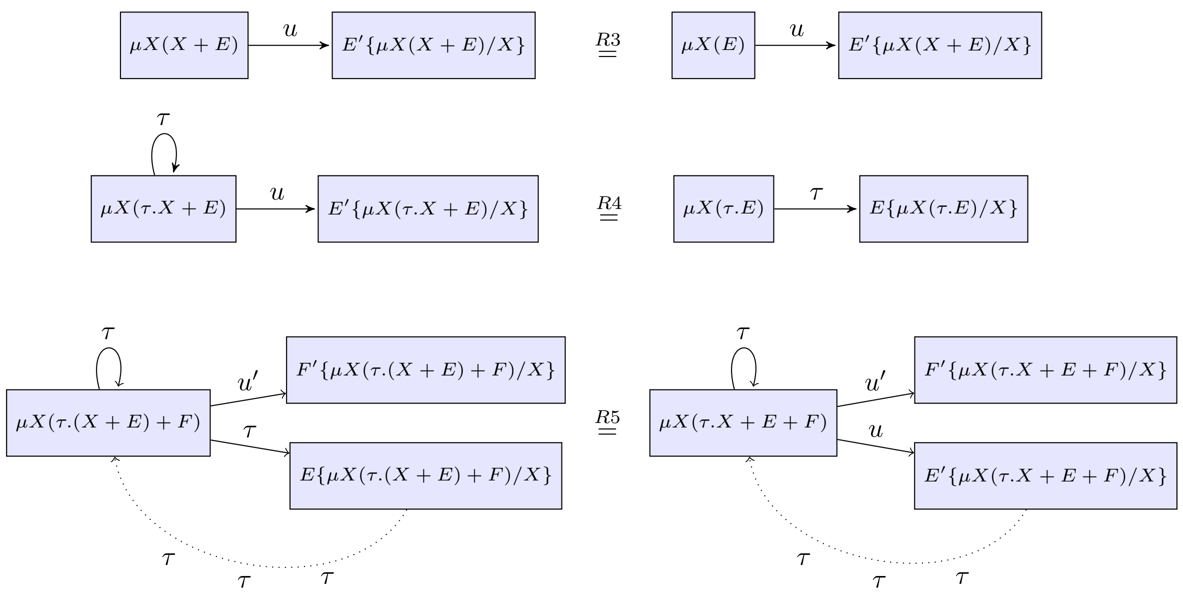


Figure 9: Graphical representation for the axioms R3, R4 and R5

4.2 Creating equation systems [Theorem 4.1]

In the next step we want to create the corresponding equation system for a given guarded behavior expression. This works in theory the same way, like also in practice. For each state in our transition relation, we create a new variable X_i and add the successor states as summands to the corresponding behavior expression H_i. Then we extend this behavior expression by the free variables in the expression. The created equation system is then a standard equation system and we do not introduce τ-loop during this process, so it is also guarded. For an example compare the behavior expression of clock 5 with the corresponding equation system given in Figure 7.

4.3 Making an equation system saturated [Lemma 3.1]

Now we want to transform a given equation system in such a way that no unnecessary τ-path exists anymore. This can be done using the τ-axioms T1, T2 and T3. Therefore, we simply replace variables X_i in the behavior expressions H_i by their full definition, given by the equation system, if they are guarded by a τ. Then using the τ-axioms, we can transform the new equation system into a standard equation system again. This corresponds to the same transformation we use to remove τ-transitions in the transition diagram and then replace them with new outgoing transitions for all outgoing transitions from the τ-transition's destination, using the same labels. An example is given in Figure 10.

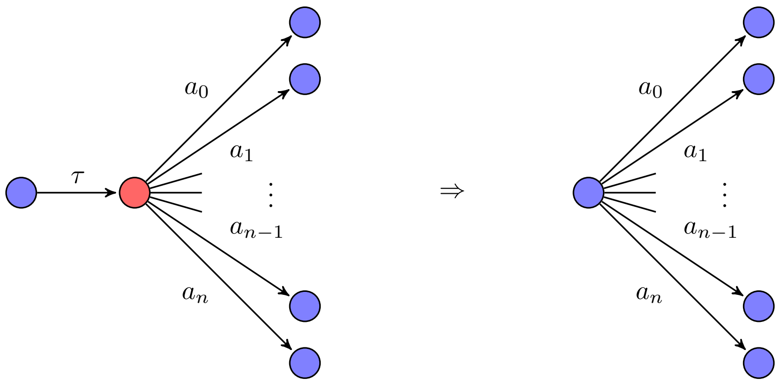


Figure 10: Removing τs in transition diagrams

4.4 Creating the joint equation system [Theorem 3.2]

Finally, we want to join both created equation systems in such a way that both original expressions provably satisfy this common equation system. The idea therefore works like follows. If we know that both equation systems represent equivalent finite-state behaviors, they also have equivalent transition diagrams. Regarding that we removed unnecessary τ-transitions from both expressions, the remaining τ-transitions must be essential. Accordingly, these τ-transitions are either initial τs or the kind of τs mentioned in the example of Figure 4. As a result, we can cover these τs like normal actions. Therewith our problem is reduced to the original problem of a bisimulation and the only problem occurring here was that a transition diagram can have multiple states that force an equivalent behavior, like for clock 2 and clock 4. In our equation systems this problem is represented by multiple variables with the same behavior expressions H_i on the right side[4]. It directly follows that we can then create a correct common equation system by using the superset of the equations, where we only have to rename indices. This means that each equation in one system has a corresponding equation in the other system. Figure 11 mentions this relation for clock 4 and clock 5, where we can join $X_{0,1,2}$ and X_4 as well as X_3 and X_5. Finally we get a new equation system which is provably satisfied by both original behavior expressions.

[4]except for the ordering of the subexpressions

$$\boxed{\begin{array}{lcl} X_0 &=& \tau.X_3 \\ X_1 &=& \tau.X_3 \\ X_2 &=& \tau.X_3 \\ X_3 &=& ring.X_1 + tick.X_2 \end{array}}$$

equation system for clock 4

$$\boxed{\begin{array}{lcl} X_4 &=& \tau.X_5 \\ X_5 &=& ring.X_4 + tick.X_4 \end{array}}$$

equation system for clock 5

$$\boxed{\begin{array}{lcl} X_{0,4} &=& \tau.X_3 \\ X_{1,4} &=& \tau.X_3 \\ X_{2,4} &=& \tau.X_3 \\ X_{3,5} &=& ring.X_1 + tick.X_2 \end{array}}$$

common equation system

Figure 11: Single and common guarded, standard and saturated equation systems for clock 4 and clock 5

Now, using these conversion rules to transform the behavior expression E into the common equation set X and going the reverse direction for F, only using the mentioned axioms, directly proves the completeness.

5 Conclusion

As we have seen, the set of axioms provided by Milner indeed gives us a feasible definition of an equality, similar to the definition of weak congruence. Recapitulating our previous analyses, we therefore first got a basic intuition of finite-state behaviors and different kind of equalities, where we decided that weak congruence fulfills our requirements. Then as a major step, we analyzed the completeness proof, which shows that by using a chain of logical transformations we are able to transform two behavior expressions, which are equal by weak congruence, in a common representation only using our axioms. Finally, we derived that this transformation also gave us a transformation from the behavior expression into an equal one.

Regarding this outcome, Milner provided one of the major results in this area. Most of todays research depends on his first analyses and many other researchers follow his approach to further extend the basic definitions, as mentioned for example in [2], [3], [4] or [5]. But Milner also continues his research by analyzing other process calculi, like the π-calculus or different action-calculi, as well as other formalisms in concurrency theory. Clearly, he counts to one of the main pioneers in this area.

References

[1] Robin Milner. Lectures on a calculus for communicating systems. In Stephen Brookes, Andrew Roscoe, and Glynn Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 197–220. Springer Berlin / Heidelberg, 1985. 10.1007/3-540-15670-4_10.

[2] R. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In Andrzej Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993*, volume 711 of *Lecture Notes in Computer Science*, pages 473–484. Springer Berlin / Heidelberg, 1993. 10.1007/3-540-57182-5_39.

[3] Alexander Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 802 of *Lecture Notes in Computer Science*, pages 530–543. Springer Berlin / Heidelberg, 1994. 10.1007/3-540-58027-1_25.

[4] Luca Aceto and Alan Jeffrey. A complete axiomatization of timed bisimulation for a class of timed regular behaviours. *Theoretical Computer Science*, 152(2):251 – 268, 1995.

[5] Mario Bravetti and Roberto Gorrieri. A complete axiomatization for observational congruence of prioritized finite-state behaviors. In Ugo Montanari, José Rolim, and Emo Welzl, editors, *Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 744–755. Springer Berlin / Heidelberg, 2000. 10.1007/3-540-45022-X_62.