Modular μ-calculus model-checking with formula-dependent hierarchical abstractions

Yves-Stan Le Cornec, Franck Pommereau

Modular μ-calculus model-checking with formula-dependent hierarchical abstractions

Yves-Stan Le Cornec, Franck Pommereau
IBISC, University Paris-Sud/Saclay
IBGBI, 23 bd de France, 91037 Évry
France
Email: yves-stan.lecornec@ibisc.univ-evry.fr, franck.pommereau@ibisc.univ-evry.fr

***Abstract*—This paper defines a formal framework for the modular and hierarchical model-checking of μ-calculus against modular transitions systems. Given a formula φ, a module can be analysed alone, in such a way that the truth value of φ may be decided without the need to analyse other modules. If no conclusion can be drawn locally, the analysis provides information allowing to reduce the module to a smaller one that is equivalent with respect to the truth value of φ. This way, modules can be incrementally analysed, reduced and composed to other reduced modules until a conclusion is reached. On the one hand, modular analysis allows to avoid modules compositions and thus the corresponding combinatorial explosion; on the other hand, hierarchical analysis allows to reduce the modules that must be composed, which limits combinatorial explosion. Moreover, by proposing three complementary formula-dependent reductions, we expect better reductions than general approaches like bisimulation or τ^* reductions. The current paper is focused on defining the theoretical tools for this approach; finding interesting strategies to apply them efficiently is left to future work.**

I. Introduction

Model-checking is a verification technique that suffers from the well-know *state explosion problem*. Among the numerous approaches to alleviate this problem, one way is to exploit modularity in the system under analysis by trying to analyse its modules independently of each other as much as possible. Doing so, subsystems smaller than the whole system are considered, which reduces combinatorial explosion. The difficulty is then to conclude globally by reassembling local conclusions instead of reassembling the system itself, which would bring back the combinatorial explosion we tried to avoid.

In this paper, we propose to combine modular analysis with hierarchical abstraction for the model-checking of μ-calculus formula against modular systems defined as the synchronised product of labelled transitions systems (LTS). Our goal is to avoid as much as possible to actually build the whole synchronisation. Take a formula φ and a modular LTS $(S_1, \dots, S_n)$ whose flat semantics is the synchronised product $S \overset{\text{df}}{=} S_1 \otimes \dots \otimes S_n$. We define tools to check φ against any S_i, allowing to either conclude about the global truth value of φ on S, or to abstract away from S_i any information that is only local to it and unrelated to φ. This results in a smaller LTS S'_i that is equivalent to S_i with respect to the truth value of φ in the context of S globally. In other words, S'_i may safely replace S_i to construct S without changing the conclusion about φ. Then, assuming that another module S_j is analysed and abstracted to S'_j, it is possible to compute $S'_i \otimes S'_j$ (that is likely to be much smaller that $S_i \otimes S_j$) and to apply on it the same technique, resulting on either a conclusion about the truth value of φ, or an abstracted representation of this subsystem, in which additionally to what was previously removed from S_i and S_j, we have also removed unnecessary information about how the two modules interact. This leads to an incremental analysis in which modules are progressively analysed, abstracted and combined in a hierarchical fashion, until a conclusion is reached. Because formula-dependent abstraction is used at every level of the construction, the combinatorial explosion is constantly limited to only what is necessary to conclude about φ. Moreover, it can be expected that a conclusion is reached before all the modules have to be considered.

The work presented in this paper is focused on defining theoretical tools to apply the method explained above. The next section defines modular LTS, then section 3 defines the μ-calculus and its semantics. Given a formula φ and an LTS S, our method consists of five steps. (1) In section IV-A a function Pass^φ (resp. Fail^φ) is computed, that returns for every state x a formula that the environment has to verify in order to ensure that φ holds at x (resp. φ does not hold). If Pass^φ (resp. Fail^φ) is true at the initial state, then we can immediately conclude about φ globally. (2) Based on Pass^φ and Fail^φ, in section IV-B, a relation $\langle\!\langle\varphi\rangle\!\rangle$ is computed such that, for any states x and y, when $x\langle\!\langle\varphi\rangle\!\rangle y$ then x and y can be interchanged to evaluate the truth of φ, whatever the environment is. (3) In section V-A, our knowledge of the formula is injected by computing a product graph (namely S^φ) between the LTS S and the formula φ. On the one hand this allows to prune off parts of the module which are unnecessary to the verification of the property, on the other hand, it provides us with information about the verification algorithm behavior which is useful in the computation of the subsequent reductions. (4) The second reduction is defined in section V-B where we show that any state x of S^φ can be replaced by another state y such that $x\langle\!\langle\varphi\rangle\!\rangle y$ as long as y does not depend on x to compute the value of φ. This reduces the graph if less states are reachable from y than from x. For instance, if the system starts with an initialization phase that is not considered in φ, then the corresponding states can be dropped this way. (5) Finally, in section V-C, we define another formula dependent equivalence relation $\approx^\varphi$ on the remaining states. Unlike $\langle\!\langle\varphi\rangle\!\rangle$ this relation is preserved by merging equivalent states, so we can quotient the LTS resulting from the previous step by this relation. This is our third reduction. Section VI showcases some experimental results obtained with the reduction from V-B. We conclude with a comparison to related works and some perspectives.

In particular, we will work on defining practical approaches to effectively use this framework and reach the expected improvements. All the proofs are given in the appendix that will be moved to a technical report if the paper is accepted.

II. Modular labelled transitions systems

We define a modular LTS as a straightforward extension of LTS based on the synchronized products with respect to some transitions labels, distinguished as synchronized actions by the fact that they appear in more than one module.

Definition 1: A LTS is a tuple $S \overset{\mathrm{df}}{=} (Q, q_0, A, R, L)$ where Q is a set of *states*, $q_0 \in Q$ is the *initial state*, A is the set of *actions* used as transition labels, $R \subseteq Q \times A \times Q$ is the set of *transitions* and $L : Q \mapsto \mathbb{B}(\mathbb{V})$ is a labelling of states with Boolean formulas on propositional variables from a set $\mathbb{V}$. A transition $(q, a, q') \in R$ is usually denoted by $q \xrightarrow{a} q'$.

Definition 2: A *modular LTS* is a collection of LTS $(Q_i, q_{0i}, A_i, R_i, L_i)_{1 \le i \le n}$ where each A_i is partitioned as $A_i^{\mathsf{loc}} \uplus A_i^{\mathsf{fus}}$ such that, for $1 \le i \le n$: $A_i^{\mathsf{loc}} \overset{\mathrm{df}}{=} A_i \setminus \bigcup_{j \ne i} A_j$ and $A_i^{\mathsf{fus}} \overset{\mathrm{df}}{=} A_i \setminus A_i^{\mathsf{loc}}$.

The behaviour of a modular LTS is the *synchronised product* of its components, where synchronisation takes place on the fused transitions (those labelled by action in some A_i^{fus}), which is the usual definition of a n-ary synchronised product. We denote by $x[i]$ the i-th component of a tuple x and by $x[i \leftarrow y_i]$ the tuple x in which the i-th component has been replaced by y_i, this latter notation is naturally extended to replace several components.

Definition 3: Let $(S_i)_{1 \le i \le n}$ be a modular LTS with $S_i \overset{\mathrm{df}}{=} (Q_i, q_{0i}, A_i, R_i, L_i)$ and $A_i \overset{\mathrm{df}}{=} A_i^{\mathsf{loc}} \uplus A_i^{\mathsf{fus}}$, the *synchronised product* of the S_i's, is the LTS (Q, q_0, A, R, L) defined by:

- $Q \overset{\mathrm{df}}{=} \prod_{1 \le i \le n} Q_i$;
- $q_0 \overset{\mathrm{df}}{=} (q_{01}, \dots, q_{0n})$;
- $A \overset{\mathrm{df}}{=} \bigcup_{1 \le i \le n} A_i$;
- R is the smallest subset of $Q \times A \times Q$ such that $x \xrightarrow{a} y \in R$ iff either
  - $\exists i$ such that $a \in A_i^{\mathsf{loc}}$: $x[i] \xrightarrow{a} y_i \in R_i$ and $y = x[i \leftarrow y_i]$,
  - $\forall i$ such that $a \in A_i^{\mathsf{fus}}$: $x[i] \xrightarrow{a} y_i \in R_i$ and $y = x[i \leftarrow y_i \mid a \in A_i^{\mathsf{fus}}]$;
- for all $x \in Q$, $L(x) \overset{\mathrm{df}}{=} \bigwedge_{1 \le i \le n} L_i(x[i])$.

Because this product is associative and commutative, we shall also use a binary notation for it: $S_1 \otimes \cdots \otimes S_n$.

In the definition of R above, the first point corresponds to the cases where a module evolves on a local transition. So only one component of the compound state evolves. The second point corresponds to the firing of a fused transition, in which case all the modules sharing this transition must simultaneously fire and the corresponding components of the compound state will simultaneously evolve. Note that by definition, if $a \in A_i^{\mathsf{fus}}$ for some i, then there exists at least one $j \ne i$ such that $a \in A_j^{\mathsf{fus}}$ also (otherwise, we would have $a \in A_i^{\mathsf{loc}}$).

III. The μ-calculus

The (modal) μ-calculus is a temporal logic that encompasses widely used logics such as, in particular, CTL* (and thus also LTL and CTL) [1], [2]. A μ-formula is derived from the following grammar, where B is a Boolean formula, X is a propositional variable and α is a set of actions:

$$\varphi \quad ::= \quad B \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\varphi \mid \mu X.\varphi \mid X$$

Moreover, in a formula $\mu X.\varphi$, φ must be positive in the variable X, *i.e.*, every free occurrence of X must be in the scope of an even number of negations $\neg$. A formula is said to be in positive form if all its negations occur applied to atomic propositions.

A formula φ is evaluated over an LTS $S \overset{\mathrm{df}}{=} (Q, q_0, A, R, L)$ and can be seen as a function of its free variables to 2^Q. In particular, if φ is a closed formula then it is a function with no arguments that returns the subset of Q where φ holds. In formula $\mu X.\langle a\rangle X \vee B$, the sub-formula $\langle a\rangle X \vee B$ defines a function that, given $X \subseteq Q$, returns the states y such that either $y \xrightarrow{a} x$ for some $x \in X$, or B holds on y. From this point of view, $\mu X.\varphi$ is the least fixed point of function φ. More generally, the semantics φ^S of φ over S is defined as follows:

- B holds in every state whose label implies B: $B^S \overset{\mathrm{df}}{=} \{x \in Q \mid L(x) \Rightarrow B\}$;
- $\varphi_1 \vee \varphi_2$ holds in every state where φ_1 or φ_2 holds: $(\varphi_1 \vee \varphi_2)^S \overset{\mathrm{df}}{=} \varphi_1^S \cup \varphi_2^S$;
- $\neg\varphi$ holds in every state where φ does not: $(\neg\varphi)^S \overset{\mathrm{df}}{=} Q \setminus \varphi^S$;
- $\langle\alpha\rangle\varphi$ holds in every state where a transition labelled by $a \in \alpha$ leads to a state where φ holds: $(\langle\alpha\rangle\varphi)^S \overset{\mathrm{df}}{=} \{x \in Q \mid \exists y \in \varphi^S, \exists a \in \alpha, x \xrightarrow{a} y \in R\}$;
- $\mu X.\varphi$ is the least fixed point of function φ: $(\mu X.\varphi)^S \overset{\mathrm{df}}{=} \bigcap_{\rho \in 2^Q \wedge \varphi(\rho) \subseteq \rho} \rho$;
- $X^S \overset{\mathrm{df}}{=} X$, can be seen as the identity function.

For closed formulas, φ^S is a subset of Q, which can be equivalently seen as the image of a function with no arguments. But for formulas with free variables, φ^S is a function exactly like φ is and the above definition can be read as transformation of functions. For instance, for a φ with a single free variable X, the definition of $(\neg\varphi)^S$ can be reformulated as $(\neg\varphi)^S(X) \overset{\mathrm{df}}{=} Q \setminus \varphi^S(X)$. Note that, to simplify the definitions in the sequel, we have considered α as a set of actions in $\langle\alpha\rangle$ instead of as a single action as usual. Moreover, because the semantics of a formula is a set of states, we may use one or the other form interchangeably.

The effective construction of $\mu X.\varphi$ is made inductively by defining $0X.\varphi \overset{\mathrm{df}}{=} \emptyset$, and $nX.\varphi \overset{\mathrm{df}}{=} \varphi[X \leftarrow (n-1)X.\varphi]$ where $\varphi[X \leftarrow Y]$ denotes φ in which variable X is substituted by Y everywhere. Knaster-Tarski's theorem ensures that there exists $k \in \mathbb{N}$ such that $kX.\varphi = \mu X.\varphi$. Indeed, φ is a monotonous function over a complete lattice, and thus the set of its fixed-points is also a complete lattice [3].

To write formulas more comfortably, we can use the following operators:

- $\varphi_1 \wedge \varphi_2 \overset{\text{df}}{=} \neg(\neg\varphi_1 \vee \neg\varphi_2)$;
- $[\alpha]\varphi \overset{\text{df}}{=} \neg\langle\alpha\rangle\neg\varphi$, which yields $([\alpha]\varphi)^S \overset{\text{df}}{=} \{x \in Q \mid \forall a \in \alpha, \forall x \xrightarrow{a} y \in T, y \in \varphi^S\}$;
- $\nu X.\varphi \overset{\text{df}}{=} \neg\mu X.\neg\varphi[X \leftarrow \neg X]$ is the greatest fixed-point of φ.

Finally, for $q \in Q$, we write $S, q \models \varphi$ iff $q \in \varphi^S$, and $S \models \varphi$ iff $S, q_0 \models \varphi$.

The closure of a formula φ is denoted $\mathcal{CL}(\varphi)$ and is constructed with the two following rules:

- $\mathcal{CL}(\varphi) = \{\varphi\} \bigcup_{\varphi'} \mathcal{CL}(\varphi')$ where φ' ranges over the subformulas of φ.
- $\mathcal{CL}(\sigma X.\varphi) = \{\sigma X.\varphi\} \cup \mathcal{CL}(\varphi(\sigma X.\varphi))$ for a fixed point formula (*i.e.*, $\sigma \in \{\mu, \nu\}$).

In the rest of the paper, we refer to the act of replacing the formula $\sigma X.\varphi$ by $\varphi(\sigma X.\varphi)$ as unfolding the formula.

IV. Formula dependent reduction

As seen in definition 3, we can build the global behavior of a modular LTS in a hierarchical way. Subsection IV-A defines the functions Pass^φ and Fail^φ. Those functions may enable the computation of the truth value of a formula by only analysing a subset of modules. They are also used to define the equivalence relations from definition 7 and figure 7, as well as the reduction from V-A where they can allow us to cut out part off the graph which we know are useless (because the analysis was already done locally).

A. Pass *and* Fail

We seek to identify states of one module S that are equivalent with respect to a formula φ, *i.e.*, states that we would not be able to tell apart by looking at the truth value of φ over the global system. The first step is to define, for any state x, the formulas $\mathsf{Pass}^\varphi(x)$ and $\mathsf{Fail}^\varphi(x)$ (φ may be omitted if understood from the context). These are Boolean formulas over the state variables appearing in φ which are external to the module S, they can be seen as propositional constants from the μ-calculus which can be evaluated on the label of any state of the environment (the environment Env being defined as the product of all the modules other that S). We want that if a state $env \in Env$ satisfies the formula $\mathsf{Pass}^\varphi(x)$ (resp. $\mathsf{Fail}^\varphi(x)$) then the formula φ is true (resp. false) in the global state (x, env). If one or the other evaluates to true on the initial state, then we can stop the analysis and provide a conclusion about φ globally. Otherwise, we will use these two functions to reduce the analysed module (see the next section).

Definition 4: Let x be a state of an LTS $S = (Q, q_0, A, R, L)$, and B a state formula. We denote by $B@x$ the formula $B[v \leftarrow v@x \mid v \in \mathbb{V}]$, that is B in which each propositional variable v that is local to S is replaced by its value in x.

Definition 5 extends the order relation based on the implication of Boolean formulas to the set of functions used in the construction of Pass and Fail.

Definition 5: Let Q be the states of an LTS S, φ a μ-calculus formula, $\mathcal{V}_{env}$ the variables from φ which are external to S, $\mathbb{B}(\mathcal{V}_{env})$ the set of all Boolean formulas on these variables, and two functions f and $g : Q \to \mathbb{B}(\mathcal{V}_{env})$. We define $\overset{Q}{\Rightarrow}$ by: $f \overset{Q}{\Rightarrow} g$ iff $f(x) \Rightarrow g(x)$ for every $x \in Q$.

$$\mathsf{Pass}^B(x) \overset{\text{df}}{=} B@x$$
$$\mathsf{Pass}^{\varphi_1 \wedge \varphi_2}(x) \overset{\text{df}}{=} \mathsf{Pass}^{\varphi_1}(x) \wedge \mathsf{Pass}^{\varphi_2}(x)$$
$$\mathsf{Pass}^{\varphi_1 \vee \varphi_2}(x) \overset{\text{df}}{=} \mathsf{Pass}^{\varphi_1}(x) \vee \mathsf{Pass}^{\varphi_2}(x)$$
$$\mathsf{Pass}^{\langle l\rangle\varphi}(x) \overset{\text{df}}{=} \bigvee_{x \xrightarrow{l} x'} \mathsf{Pass}^\varphi(x')$$
$$\mathsf{Pass}^{\langle s\rangle\varphi}(x) \overset{\text{df}}{=} \bot$$
$$\mathsf{Pass}^{\langle e\rangle\varphi}(x) \overset{\text{df}}{=} \bot$$
$$\mathsf{Pass}^{[l]\varphi}(x) \overset{\text{df}}{=} \bigwedge_{x \xrightarrow{l} x'} \mathsf{Pass}^\varphi(x')$$
$$\mathsf{Pass}^{[s]\varphi}(x) \overset{\text{df}}{=} \begin{cases} \top & \text{if } \bigwedge_{x \xrightarrow{s} x'} \mathsf{Pass}^\varphi(x') = \top \\ \bot & \text{otherwise} \end{cases}$$
$$\mathsf{Pass}^{[e]\varphi}(x) \overset{\text{df}}{=} \begin{cases} \top & \text{if } \mathsf{Pass}^\varphi(x) = \top \\ \bot & \text{otherwise} \end{cases}$$
$$\mathsf{Pass}^{\mu X.\varphi}(x) \overset{\text{df}}{=} \bigwedge\{f(x) \mid \mathsf{Pass}^\varphi(f) \overset{Q}{\Rightarrow} f\}$$
$$\mathsf{Pass}^{\nu X.\varphi}(x) \overset{\text{df}}{=} \bigvee\{f(x) \mid \mathsf{Pass}^\varphi(f) \overset{Q}{\Leftarrow} f\}$$
$$\mathsf{Pass}^X(x) \overset{\text{df}}{=} X$$
$$\mathsf{Fail}^\varphi(x) \overset{\text{df}}{=} \mathsf{Pass}^{\neg\varphi}(x)$$

Fig. 1. Definition of Pass and Fail, where $\bot$ and $\top$ stand for Boolean values false and true respectively, and l, s and e are respectively local, shared and external sets of actions. Braces correspond to definitions by cases.

We are now in position to define Pass and Fail, with their main properties.

*Definition 6 (*Pass *and* Fail*):* Take a formula φ in positive form, Q the states of one module and $x \in Q$. Pass^φ and Fail^φ for this state are defined by the rules given in figure 1.

Lemma 1: The computation of Pass and Fail terminates.

*Lemma 2 (*Pass *and* Fail *are compositional):* Let S_1 and S_2 be two modules from a modular LTS. For any states x_1 from S_1 and x_2 from S_2, we have $\mathsf{Pass}^\varphi(x_1)@x_2 \Rightarrow \mathsf{Pass}^\varphi((x_1, x_2))$ and $\mathsf{Fail}^\varphi(x_1)@x_2 \Rightarrow \mathsf{Fail}^\varphi((x_1, x_2))$.

*Lemma 3 (*Pass *and* Fail *are correct):* For any $x \in S$ and $env \in Env$, if $env \models \mathsf{Pass}^\varphi(x)$, then $(x, env) \models \varphi$ and $env \models \mathsf{Fail}^\varphi(x) \Rightarrow (x, env) \not\models \varphi$

Theorem 1 (relation with the μ-calculus semantics): Let φ be a formula that only contains actions and propositional constants which are local to an LTS S, then $\varphi^S = \{x \mid \mathsf{Pass}^\varphi(x) = \top\}$.

Example 1: let us consider the formula $\Phi \overset{\text{df}}{=} \mu X.(a \Leftrightarrow v) \vee \langle l\rangle X$, which means that a state where a has the same value as v is reachable by only firing action l, and compute Pass^Φ on the LTS N from figure 2.

With respect to this graph, a is a local variable, v is an external variable and l is a local action. Because Φ is a fixed-point formula, Pass^Φ will also be computed as a fixed point. The initialization P_0 is the function which return $\bot$ for

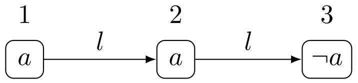

Fig. 2. Example of an LTS N

every state, and we can compute the fixed point by using the following recurrence relation. For any state x from N,

$$\begin{aligned}
P_0(x) &\overset{\text{df}}{=} \bot \\
P_n(x) &\overset{\text{df}}{=} (\mathsf{Pass}^{a\Leftrightarrow v \vee \langle l \rangle \Phi}(x))[Pass^{\Phi} \leftarrow P_{n-1}] \\
&= (\mathsf{Pass}^{a\Leftrightarrow v}(x) \vee \mathsf{Pass}^{\langle l \rangle \Phi}(x))[Pass^{\Phi} \leftarrow P_{n-1}] \\
&= (a \Leftrightarrow v)@x \vee \bigvee_{x \xrightarrow{l} x'} P_{n-1}(x')
\end{aligned}$$

The computation of Pass^{Φ} on graph N is thus performed as follows:

$$\begin{array}{ll}
\text{step 0:} & P_0(x) = \bot \qquad \forall x \\
\text{step 1:} & P_1(1) = v \vee P_0(2) = v \\
 & P_1(2) = v \vee P_0(3) = v \\
 & P_1(3) = \neg v \\
\text{step 2:} & P_2(1) = v \vee P_1(2) = v \vee v = v \\
 & P_2(2) = v \vee P_1(3) = v \vee \neg v = \top \\
 & P_2(3) = \neg v \\
\text{step 3:} & P_3(1) = v \vee P_2(2) = v \vee \top = \top \\
 & P_3(2) = v \vee P_2(3) = v \vee \neg v = \top \\
 & P_3(3) = \neg v \\
\text{step 4:} & \quad P_4 = P_3 \qquad \text{(the fixed-point is reached)}
\end{array}$$

Knowing that module N is in state 1 or 2, is thus sufficient to conclude that the formula Φ holds on the global system. However if N is in state 3, we know that Φ holds if the environment is in a state verifying $\neg v$.

B. Local implication

This subsection is about finding local states which are indistinguishable from one another by the formula, so we can use this information in section V-B to reduce the size of the module. More precisely, we are interested in finding *local implication relations* between the states of a module S. Given a formula φ, we look for pairs of local states (x, y) such that for any state of the environment $env \in Env$, each time φ is true in state (x, env) then it will also be true in state (y, env). In order to find these equivalent states, we define a formula that has to be verified by the state label of the environment in order to ensure that this implication holds.

Definition 7 (local implication conditions): Let Q be the states of a module S, and φ and ψ be two formulas. Let $\mathcal{V}$ be the states variables from φ and ψ which are external to S. We define the local implication conditions $\langle\langle \varphi, \psi \rangle\rangle : Q^2 \mapsto \mathbb{B}(\mathcal{V}_{env})$ by the rules given in figure 3. We also write $x\langle\langle \varphi \rangle\rangle y$ iff $\langle\langle \varphi, \varphi \rangle\rangle(x, y) = \langle\langle \varphi, \varphi \rangle\rangle(x, y) = \top$, in which case x and y are said to be locally equivalent.

Intuitively, the rules from figure 3 can be understood as follows:

$$\begin{aligned}
&\langle\langle B_1, B_2 \rangle\rangle(x, y) = B_1@x \Rightarrow B_2@y \\
&\langle\langle \psi, \varphi_1 \vee \varphi_2 \rangle\rangle(x, y) = \langle\langle \psi, \varphi_1 \rangle\rangle(x, y) \vee \langle\langle \psi, \varphi_2 \rangle\rangle(x, y) \\
&\langle\langle \psi, \langle l \rangle \varphi \rangle\rangle(x, y) = \mathsf{Fail}^{\psi}(x) \vee \bigvee_{y \xrightarrow{l} y'} \langle\langle \psi, \varphi \rangle\rangle(x, y') \\
&\langle\langle \langle s \rangle \psi, \langle s \rangle \varphi \rangle\rangle(x, y) = \mathsf{Fail}^{\langle s \rangle \psi}(x) \\
&\qquad \vee \begin{cases} \top & \text{if } \bigwedge_{x \xrightarrow{s} x'} \mathsf{Fail}^{\psi}(x') \vee \bigvee_{y \xrightarrow{s} y'} \langle\langle \psi, \varphi \rangle\rangle(x', y') \\ \bot & \text{otherwise} \end{cases} \\
&\langle\langle \langle e \rangle \psi, \langle e \rangle \varphi \rangle\rangle(x, y) = \begin{cases} \top & \text{if } \langle\langle \psi, \varphi \rangle\rangle(x, y) \\ \bot & \text{otherwise} \end{cases} \\
&\langle\langle \mu X.\psi, \varphi \rangle\rangle(x, y) = \bigvee \{ f(x, y) \mid \langle\langle \psi, \varphi \rangle\rangle f \overset{Q^2}{\Leftarrow} f \} \\
&\langle\langle \psi, \varphi \rangle\rangle(x, y) = \langle\langle \neg\varphi, \neg\psi \rangle\rangle(y, x) \\
&\langle\langle \psi, \varphi_1 \wedge \varphi_2 \rangle\rangle(x, y) = \langle\langle \psi, \varphi_1 \rangle\rangle(x, y) \wedge \langle\langle \psi, \varphi_2 \rangle\rangle(x, y) \\
&\langle\langle \psi, [l] \varphi \rangle\rangle(x, y) = \bigwedge_{y \xrightarrow{l} y'} \langle\langle \psi, \varphi \rangle\rangle(x, y') \\
&\langle\langle \nu X.\psi, \varphi \rangle\rangle(x, y) = \bigwedge \{ f(x, y) \mid \langle\langle \psi, \varphi \rangle\rangle f \overset{Q^2}{\Rightarrow} f \} \\
&\langle\langle \psi, \varphi \rangle\rangle(x, y) = \mathsf{Fail}^{\psi}(x) \vee \mathsf{Pass}^{\varphi}(y)
\end{aligned}$$

Fig. 3. Definition of the local implication conditions, when several rules can be applied, we choose the one that is first defined in the list above.

- case $\langle\langle B_1, B_2 \rangle\rangle$: B_1 and B_2 are state properties which can be evaluated on the global system. Note that $\langle\langle B_1, B_2 \rangle\rangle(x, y)$ is equal to the formula $\mathsf{Fail}^{B_1}(x) \vee \mathsf{Pass}^{B_2}(y)$. If a state $env \in Env$ verifies this property then $(x, env) \models B_1 \Rightarrow (y, env) \models B_2$ holds;
- case $\langle\langle \psi, \varphi_1 \vee \varphi_2 \rangle\rangle$: if for any env, each time that $(x, env) \models \psi$ then $(y, env) \models \varphi_1$ or $(y, env) \models \varphi_2$, then we have $(x, env) \models \psi \Rightarrow (y, env) \models \varphi_1 \vee \varphi_2$;
- case $\langle\langle \psi, \langle l \rangle \varphi \rangle\rangle$: if $env \models \langle\langle \psi, \langle l \rangle \varphi \rangle\rangle(x, y)$ then either $env \models \mathsf{Fail}^{\psi}(x)$ (in which case we know that $(x, env) \not\models \psi$) or there exists some local transition $y \xrightarrow{l} y'$ such that $\langle\langle \psi, \varphi \rangle\rangle(x, y')$. If $(x, env) \models \psi$ implies $(y', env) \models \varphi$ then we know that it implies $(y, env) \models \langle l \rangle \varphi$. Because l is a local action, we will always be able to fire it as part of the global system;
- case $\langle\langle \langle s \rangle \psi, \langle s \rangle \varphi \rangle\rangle$: s is an action which is synchronised between the local module and the environment. Because we have no knowledge of the environment, we need that for every x' there exists y' such that $\langle\langle \psi, \varphi \rangle\rangle(x', y')$. This way if $(x, env) \models \langle s \rangle \psi$, then we can find a y' such that $(y', env') \models \varphi$ which means that $(y, env) \models \langle s \rangle \varphi$;
- case $\langle\langle \langle e \rangle \psi, \langle e \rangle \varphi \rangle\rangle$: e is an external action. As before, since we have no information about the environment, we need $\langle\langle \psi, \varphi \rangle\rangle(x, y)$ to be true on any possible state;
- case $\langle\langle \mu X \psi, \varphi \rangle\rangle$: define $\Psi \overset{\text{df}}{=} \mu X \psi$, and note that ψ contains only the free variable X while φ contains no free variable. The computation of $\langle\langle \psi(\Psi), \varphi \rangle\rangle$ can lead to an expression which depends on $\langle\langle \Psi, \varphi \rangle\rangle$ as shown on figure 4. We thus have the recurrence relation $\langle\langle \Psi, \varphi \rangle\rangle = \langle\langle \psi, \varphi \rangle\rangle(\langle\langle \Psi, \varphi \rangle\rangle)$, where $\langle\langle \psi, \varphi \rangle\rangle$ is a function from $(Q^2 \to \mathbb{B}(\mathcal{V}))$ to $(Q^2 \to \mathbb{B}(\mathcal{V}))$ and we compute the greatest fixed point of this function

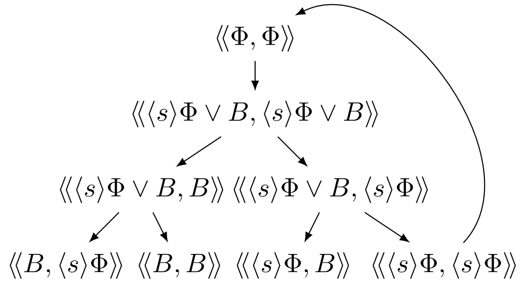

Fig. 4. Dependency graph of $\langle\!\langle \Phi, \Phi \rangle\!\rangle$ (where $\Phi \overset{\text{df}}{=} \mu X.\langle s \rangle X \vee B$).

(always in a bounded number of steps as stated in theorem 4);

- case $\langle\!\langle \psi, \varphi \rangle\!\rangle(x, y) = \langle\!\langle \neg\varphi, \neg\psi \rangle\!\rangle(y, x)$: this is a shortcut to express the dual rules of the above ones.
- last case: if none of the above rules is applicable, we cannot do any better than to check if (x, env) never verifies ψ or if (y, env) always verifies φ.

Lemma 4: $\langle\!\langle \varphi, \psi \rangle\!\rangle$ is well defined and finitely computable.

The local equivalence is used in section V-B to reduce the size of a module. For this reduction to work as intended, the relation must preserve the truth value of the formula, *i.e.*, we could indifferently use any state from an equivalence class during the verification.

Theorem 2 (preservation of the formula): Let x and y be two states of a module such that $x\langle\!\langle \varphi \rangle\!\rangle y$, then $\mathsf{Pass}^{\varphi}(x) \Leftrightarrow \mathsf{Pass}^{\varphi}(y)$ and $\mathsf{Fail}^{\varphi}(x) \Leftrightarrow \mathsf{Fail}^{\varphi}(y)$.

We also have that this relation is a congruence with respect to the synchronised product of LTS, which enables the use of reduced graphs during the incremental construction of the product with the guarantee that we always work with equivalent graphs.

Theorem 3 (modularity): Let x_1, y_1 be two states of an LTS, x_2, y_2 be two states from another and φ be a formula. If $x_1\langle\!\langle \varphi \rangle\!\rangle y_1$ and $x_2\langle\!\langle \varphi \rangle\!\rangle y_2$ then we have $(x_1, x_2)\langle\!\langle \varphi \rangle\!\rangle(y_1, y_2)$.

V. Reductions

In this section, we define three complementary ways of reducing a module. The goal of these reductions is to build the smallest possible equivalent to one module S, given that we know the formula φ that is to be checked on the global system, as well as the initial state of module S. These reductions are computed without knowledge of the environment and yield a smaller LTS which can be used in place of the initial one by any modular verification techniques.

A. Useless paths pruning

The first step lets us use our knowledge of both the formula and the initial state of the module. It consists in building the LTS S^{φ} as shown in definition 8 which is a product between the module (S) and the formula (φ) and serves three different purposes. Firstly it prunes off the states and the transitions which are not needed in the verification of the formula. For example, if a state is only reachable by using an action which does not appear in the formula, it can be safely removed from the LTS. States which are part of the Pass or Fail set of a sub-formula may also be left out of S^{φ} if we know that they will not bring anything of interest to the formula. Secondly it shows information about which sub-formula may be verified on which state, which is used in the subsequent reductions from subsections V-B and V-C. Lastly, we can use it to know is a state *depends* on another for the computation of a sub-formula which is needed in the definition of the reduction from subsection V-B.

Definition 8 (reachability according to a formula): Let $S = (Q, i, A, R, L)$ be an LTS and φ be a formula. We build LTS $S^{\varphi} = (Q', i', A', R', L')$ from S in which each state is labelled by the formulas that may be verified on it. Below, f is a Boolean function, and $\varphi_1, \ldots, \varphi_k$ are formulas which are either in form $\varphi_j = \langle a \rangle \varphi'_j$ or $\varphi_j = [a]\varphi'_j$. Any formula which is not composed only of fixed points can be represented that way, unfolding it if necessary (this is done implicitly in the following). S^{φ} is defined by:

- the set of states is $Q' \overset{\text{df}}{=} \mathsf{St}(\varphi, i)$, where

$$\mathsf{St}(f(\varphi_1, \ldots, \varphi_k), x) \overset{\text{df}}{=} \{(x, f(\varphi_1 \ldots \varphi_k))\} \cup \bigcup_{j=1\ldots k} \begin{cases} \bigcup_{x \xrightarrow{a} x', Fail^{\varphi'_j}(x') \neq \top} \mathsf{St}(\varphi'_j, x') \\ \qquad \text{if } \varphi_j = \langle l \rangle \varphi'_j \text{ or } \varphi_j = \langle s \rangle \varphi'_j \ , \\ \bigcup_{x \xrightarrow{a} x', Pass^{\varphi'_j}(x') \neq \top} \mathsf{St}(\varphi'_j, x') \\ \qquad \text{if } \varphi_j = [l]\varphi'_j \text{ or } \varphi_j = [s]\varphi'_j \ , \\ \mathsf{St}(\varphi'_j, x) \\ \qquad \text{if } \varphi_j = \langle e \rangle \varphi'_j \text{ or } \varphi_j = [e]\varphi'_j \ ; \end{cases}$$

- the initial state is $i' \overset{\text{df}}{=} (i, \varphi)$;
- $A' \overset{\text{df}}{=} A \cap \{\alpha \mid \alpha \text{ appears in } \varphi\}$
- R' is defined such that if $(x, f(\varphi_1, \ldots, \varphi_k))$ and (x', ψ) belong to St, then $(x, f(\varphi_1, \ldots, \varphi_k)) \xrightarrow{a} (x', \psi) \in R'$ iff $x \xrightarrow{a} x' \in R$ and $\exists j \in 1 \ldots k$ such that $\varphi_j = \langle a \rangle \psi$ or $\varphi_j = [a]\psi$;
- $L' \overset{\text{df}}{=} L \cap \{\lambda \mid \lambda \text{ appears in } \varphi\}$.

Lemma 5: The computation of St terminates.

Graph S^{φ} can be larger than S because each of its state (from Q) is constituted from a state of S associated to a formula. We can however reduce it to a graph that is smaller than S, by removing these formulas using the following operation. If x is a state of a LTS S, we define $\mathsf{Form}^{\varphi}(x) \overset{\text{df}}{=} \{\psi \mid (x, \psi) \in Q\}$.

Definition 9 (reduced reachable graph): Let S be an LTS and φ a formula such that $S^{\varphi} \overset{\text{df}}{=} (Q, i, A, R, L)$. The reachable sub-graph of S with respect to φ is $\mathsf{Reduced}(S^{\varphi}) \overset{\text{df}}{=} (Q', i', A, R', L)$ where:

- $Q' = \{x \mid \mathsf{Form}(x) \neq \emptyset\}$;
- $(i', \varphi) = i$;
- $x \xrightarrow{a} y \in R'$ iff it exists φ_1 and φ_2 such that $(x, \varphi_1) \xrightarrow{a} (y, \varphi_2) \in R$.

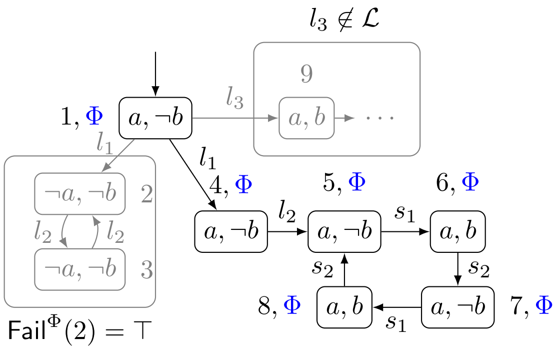

Fig. 5. Formula dependent exploration with pruning of useless paths.

Finally, if we are verifying formula φ on a state where the module S in its initial state, we can use this reduction without losing relevant information with respect to φ.

Theorem 4 (local equivalence preservation): Let S and S' be two LTSs and φ a formula. We write $S\langle\!\langle\varphi\rangle\!\rangle S'$ if the initial states of S and S' are equivalent with respect to $\langle\!\langle\varphi\rangle\!\rangle$. We have both, $S\langle\!\langle\varphi\rangle\!\rangle S^{\varphi}$ and $\mathsf{Reduced}(S^{\varphi})\langle\!\langle\varphi\rangle\!\rangle S^{\varphi}$.

Example 2: figure 5 shows the exploration of an LTS according to formula $\Phi \overset{\text{df}}{=} \mu X.\langle\mathcal{L}\rangle X \vee B$ where $\mathcal{L} \overset{\text{df}}{=} \{l1, l2, s1, s2\}$ and $B \overset{\text{df}}{=} ((a \wedge v) \vee (b \wedge w))$. We know that the verification algorithm will never execute a transition labelled by l_3 because it does not appear in the formula. We thus do not need to keep track of the right side of the graph in order to verify the formula on the global system. Likewise, since $\mathsf{Fail}^{\Phi}(2) = \top$ we can remove action l_1 heading to state 2, and forget about the states 2 and 3, because they will not bring anything more to the computation of the truth value of Φ.

B. Paths cutting

Let x and y be two states of a module such that $x\langle\!\langle\varphi\rangle\!\rangle y$. We have shown that for any state env of the environment, $(x, env) \models \varphi \Leftrightarrow (y, env) \models \varphi$. The idea is to choose the state which will generate the smallest graph reachable by the formula φ (as defined in section V-A). For instance when a formula addresses executions of a system after an initialization phase: the states corresponding to this initial part will be removed. In some cases the graph S^{φ} will be composed of one big strongly connected component (SCC) and this reduction will not be relevant. We show in theorem 5 that we are able to modify the graph S^{φ} by *replacing* a state (x, ψ) by another one (y, ψ) as long as we have $(x, \psi)\langle\!\langle\psi\rangle\!\rangle(y, \psi)$ and if there is no path from (y, ψ) to (x, ψ) in S^{φ}. This *replacement* operation allows us to "jump over" a state. The reduction is thus, for each state (x, ψ) (including but not only the initial one), to replace it if possible by another one (y, ψ) that is equivalent and such that there are fewer states reachable from (x, ψ) than from (y, ψ).

The following replacement operation is defined on LTS which states belong to $Q \times \mathcal{CL}(\varphi)$ and can be iterated on the *reachable graph* from section V-A before applying $\mathsf{Reduced}$.

Definition 10 (state replacement): Let $S^{\psi} \overset{\text{df}}{=} (Q, i, A, R, L)$ be an LTS obtained from definition 9, and (x, φ) and $(y, \varphi) \in Q$. $\mathsf{Replace}(S^{\psi}, (x, \varphi), (y, \varphi))$ is constructed by connecting all the input transitions of (x, φ) to (y, φ) instead. Formally, $\mathsf{Replace}(S^{\psi}, (x, \varphi), (y, \varphi)) \overset{\text{df}}{=} (Q', i', A, R', L)$ where:

- $Q' \overset{\text{df}}{=} Q \setminus (x, \varphi)$;
- $i' \overset{\text{df}}{=} (y, \varphi)$ if $i = (x, \varphi)$, otherwise $i' \overset{\text{df}}{=} i$;
- $R' \overset{\text{df}}{=} (R \setminus \{(x', \varphi') \xrightarrow{a} (x, \varphi)\})$
  $\cup \{(x', \varphi') \xrightarrow{a} (y, \varphi) \mid (x', \varphi') \xrightarrow{a} (x, \varphi) \in R\}$.

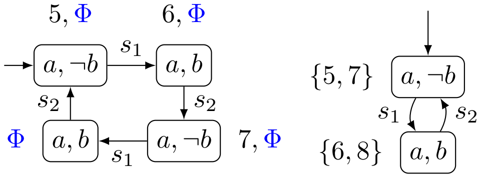

Fig. 6. Path cutting (left) followed by quotienting (right).

The following theorem provides the conditions under which we can safely replace one state by another.

Theorem 5: Let S^{ψ} be an LTS, and (x, φ) and (y, φ) be two states such that $(x, \varphi)\langle\!\langle\varphi\rangle\!\rangle(y, \varphi)$. If there is no path from (y, φ) to (x, φ) in S^{ψ}, then we have $S^{\psi}\langle\!\langle\psi\rangle\!\rangle\mathsf{Replace}(S^{\psi}, (x, \varphi), (y, \varphi))$.

This operation allows us to avoid state (x, φ) and to go directly to (y, φ). This may lead to an important reduction when (x, φ) and subsequent states become unreachable, in which case they can be simply removed.

Example 3: the left LTS from figure 6 is obtained by replacing $(1, \Phi)$ with $(5, \Phi)$ as the initial state in the graph from figure 5. We can do so because all the states here are equivalent with respect to $\langle\!\langle\Phi\rangle\!\rangle$ (1 and 5 in particular), and because there is no path from $(5, \Phi)$ to $(1, \Phi)$. This also allows to remove states 1 and 4 that become no longer reachable.

In this example, we do not show how $(5, \Phi)$ is actually chosen. In general, many choices are possible and it is left to future work to define strategies to make good choices. The prototype used in section VI proceeds by exploring the graph S^{φ} and replacing any new state (x, ψ) by an equivalent one (y, ψ) which is reachable from (x, φ) as follows: it explores the SCC reachable from (x, ψ) in reverse topological order (*i.e.*, from the farther ones to the closer ones) and chooses (y, ψ) in the first SCC where it can be found. This technique is however non-deterministic and there is still room for improvement by better choosing a SCC among the possible ones.

C. Reduction by quotienting

If two states are dependent on each other according to an exploration of the graph we cannot try to delete one of them, but we may be able to merge them in some cases. φ being the property checked on the initial state, we define a formula dependent equivalence relation $\approx^{\varphi}$ which is resilient to the reduction of the graph by quotienting. We start by defining a first relation $\sim^{\psi}$ such that equivalent states are indistinguishable by ψ. We then make use of the LTS S^{φ} form subsection V-A in order to know which sub-formulas may be

$$
\begin{aligned}
\sim^{B} (x,y) &\overset{\text{df}}{=} B@x \Leftrightarrow B@y \\
\sim^{\varphi_1 \wedge \varphi_2} (x,y) &\overset{\text{df}}{=} \sim^{\varphi_1} (x,y) \wedge \sim^{\varphi_2} (x,y) \\
\sim^{\langle l \rangle \varphi} (x,y) &\overset{\text{df}}{=} \mathsf{Fail}^{\langle l \rangle \varphi}(x) \vee \bigwedge_{x \xrightarrow{l} x'} \bigvee_{y \xrightarrow{l} y'} \sim^{\varphi} (x',y') \\
&\qquad \bigwedge \mathsf{Fail}^{\langle l \rangle \varphi}(y) \vee \bigwedge_{y \xrightarrow{l} y'} \bigvee_{x \xrightarrow{l} x'} \sim^{\varphi} (y',x') \\
\sim^{\langle s \rangle \varphi} (x,y) &\overset{\text{df}}{=} \mathsf{Fail}^{\langle s \rangle \varphi}(x) \\
&\qquad \vee \begin{cases} \top & \text{if } \bigwedge_{x \xrightarrow{s} x'} \bigvee_{y \xrightarrow{s} y'} \sim^{\varphi} (x',y') \\ \bot & \text{otherwise} \end{cases} \\
&\qquad \bigwedge \mathsf{Fail}^{\langle s \rangle \varphi}(y) \\
&\qquad \vee \begin{cases} \top & \text{if } \bigwedge_{y \xrightarrow{s} y'} \bigvee_{x \xrightarrow{s} x'} \sim^{\varphi} (y',x') \\ \bot & \text{otherwise} \end{cases} \\
\sim^{\langle e \rangle \varphi} (x,y) &\overset{\text{df}}{=} \begin{cases} \top & \text{if } \sim^{\varphi} (x,y) \\ \bot & \text{otherwise} \end{cases} \\
\sim^{\mu X.\varphi} (x,y) &\overset{\text{df}}{=} \bigvee \{ f(x,y) \mid \sim^{\varphi} f \overset{Q^2}{\Leftarrow} f \} \\
\sim^{\neg \varphi} (x,y) &\overset{\text{df}}{=} \sim^{\varphi} (x,y)
\end{aligned}
$$

Fig. 7. Definition of the merging equivalence relation

verified on which state. This allows us to merge states which are equivalent (according to $\sim$) on at least these sub-formulas, whereas without the information form graph S^{φ} we would need equivalency on every possible sub-formula.

For this purpose, given an LTS S, two states x, y, and a formula φ, we define the function $\sim_S^{\varphi}$ as shown in figure 7 and write $x \sim^{\varphi} y$ iff $\sim^{\varphi} (x,y) = \top$.

Lemma 6: For any formula φ, $\sim_s^{\varphi}$ is well defined and is an equivalence relation.

The reduction of the graph is made by merging the states which are equivalent with respect to all the formulas that may be verified on this state.

Definition 11: For any LTS S and formula ψ, we write $x \approx^{\psi} y$ iff $\mathsf{Form}^{\psi}(x) = \mathsf{Form}^{\psi}(y)$ and $x \sim^{\varphi} y$ for any $\varphi \in \mathsf{Form}^{\psi}(x)$.

We then build the reduced graph by quotienting according to $\approx^{\psi}$, which is quite a classical construction.

Definition 12: Let $S \overset{\text{df}}{=} (Q, i, A, R, L)$ be an LTS with $A \overset{\text{df}}{=} A^{\mathsf{loc}} \uplus A^{\mathsf{fus}}$, and φ be a formula. The *reduction of S with respect to φ*, is the LTS (Q', i', A', R', L') such that:

- $Q' \overset{\text{df}}{=} Q/{\approx^{\varphi}}$ is the quotient set of Q by $\approx^{\varphi}$;
- $i' \overset{\text{df}}{=} [i]$ is the equivalence class containing i;
- A' is exactly A and is partitioned the same way;
- $R' \overset{\text{df}}{=} \{(c_1, a, c_2) \in Q' \times A' \times Q' \mid \exists q_1 \in c_1, \exists q_2 \in c_2, (q_1, a, q_2) \in R\}$;
- $L'(c) \overset{\text{df}}{=} \bigvee_{q \in [c]} L(q)$.

Similarly to before, the next three theorems ensure that this reduction operation preserves the truth value of the formula at a global level.

Theorem 6 (compositionality): Let x_1, y_1 be two states of S_1, x_2, y_2 be two states of S_2, and φ be a formula. If $x_1 \approx^{\varphi} y_1$ and $x_2 \approx^{\varphi} y_2$ then $(x_1, x_2) \approx^{\varphi} (y_1, y_2)$.

*Theorem 7 (*Pass *and* Fail *preservation):* Let x and y be two states of S, and ψ be a formula. If $x \approx^{\psi} y$ then for any $\varphi \in \mathsf{Form}^{\psi}(x)$, $\mathsf{Pass}^{\varphi}(x) \Leftrightarrow \mathsf{Pass}^{\varphi}(y)$ and $\mathsf{Fail}^{\varphi}(x) \Leftrightarrow \mathsf{Fail}^{\varphi}(y)$.

Theorem 8 (equivalence preservation): For all state x and formula ψ, $x \approx^{\psi} [x]$.

Example 4: The right LTS of figure 6 is obtained from the left one by applying the quotienting with respect to $\approx^{\Phi}$ (Φ being defined in example 2).

VI. Experiments

A prototype implementation of the method presented above is currently in progress. We report here limited experiments about the efficiency of the combined reductions from subsections V-A and V-B (the third reduction is not currently implemented), referred to as SKIP, on different kinds of models and formulas. This is far from a rigorous benchmarks but this already shows the potential of our approach compared with a generic reduction consisting of quotienting by the safety equivalence after having replaced as many actions as possible by τ-transitions (hide them) [4], referred to as SAFETY.

We have considered a toy example representing a mutual exclusion system, expressed as the modular Petri net depicted in figure 8 which provides us with a way of generating state spaces of different sizes by changing the initial number of token in place i. Even though modular Petri nets are not formally defined in this paper, they are suitable candidates to apply our techniques because their behaviour is defined exactly as the semantics of modular LTS from definition 2: each module is a Petri net whose semantics is a LTS, the semantics of the whole system is obtained as the synchronous product of the semantics of its modules [5].

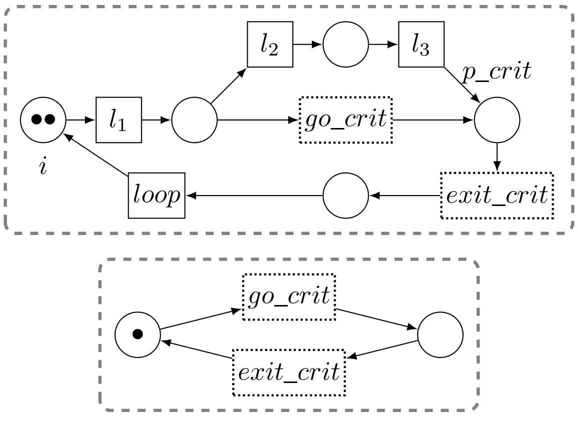

Fig. 8. Modular Petri net modeling a mutual exclusion system, each module is surrounded by a dashed box, fused transitions are draw with dotted borders.

We have considered the reduction of the behavior of one module (that is drawn above in figure 8) obtained by SKIP considering formulas that preserve some safety and reachability properties so that a comparison with SAFETY makes sense. We thus have $A \overset{\text{df}}{=} \{l_1, l_2, l_3, go\_crit, exit\_crit\}$ the set of all transitions (setting aside $loop$), and define p a propositional constant that is true if and only if there is strictly more than one token in place p_crit. We also defined H^{φ} as the maximal set of actions that it is possible to hide before quotienting the module by the safety equivalence. We have considered three formulas presented now on, the first one is a counter example where we obtain no reduction from our method; the second one is a case where SKIP is dramatically efficient, reducing the LTS to only one state; the third one is more balanced. The corresponding results are displayed in figure 9.

$$\varphi_1 \overset{\text{df}}{=} \mu X(\langle go\_crit \rangle true \vee \langle A \cup \{loop\} \rangle X) \tag{1}$$

This formula means that we can reach a state were go_crit is a possible action. φ_1 checked on this particular graph will not benefit from the SKIP reduction for two reasons. Firstly, the product graph $S_i^{\varphi_1}$ is composed of only one SCC so SKIP will never be able to find a suitable replacement for a state. Secondly no state of the graph is part of either $Pass^{\varphi_1}$ or $Fail^{\varphi_1}$, so they all have to be included in the product graph.

In order to obtain an example on which it is relevant to apply reduction SKIP, we now consider a system where transition $loop$ and the corresponding actions have been removed.

$$\varphi_2 \overset{\text{df}}{=} \mu X(p \vee \langle A \rangle X) \tag{2}$$

Formula φ_2 means that a state with at least two tokens in place p_crit is reachable. This property will be preserved by the safety equivalence if we hide every local actions, so $H^{\varphi_2} \overset{\text{df}}{=} \{l_1, l_2, l_3\}$. As a matter of fact it is always possible to conclude locally about the truth value of φ_2. This is why the SKIP reduction always returns a LTS composed of only one state. In the case where there is only one token in i initially, the initial state belongs to $\mathsf{Fail}^{\varphi_2}$ because it is never possible to have two tokens in p_{crit} in the future. So the module can be reduced to only one state that satisfies $\neg p$. In the cases where they are at least two tokens in i initially, it is possible to place two tokens in p_{crit} by using only local transitions, so the initial state belongs to $\mathsf{Pass}^{\varphi_2}$ and the module is equivalent to the one composed of only one state that satisfies p.

$$\varphi_3 \overset{\text{df}}{=} \nu Y(\neg p \wedge [A \setminus l_2]Y) \tag{3}$$

Formula φ_3 means that no path which does not contain l_2 leads to a state verifying p. For this property we cannot hide action l_2 because we need to be able to differentiate it from the other local actions, so we have $H^{\varphi_3} \overset{\text{df}}{=} \{l_1, l_3\}$. This local action is however ignored when computing the product between the graph and the formula as well as the equivalence relation. If there is only one token in the net, then the initial state belongs to $\mathsf{Pass}^{\varphi_3}$ and the graph is reduced to a single state. In the other cases, we have to keep track of all the synchronized transitions. As mentioned before finding the best replacement for a state among the equivalent ones is not trivial and the algorithm used in this prototype is only a simple heuristic.

Number of initial tokens		1	2	5	10
Full LTS (no reduction)		8	18	128	1003
φ_1	SAFETY	4	7	22	67
	SKIP	8	18	128	1003
φ_2	SAFETY	4	7	22	67
	SKIP	1	1	1	1
φ_3	SAFETY	5	11	57	287
	SKIP	1	4	28	102

Fig. 9. Number of states depending on formulas, number of initial tokens, and reductions used.

Even if on this example SKIP yields a better reduction than SAFETY, it may be a good choice in practice to combine both reductions. Indeed, SAFETY is a generic method that is likely to have a lower complexity than SKIP (and to be more efficiently implementable). So SKIP could be applied quicker on a LTS already reduced by SAFETY. This holds on this example but also in general, in particular, with formula φ_1, chaining both reductions allows to gain the reduction from SAFETY even when SKIP yields no reduction at all.

VII. Conclusion and perspectives

We have presented a formal framework to perform the model-checking of μ-calculus formula on modular transitions systems. This framework allows for: (1) separate analysis of modules, possibly yielding a conclusion on the truth value of the formula for the complete system; (2) abstraction of the analysed modules that do not allow to conclude, using the information collected during analysis; (3) incremental recombination of the abstracted modules, the result being again candidate to analysis, possibly yielding a conclusion or allowing for further abstraction; (4) incremental bottom-up analysis with hierarchical abstraction at each step, with a possibility to conclude before the whole system is analyzed.

Within this framework, we now intend to explore strategies to obtain the best possible reductions. We have indeed identified a number of places where various choices are possible, for instance as discussed in section V-B. More generally, the order in which modules are considered and then recombined should widely influence the overall performance. Indeed, a modular system is a set of modules that can be analysed bottom-up in any order, and recombined arbitrarily. The need arises to develop strategies to always chose the best order, or at least a good one, or to avoid the worst ones. A good order should allow to: (1) conclude earlier, avoiding the analysis of most modules; (2) reduce efficiently the analysed modules; (3) limit combinatorial explosion when abstracted subsystems are recombined [6], [7].

Defining strategies to guarantee a good order is a difficult problem, searching for optimal orders is far more difficult. We intend to proceed by first finishing our prototype in order to be able to run a variety of case studies to validate our approach in practice. We intend then to exploit the results of these case studies to identifies sources of efficiencies and inefficiencies so that we will be able to define good or bad ways to put hierarchical abstraction into practice, which should lead to strategies that will be progressively refined with the aim to introduce performance guarantees.

Another future work is to define precisely how a counter-example can be constructed when our analysis concludes that a formula is not satisfied. This possibility to exhibit a trace is a key feature of model-checking that we consider as critical to preserve.

A. Related works

While modular model-checking is quite a well explored approach [8], [9], [10], [11], [5], [12], it is not always amened to hierarchical analysis. Indeed, in many cases, the recombination of modules is often an object of a different nature than the modules themselves. For example, in [13], modules are combined using a synchronisation graph that cannot itself be recombined further. Approaches proposing hierarchical abstraction are usually based on a generic reduction of the modules preserving a full class of properties. Some for instance focus on reachability [10], [14], [15] or safety [4]; others preserve all the formulas from a given logic [16], [8], [17], [18]. In [19], hierarchical abstraction is proposed using a generic reduction that hides and collapses sequences of local transitions in the modules before they are recombined; model-checking is then performed on-the-fly using the CADP toolbox [20].

To the best of our knowledge, formula-dependent abstractions are always approached through a reduction that preserves a fragment of a logic. Either, this fragment is predetermined before the analysis, or it is determined according to the formula of interest [21], [22]. This differs from our proposal where only one formula is preserved by the reductions. Hierarchical abstraction preserving only one formula was used in [23], in which a preliminary version of the current work has been proposed, namely the “reduction by quotienting” presented here in section V-C; moreover, the current version has been completely reworked, with a slight generalisation and a wide simplification of the definitions and proofs.

In [24], the authors propose a compositional verification technique which shows similarities with our work. Given a μ-calculus formula, they are able to locally study some part of a Kripke modal transition system with the use of a 3-valued model checking game. Setting aside the differences due to the models, the local exploration techniques are similar and the game graph resembles the product graph from section V-A. (More generally, building a product between a LTS and a formula is quite a classical method in the field of model-checking.) However, our approach and that from [24] differ in their goals and actually serve complementary purposes. In [24], a module is viewed as an abstraction of the global system that be studied locally with the help of the game graph. Information about the environment is then injected in the form of a refinement of the system in the parts where partial analysis yields an indefinite result. This results in a compositional method that ends with the validation/invalidation of the formula. In the present work however, the local reductions yield an object of the same nature as the input module and does so without information about the rest of the system. As such, our reductions can be used in conjunction with another compositional verification method.

Partial analysis is also encountered in the literature as a dual method for the analysis of modular systems. For example, in [25], [26], [27], a formula is combined with a module, yielding a (possibly big) new formula that the rest of the system has to validate. It is then possible to apply this principle incrementally by considering one module after the other.

References

[1] D. Kozen, “Results on the propositional μ-calculus,” *Theoretical Computer Science*, vol. 27, no. 3, pp. 333–354, 1983.

[2] C. Stirling, *Modal and temporal properties of processes*. Springer, 2001.

[3] A. Tarski, “A lattice-theoretical fixpoint theorem and its applications,” *Pacific journal of Mathematics*, vol. 5, no. 2, pp. 285–309, 1955.

[4] A. Bouajjani, J.-C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis, “Safety for branching time semantics,” in *Automata, Languages and Programming*. Springer, 1991, pp. 76–92.

[5] S. Christensen and L. Petrucci, “Modular analysis of Petri nets,” *The Computer Journal*, vol. 43, no. 3, pp. 224–242, 2000.

[6] P. Crouzen and H. Hermanns, “Aggregation ordering for massively compositional models,” in *Application of Concurrency to System Design (ACSD), 2010 10th International Conference on*. IEEE, 2010, pp. 171–180.

[7] P. Crouzen and F. Lang, “Smart reduction,” in *Fundamental Approaches to Software Engineering*. Springer, 2011, pp. 111–126.

[8] R. Milner, *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[9] S. Graf and B. Steffen, “Compositional minimization of finite state systems,” in *Computer-Aided Verification*. Springer, 1991, pp. 186–196.

[10] A. Valmari, *Compositional state space generation*. Springer, 1993.

[11] J.-P. Krimm and L. Mounier, “Compositional state space generation from lotos programs,” in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 1997, pp. 239–258.

[12] F. Lang, “Refined interfaces for compositional verification,” in *Formal Techniques for Networked and Distributed Systems-FORTE 2006*. Springer, 2006, pp. 159–174.

[13] C. Lakos and L. Petrucci, “Modular analysis of systems composed of semiautonomous subsystems,” in *proc. of ACSD’04*. IEEE Computer Society Press, 2004.

[14] K.-C. Tai and P. V. Koppol, “An incremental approach to reachability analysis of distributed programs,” in *Software Specification and Design, 1993., Proceedings of the Seventh International Workshop on*. IEEE, 1993, pp. 141–150.

[15] D. Giannakopoulou, J. Kramer, and S. C. Cheung, “Behaviour analysis of distributed systems using the tracta approach,” *Automated Software Engineering*, vol. 6, no. 1, pp. 7–35, 1999.

[16] K. Klai, L. Petrucci, and M. Reniers, “An incremental and modular technique for checking ltl\ x properties of petri nets,” *Formal Techniques for Networked and Distributed Systems–FORTE 2007*, pp. 280–295, 2007.

[17] D. Dams, O. Grumberg, and R. Gerth, “Generation of reduced models for checking fragments of ctl,” in *Computer Aided Verification*. Springer, 1993, pp. 479–490.

[18] R. J. Van Glabbeek and W. P. Weijland, “Branching time and abstraction in bisimulation semantics,” *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 555–600, 1996.

[19] N. D. Mendes, F. Lang, Y.-S. Le Cornec, R. Mateescu, G. Batt, and C. Chaouiya, “Composition and abstraction of logical regulatory modules: application to multicellular systems,” *Bioinformatics*, vol. 29, no. 6, pp. 749–757, 2013.

[20] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, “Cadp 2010: a toolbox for the construction and analysis of distributed processes,” in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 372–387.

[21] R. Barbuti, N. De Francesco, A. Santone, and G. Vaglini, “Selective mu-calculus and formula-based equivalence of transition systems,” *Journal of Computer and System Sciences*, vol. 59, no. 3, pp. 537–556, 1999.

[22] R. Mateescu and A. Wijs, “Property-dependent reductions for the modal mu-calculus,” in *Model Checking Software*. Springer, 2011, pp. 2–19.

[23] Y.-S. Le Cornec, “Compositional analysis of modular petri nets using hierarchical state space abstraction,” *Joint Proceedings of:» LAM’12 «*, p. 119, 2012.

[24] S. Shoham and O. Grumberg, “Compositional verification and 3-valued abstractions join forces,” in *Static Analysis*. Springer, 2007, pp. 69–86.

[25] H. R. Andersen, “Partial model checking,” in *Logic in Computer Science, 1995. LICS’95. Proceedings., Tenth Annual IEEE Symposium on*. IEEE, 1995, pp. 398–407.

[26] H. R. Andersen and J. Lind-Nielsen, “Partial model checking of modal equations: A survey,” *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 3, pp. 242–259, 1999.

[27] F. Lang and R. Mateescu, “Partial model checking using networks of labelled transition systems and boolean equation systems,” in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2012, pp. 141–156.

Appendix

The following proofs are most of the time done by induction on the structure of the formulas. They are not exhaustive because we choose to only show one case when faced with a set of very similar ones. We also do not deal directly with the cases of fixed-points, because as shown in the proofs A and F, any formula can be replaced by an equivalent fixed-point free formula on a given finite model.

A. Lemma 1: the computation of Pass *terminates*

Let us consider a finite set of variables $\mathcal{V}$ and a set Q of states. The set of functions from Q to $\mathbb{B}(\mathcal{V})$ is a complete lattice with respect to $\overset{Q}{\Rightarrow}$, as long as we make it finite by choosing one canonical representative for each set of equivalent formulas (for instance the disjunctive normal form). For any formula $\Phi = \mu.X\varphi$, the only negation operators in Pass^{φ} can arise form the base case. This means that Pass^{φ} is monotonous in X and we can use the Knaster-Tarski theorem to show that the least and greatest fixed point are well defined. A corollary often used in the following proofs, is that for a given model, there exists $n \in \mathbb{N}$ such that $\mathsf{Pass}^{\mu X\varphi} = \mathsf{Pass}^{\varphi^n(\bot)}$ ($\mathsf{Pass}^{\nu X\varphi} = \mathsf{Pass}^{\varphi^n(\top)}$).

B. Lemma 2: Pass *is compositional*

Let us show that $\mathsf{Pass}^{\varphi}(x_1)@x_2 \Rightarrow \mathsf{Pass}^{\varphi}(x_1, x_2)$.
The property for Fail follows because we have $\mathsf{Fail}^{\varphi} = \mathsf{Pass}^{\neg\varphi}$.

Case B: $B@x_1@x_2$ is equal to $B@(x_1, x_2)$.

Case $\varphi_1 \wedge \varphi_2$: By applying the induction hypothesis, we know the following. $\mathsf{Pass}^{\varphi_1}(x_1)@x_2 \wedge \mathsf{Pass}^{\varphi_2}(x_1)@x_2$ implies $\mathsf{Pass}^{\varphi_1}((x_1, x_2)) \wedge \mathsf{Pass}^{\varphi_2}((x_1, x_2))$, which is equal to $\mathsf{Pass}^{\varphi_1 \wedge \varphi_2}((x_1, x_2))$ by definition.

Case $\langle l\rangle\varphi$: Similarly, $\bigvee_{x_1 \xrightarrow{l} x_1'} \mathsf{Pass}^{\varphi}(x_1')@x_2$ implies $\bigvee_{x_1 \xrightarrow{l} x_1'} \mathsf{Pass}^{\varphi}((x_1', x_2))$.
Which is the same as $\bigvee_{(x_1,x_2) \xrightarrow{l} (x_1,x_2)'} \mathsf{Pass}^{\varphi}((x_1, x_2)')$ since l is an action which is local to module 1.

Cases $\langle s\rangle\varphi$ and $\langle e\rangle\varphi$: In these cases Pass is equal to $\bot$ so the implication holds.

Case $[e]\varphi$: Here we can directly apply the induction hypothesis.

C. Theorem 1: relation with the μ-calculus semantics

This proof is based on the fact that if a formula φ only contains local variables, then $\mathsf{Pass}^{\varphi}(x)$ can only evaluate to $\top$ or $\bot$.

Case B: If $B@x = \top$ then x belongs to B^S, otherwise we have $B@x = \bot$ and x is in $Q \setminus B^S$.

Case $\varphi_1 \vee \varphi_2$: $\mathsf{Pass}^{\varphi_1 \vee \varphi_2}(x) = \mathsf{Pass}^{\varphi_1}(x) \vee \mathsf{Pass}^{\varphi_2}(x)$. If this is equal to $\top$, we have $x \in \varphi_1^S \cup \varphi_2^S$ with the induction hypothesis, *i.e.* $x \in (\varphi_1 \vee \varphi_2)^S$.

Case $\langle l\rangle\varphi$: $\mathsf{Pass}^{\langle l\rangle\varphi}(x) = \bigvee_{x \xrightarrow{l} x'}(\mathsf{Pass}^{\varphi}(x'))$ so $\mathsf{Pass}^{\langle l\rangle\varphi}(x) = \top$ if and only if exists $x \xrightarrow{l} x'$ such that $x' \in \varphi^S$.

Case $\mu X\varphi$: For any $n \in \mathbb{N}, \mathsf{Pass}^{n.X\varphi}(x) = \top$ if and only if $x \in (n.X\varphi)^S$.

D. Lemma 3: Pass *and* Fail *are correct*

Let us show that for any $x \in S$ and $env \in Env$ then $env \models \mathsf{Pass}^{\varphi}(x) \Rightarrow (x, env) \models \varphi$. The result about Fail follows because Fail^{φ} is defined as $\mathsf{Pass}^{\neg\varphi}$ and $(x, env) \not\models \varphi$ is equivalent to $(x, env) \models \neg\varphi$. The property is proven by induction on the structure of the formula.

Case $\varphi = B$: If env $\models B@x$ then $(x, env) \models B$.

Case $\varphi = \varphi_1 \wedge \varphi_2$: We have $env \models \mathsf{Pass}^{\varphi_1}(x)$ and $env \models \mathsf{Pass}^{\varphi_2}(x)$. The IH allows us to deduce $(x, env) \models \varphi_1$ and $(x, env) \models \varphi_2$, *i.e.* $(x, env) \models \varphi_1 \wedge \varphi_2$.

Case $\varphi = \langle l\rangle\psi$: $\mathsf{Pass}^{\varphi}(x) = \bigvee_{x'} \mathsf{Pass}^{\psi}(x')$. With the IH, we know that exists x' such that $(x, env) \models \psi$. From which we deduce $(env, x) \models \varphi$.

Case $\varphi = [s]\psi$: We suppose that for every $x \xrightarrow{s} x'$ we have $\mathsf{Pass}^{\psi}(x') = \top$. So for every $(x, env) \xrightarrow{s} (x, env)'$ we have $(x, env)' \models \psi$ *i.e.* $(x, env) \models \varphi$.

Case $\varphi = \mu X\psi$: for any $n \in \mathbb{N}, env \models \mathsf{Pass}^{nX.\psi}(x) \Rightarrow (x, env) \models nX.\psi$.

E. Lemma 4: $\langle\langle\varphi, \psi\rangle\rangle$ is effectively computable

The proof is similar to lemma 1. The difference is that we consider functions from Q^2 to $\mathbb{B}(\mathcal{V})$. For the case $\langle\langle\Phi, \psi\rangle\rangle$ where $\Phi = \sigma X\varphi$, since the rules from figure 3 only build formulas which contain no negation, the function $\langle\langle\varphi, \psi\rangle\rangle : (Q^2 \to \mathbb{B}(\mathcal{V})) \to (Q^2 \to \mathbb{B}(\mathcal{V}))$ is monotonous in the complete lattice $(Q^2 \to \mathbb{B}(\mathcal{V}), \overset{Q^2}{\Rightarrow})$.

F. Unfolding

From lemma 4 we can deduce that for a model S and two formulas φ, ψ, there thus exists two unfolded formulas φ', ψ' which do not contain fixed-point terms, such that $\langle\langle\varphi, \psi\rangle\rangle = \langle\langle\varphi', \psi'\rangle\rangle$. Moreover for a fixed-point formula $\mu.X\varphi$ (resp.$\nu X\varphi$), we can find $n \in \mathbb{N}$ such that $\langle\langle\mu X\varphi, \psi\rangle\rangle = \langle\langle\varphi^n(\bot), \psi\rangle\rangle$ (resp. $\langle\langle\nu X\varphi, \psi\rangle\rangle = \langle\langle\varphi^n(\top), \psi\rangle\rangle$), where n corresponds to the maximum number of unfolding of φ in the unfolded formula. This property allows us to only prove the various theorems on formulas which does not contain fixed-point subformulas.

G. Transitivity of $\langle\langle\varphi\rangle\rangle$

Let us prove that if we have $\langle\langle\varphi, \psi\rangle\rangle(x, y)$ and $\langle\langle\psi, \xi\rangle\rangle(y, z)$ then we also have $\langle\langle\varphi, \xi\rangle\rangle(x, z)$.

Base case: We have $\langle\langle\varphi, \psi\rangle\rangle(x, y) = \mathsf{Fail}^{\varphi}(x) \vee \mathsf{Pass}^{\psi}(y)$ and $\langle\langle\psi, \xi\rangle\rangle(y, z) = \mathsf{Fail}^{\psi}(x) \vee \mathsf{Pass}^{\xi}(z)$. If we have $\mathsf{Fail}^{\varphi}(x)$ then $\langle\langle\varphi, \xi\rangle\rangle(x, z)$ holds. If $\mathsf{Pass}^{\psi}(y)$ is verified then $\mathsf{Fail}^{\psi}(y)$ is not possible. This means that we have $\mathsf{Pass}^{\xi}(z)$ which implies $\langle\langle\varphi, \xi\rangle\rangle(x, z)$. Note that case in fact includes the case B_1, B_2, which only appears in the definition for the sake of clarity.

Case $\langle\langle\varphi_1 \wedge \varphi_2, \psi\rangle\rangle$ and $\langle\langle\psi, \xi\rangle\rangle$: We have either $\langle\langle\varphi_1, \psi\rangle\rangle(x, y) \wedge \langle\langle\psi, \xi\rangle\rangle(y, z)$ or $\langle\langle\varphi_2, \psi\rangle\rangle(x, y) \wedge \langle\langle\psi, \xi\rangle\rangle(y, z)@y_1$. Which gives us $\langle\langle\varphi_1 \wedge \varphi_2, \xi\rangle\rangle(x, z)$ using the induction hypothesis.

Case $\langle\langle\varphi, \psi_1 \wedge \psi_2\rangle\rangle$ and $\langle\langle\psi_1 \wedge \psi_2, \xi\rangle\rangle$: We have $\langle\langle\varphi, \psi_1\rangle\rangle(x, y)$,$\langle\langle\varphi, \psi_2\rangle\rangle(x, y)$ and $\langle\langle\psi_1, \xi\rangle\rangle(y, z) \vee \langle\langle\psi_2, \xi\rangle\rangle(y, z)$.If $\langle\langle\psi_1, \xi\rangle\rangle(y, z)$ holds then can deduce
$\langle\langle\varphi, \xi\rangle\rangle(x, z)$. Similarly if we have $\langle\langle\psi_2, \xi\rangle\rangle(y, z)$.

Case $\langle\langle\langle l\rangle\varphi', \psi\rangle\rangle$ and $\langle\langle\psi, \xi\rangle\rangle$:

Suppose we have $\bigwedge\limits_{x \xrightarrow{l} x'} \langle\langle\varphi', \psi\rangle\rangle(x', y)$ and $\langle\langle\psi, \xi\rangle\rangle(y, z)$.
The induction hypothesis gives us:
$\bigwedge\limits_{x \xrightarrow{l} x'} \langle\langle\varphi', \xi\rangle\rangle(x', z) = \langle\langle\langle l\rangle\varphi', \xi\rangle\rangle(x, z)$.

Case $\langle\langle\varphi, \langle l\rangle\psi'\rangle\rangle$ and $\langle\langle\langle l\rangle\psi', \xi\rangle\rangle$:

Suppose we have $\bigvee\limits_{y \xrightarrow{l} y'} \langle\langle\varphi, \psi'\rangle\rangle(x, y')$ and $\bigwedge\limits_{y \xrightarrow{l} y'} \langle\langle\psi', \xi\rangle\rangle(y', z)$.
This implies: $\bigvee\limits_{y \xrightarrow{l} y'} \langle\langle\varphi, \xi\rangle\rangle(x, z) = \langle\langle\varphi, \xi\rangle\rangle(x, z)$.

Case $\langle\!\langle \langle s\rangle\varphi', \langle s\rangle\psi' \rangle\!\rangle$ and $\langle\!\langle \langle s\rangle\psi', \langle s\rangle\xi' \rangle\!\rangle$:

$$\langle\!\langle \varphi, \psi \rangle\!\rangle(x,y) = \begin{cases} \top & \text{if } \bigwedge_{x \xrightarrow{s} x'} \bigvee_{y \xrightarrow{s} y'} \langle\!\langle \varphi', \psi' \rangle\!\rangle(x',y') \\ \bot & \text{otherwise} \end{cases}$$

$$\langle\!\langle \psi, \xi \rangle\!\rangle(y,z) = \begin{cases} \top & \text{if } \bigwedge_{y \xrightarrow{s} y'} \bigvee_{z \xrightarrow{s} z'} \langle\!\langle \psi', \xi' \rangle\!\rangle(y',z') \\ \bot & \text{otherwise} \end{cases}$$

We have for any x', $\displaystyle\bigvee_{y \xrightarrow{s} y'} \langle\!\langle \varphi', \psi' \rangle\!\rangle(x',y') = \top$.

We also know that for any y', $\displaystyle\bigvee_{z \xrightarrow{s} z'} \langle\!\langle \psi', \xi' \rangle\!\rangle(y',z') = \top$ holds so:

$$\text{for any } x',\ \bigvee_{y \xrightarrow{s} y'} \Big(\langle\!\langle \varphi', \psi' \rangle\!\rangle(x',y') \wedge \bigvee_{z \xrightarrow{s} z'} \langle\!\langle \psi', \xi' \rangle\!\rangle(y',z') \Big) = \top$$

$$\Leftrightarrow \bigvee_{y \xrightarrow{s} y'} \bigvee_{z \xrightarrow{s} z'} \langle\!\langle \varphi', \psi' \rangle\!\rangle(x',y') \wedge \langle\!\langle \psi', \xi' \rangle\!\rangle(y',z') = \top$$

$$\Rightarrow \bigvee_{y \xrightarrow{s} y'} \bigvee_{z \xrightarrow{s} z'} \langle\!\langle \varphi', \xi' \rangle\!\rangle(x',z') = \top$$

$$\Rightarrow \bigvee_{z \xrightarrow{s} z'} \langle\!\langle \varphi', \xi' \rangle\!\rangle(x',z') = \top.$$

H. Theorem 2: preservation of the formula by $\langle\!\langle \varphi \rangle\!\rangle$

Let us show that $\langle\!\langle \psi, \varphi \rangle\!\rangle(x,y) \Rightarrow (\mathsf{Pass}^{\psi}(x) \Rightarrow \mathsf{Pass}^{\varphi}(y))$. The result for Fail follows because we have $\mathsf{Fail}^{\varphi} = \mathsf{Pass}^{\neg\varphi}$ and $\langle\!\langle \psi, \varphi \rangle\!\rangle(x,y) = \langle\!\langle \neg\varphi, \neg\psi \rangle\!\rangle(x,y)$. We define the following rules for the construction of $\langle\!\langle \varphi, \psi \rangle\!\rangle^P$. This is another way of computing the formula $\mathsf{Pass}^{\varphi}(x) \Rightarrow \mathsf{Pass}^{\psi}(y)$. This reformulation is meant to be compared to the definition of $\langle\!\langle \varphi, \psi \rangle\!\rangle$ more easily.

$$\langle\!\langle B_1, B_2 \rangle\!\rangle^P(x,y) = B_1@x \Rightarrow B_2@y$$

$$\langle\!\langle \psi, \varphi_1 \vee \varphi_2 \rangle\!\rangle^P(x,y) = \langle\!\langle \psi, \varphi_1 \rangle\!\rangle^P(x,y) \vee \langle\!\langle \psi, \varphi_2 \rangle\!\rangle^P(x,y)$$

$$\langle\!\langle \psi, \langle l\rangle\varphi \rangle\!\rangle^P(x,y) = \neg\mathsf{Pass}^{\psi}(x) \vee \bigvee_{y \xrightarrow{l} y'} \langle\!\langle \psi, \varphi \rangle\!\rangle^P(x,y')$$

$$\langle\!\langle \langle s\rangle\psi, \langle s\rangle\varphi \rangle\!\rangle^P(x,y) = \top$$

$$\langle\!\langle \langle e\rangle\psi, \langle e\rangle\varphi \rangle\!\rangle^P(x,y) = \top$$

$$\langle\!\langle \mu X.\psi, \varphi \rangle\!\rangle^P(x,y) = \bigvee\{f(x,y) \mid \langle\!\langle \psi, \varphi \rangle\!\rangle^P f \overset{Q^2}{\Leftarrow} f\}$$

$$\langle\!\langle \psi, \varphi \rangle\!\rangle^P(x,y) = \langle\!\langle \neg\varphi, \neg\psi \rangle\!\rangle^P(y,x)$$

$$\langle\!\langle \nu X.\psi, \varphi \rangle\!\rangle^P(x,y) = \bigwedge\{f(x,y) \mid \langle\!\langle \psi, \varphi \rangle\!\rangle^P f \overset{Q^2}{\Rightarrow} f\}$$

$$\langle\!\langle \psi, \varphi_1 \wedge \varphi_2 \rangle\!\rangle^P(x,y) = \langle\!\langle \psi, \varphi_1 \rangle\!\rangle^P(x,y) \wedge \langle\!\langle \psi, \varphi_2 \rangle\!\rangle^P(x,y)$$

$$\langle\!\langle \psi, [l]\varphi \rangle\!\rangle^P(x,y) = \bigwedge_{y \xrightarrow{l} y'} \langle\!\langle \psi, \varphi \rangle\!\rangle^P(x,y')$$

$$\langle\!\langle \psi, \varphi \rangle\!\rangle^P(x,y) = \mathsf{Pass}^{\psi}(x) \Rightarrow \mathsf{Pass}^{\varphi}(y)$$

By comparing the rules and because of the monotony of our expressions, we see that $\langle\!\langle \varphi, \psi \rangle\!\rangle \overset{Q}{\Rightarrow} \langle\!\langle \varphi, \psi \rangle\!\rangle^P$.

I. Theorem 3: modularity of $\langle\!\langle \varphi \rangle\!\rangle$

Let us show that if we have $\langle\!\langle \varphi, \psi \rangle\!\rangle(x_1, y_1)@x_2$ and $\langle\!\langle \psi, \xi \rangle\!\rangle(x_2, y_2)@y_1$
then $\langle\!\langle \varphi, \xi \rangle\!\rangle((x_1, x_2), (y_1, y_2))$ holds. From there we can obtain that $x_1 \langle\!\langle \varphi \rangle\!\rangle y_1$ and $x_2 \langle\!\langle \varphi \rangle\!\rangle y_2$ implies $(x_1, y_1) \langle\!\langle \varphi \rangle\!\rangle (x_2, y_2)$.

Base case:

$$\langle\!\langle \varphi, \psi \rangle\!\rangle(x_1, y_1) = \mathsf{Fail}^{\varphi}(x_1) \vee \mathsf{Pass}^{\psi}(y_1)$$

and $\langle\!\langle \psi, \xi \rangle\!\rangle(x_2, y_2) = \mathsf{Fail}^{\psi}(x_2) \vee \mathsf{Pass}^{\xi}(y_2)$.

If we have $\mathsf{Fail}^{\varphi}(x_1)@x_2$ then $\mathsf{Fail}^{\varphi}(x_1, x_2)$ is verified,
which implies $\langle\langle\varphi, \xi\rangle\rangle((x_1, x_2), (y_1, y_2))$. If $\mathsf{Pass}^{\psi}(y_1)@x_2$ then $\mathsf{Fail}^{\psi}(x_2)@y_1$ is not possible. This means that we have $\mathsf{Pass}^{\xi}(y_2)@y_1$, then $\mathsf{Pass}^{\xi}(y_1, y_2)$
which implies $\langle\langle\varphi, \xi\rangle\rangle((x_1, x_2), (y_1, y_2))$.

Case $\langle\langle\varphi_1 \wedge \varphi_2, \psi\rangle\rangle$ **and** $\langle\langle\psi, \xi\rangle\rangle$ **:**
We have $\langle\langle\varphi_1, \psi\rangle\rangle(x_1, y_1)@x_2 \wedge \langle\langle\psi, \xi\rangle\rangle(x_2, y_2)@y_1$ or
$\langle\langle\varphi_2, \psi\rangle\rangle(x_1, y_1)@x_2 \wedge \langle\langle\psi, \xi\rangle\rangle(x_2, y_2)@y_1$.
Which gives us $\langle\langle\varphi_1 \wedge \varphi_2, \xi\rangle\rangle((x_1, x_2), (y_1, y_2))$ using the IH.

Case $\langle\langle\varphi_1 \vee \varphi_2, \psi\rangle\rangle$ **and** $\langle\langle\psi, \xi\rangle\rangle$ **:**
We have $\langle\langle\varphi_1, \psi\rangle\rangle(x_1, y_1)@x_2 \wedge \langle\langle\psi, \xi\rangle\rangle(x_2, y_2)@y_1$ and
$\langle\langle\varphi_2, \psi\rangle\rangle(x_1, y_1)@x_2 \wedge \langle\langle\psi, \xi\rangle\rangle(x_2, y_2)@y_1$.
Which gives us $\langle\langle\varphi_1 \vee \varphi_2, \xi\rangle\rangle((x_1, x_2), (y_1, y_2))$ using the IH.

Case $\langle\langle\varphi, \psi_1 \wedge \psi_2\rangle\rangle$ **and** $\langle\langle\psi_1 \wedge \psi_2, \xi\rangle\rangle$ **:**
We have $\langle\langle\varphi, \psi_1\rangle\rangle(x_1, y_1)@x_2$,$\langle\langle\varphi, \psi_2\rangle\rangle(x_1, y_1)@x_2$ and
$\langle\langle\psi_1, \xi\rangle\rangle(x_2, y_2)@y_1 \vee \langle\langle\psi_2, \xi\rangle\rangle(x_2, y_2)@y_1$.If $\langle\langle\psi_1, \xi\rangle\rangle(x_2, y_2)@y_1$ holds then can deduce $\langle\langle\varphi, \xi\rangle\rangle((x_1, x_2), (y_1, y_2))$. Same thing if we have $\langle\langle\psi_2, \xi\rangle\rangle(x_2, y_2)@y_1$.

Case $\langle\langle\langle l\rangle\varphi', \psi\rangle\rangle$ **and** $\langle\langle\psi, \xi\rangle\rangle$ **:**

Suppose we have $\displaystyle\bigwedge_{x_1 \xrightarrow{l} x_1'} \langle\langle\varphi', \psi\rangle\rangle(x_1', y_1)@x_2$ and $\langle\langle\psi, \xi\rangle\rangle(x_2, y_2)@y_1$

The induction hypothesis gives us:

$$\bigwedge_{x_1 \xrightarrow{l} x_1'} \langle\langle\varphi', \xi\rangle\rangle((x_1', x_2), (y_1, y_2))$$
$$= \bigwedge_{(x_1, x_2) \xrightarrow{l} (x_1, x_2)'} \langle\langle\varphi', \xi\rangle\rangle((x_1, x_2)', (y_1, y_2))$$
$$= \langle\langle\langle l\rangle\varphi', \xi\rangle\rangle((x_1, x_2), (y_1, y_2))$$

Case $\langle\langle\langle s\rangle\varphi', \langle s\rangle\psi'\rangle\rangle$ **and** $\langle\langle\langle s\rangle\psi', \langle s\rangle\xi'\rangle\rangle$ **:**

$$\langle\langle\varphi, \psi\rangle\rangle(x_1, y_1)@y_2 = \begin{cases} \top & \text{if } \bigwedge_{x_1 \xrightarrow{s} x_1'} \bigvee_{y_1 \xrightarrow{s} y_1'} \langle\langle\varphi', \psi'\rangle\rangle(x_1', y_1')@x_2 \\ \bot & \text{otherwise} \end{cases}$$

$$\langle\langle\psi, \xi\rangle\rangle(x_2, y_2)@y_1 = \begin{cases} \top & \text{if } \bigwedge_{x_2 \xrightarrow{s} x_2'} \bigvee_{y_2 \xrightarrow{s} y_2'} \langle\langle\psi', \xi'\rangle\rangle(x_2', y_2')@y_1 \\ \bot & \text{otherwise} \end{cases}$$

So we have for any x_1' and x_2',

$$\bigvee_{y_1 \xrightarrow{s} y_1'} \langle\langle\varphi', \psi'\rangle\rangle(x_1', y_1')@x_2 \text{ and } \bigvee_{y_2 \xrightarrow{s} y_2'} \langle\langle\psi', \xi'\rangle\rangle(x_2', y_2')@y_1$$
$$= \bigwedge_{(x_1, x_2) \xrightarrow{s} (x_1, x_2)'} \bigvee_{(y_1, y_2) \xrightarrow{s} (y_1, y_2)'} \langle\langle\varphi', \psi'\rangle\rangle(x_1', y_1')@x_2 \wedge \langle\langle\psi', \xi'\rangle\rangle(x_2', y_2')@y_1$$

and we can conclude by using the induction hypothesis.

J. Lemma 5: the construction of S^{φ} terminates

$\mathsf{St}(\sigma X.\varphi')$ requires the computation of the least fixed-point of a monotonous function over $Q \times \mathcal{CL}(\varphi)$, which is finite.

K. Theorem 4: $S^{\varphi}\langle\langle\varphi\rangle\rangle S$

We begin to show the property: $\mathsf{Pass}^{\varphi} \Leftrightarrow \mathsf{Pass}^{\varphi}((x, f))$ if f is a boolean formula containing φ.

Case B**:** x and (x, f) have the same label.

Case $\varphi_1 \wedge \varphi_2$**:** $\mathsf{Pass}^{\varphi_1}(x) \wedge \mathsf{Pass}^{\varphi_2}(x) = \mathsf{Pass}^{\varphi_1}(x, f) \wedge \mathsf{Pass}^{\varphi_2}(x, f)$ (if f contains $\varphi_1 \wedge \varphi_2$ it contains φ_1 and φ_2).

Case $\langle l\rangle\varphi$: $\bigvee_{x\xrightarrow{l}x'} \mathsf{Pass}^{\varphi}(x') = \bigvee_{(x,f)\xrightarrow{l}(x',\varphi)} \mathsf{Pass}^{\varphi}(x',\varphi)$. The x' which do not appear in the second formula are those such that $\mathsf{Fail}^{\varphi}(x') = \top$, so we know that $\mathsf{Pass}^{\varphi}(x') = \bot$ for those.

Let us now show that for any states x, y and any formulas φ_1, φ_2,
$\langle\langle\varphi_1,\varphi_2\rangle\rangle(x,y) \Leftrightarrow \langle\langle\varphi_1,\varphi_2\rangle\rangle((x,\varphi_1),(y,\varphi_2))$. Then we have:
$\langle\langle\varphi_1,\varphi_2\rangle\rangle(x,(x,\varphi_2)) \Leftrightarrow \langle\langle\varphi_1,\varphi_2\rangle\rangle((x,\varphi_1),((x,\varphi_2),\varphi_2))$
$\Leftrightarrow \langle\langle\varphi_1,\varphi_2\rangle\rangle((x,\varphi_1),(x,\varphi_2)) \Leftrightarrow \langle\langle\varphi_1,\varphi_2\rangle\rangle(x,x)$.

Case $\mathsf{Fail}^{\psi}, \mathsf{Pass}^{\varphi}$: Fail and Pass are preserved.

Case $\psi, \varphi_1\wedge\varphi_2$: $\langle\langle\psi,\varphi_1\wedge\varphi_2\rangle\rangle(x,y) = \langle\langle\psi,\varphi_1\rangle\rangle(x,y) \wedge \langle\langle\psi,\varphi_2\rangle\rangle(x,y)$
$= \langle\langle\psi,\varphi_1\rangle\rangle((x,\psi),(y,\varphi_1\wedge\varphi_2)) \wedge \langle\langle\psi,\varphi_2\rangle\rangle((x,\psi),(y,\varphi_1\wedge\varphi_2))$
$= \langle\langle\psi,\varphi_1\wedge\varphi_2\rangle\rangle((x,\psi),(y,\varphi_1\wedge\varphi_2))$.

Case $\psi, \langle l\rangle\varphi$: $\langle\langle\psi,\langle l\rangle\varphi\rangle\rangle(x,y) = \mathsf{Fail}^{\psi}(x) \vee \bigvee_{y\xrightarrow{l}y'} \langle\langle\psi,\varphi\rangle\rangle(x,y') = \mathsf{Fail}^{\psi}((x,\psi)) \vee \bigvee_{(y,\langle l\rangle\varphi)\xrightarrow{l}(y',\varphi)} \langle\langle\psi,\varphi\rangle\rangle((x,\psi),(y',\varphi))$. Because for any y' such that $\mathsf{Fail}^{\varphi}(y') = \top$ (those which are not mapped to a state (y',φ) of reachability graph), we have $\langle\langle\psi,\varphi\rangle\rangle(x,y') \Rightarrow \mathsf{Fail}^{\psi}(x)$, which make them useless in the formula.

Case $\langle s\rangle\psi, \langle s\rangle\varphi$: $\langle\langle\langle s\rangle\psi,\langle s\rangle\varphi\rangle\rangle(x,y) =$
$\mathsf{Fail}^{\langle s\rangle\psi}(x) \vee \begin{cases} \top & \text{if } \bigwedge_{x\xrightarrow{s}x'} \mathsf{Fail}^{\psi}(x') \vee \bigvee_{y\xrightarrow{s}y'} \langle\langle\psi,\varphi\rangle\rangle(x',y') \\ \bot & \text{otherwise} \end{cases}.$
The argument is similar to the one above. If $\mathsf{Fail}^{\psi}(x') = \top$ then it can be removed from the conjunction. And if $\mathsf{Fail}^{\varphi}(y') = \top$ then we have $\langle\langle\psi,\varphi\rangle\rangle(x',y') \Rightarrow \mathsf{Fail}^{\psi}(x')$, so the term $\langle\langle\psi,\varphi\rangle\rangle(x',y')$ brings nothing to the conjunction.

L. Theorem 5: the replacement operation preserves $\langle\langle\varphi\rangle\rangle$

The transitivity of $\langle\langle\psi,\varphi\rangle\rangle$ implies that if $x\langle\langle\varphi\rangle\rangle y$ then for any z and ψ,
$\langle\langle\psi,\varphi\rangle\rangle(z,x) \Leftrightarrow \langle\langle\psi,\varphi\rangle\rangle(z,y)$ as well as $\langle\langle\varphi,\psi\rangle\rangle(x,z) \Leftrightarrow \langle\langle\varphi,\psi\rangle\rangle(y,z)$.
In addition, the fact that there is no paths from y to x means that the state x will not be reachable by the computation of $\langle\langle\psi,\varphi\rangle\rangle(z,y)$ or $\langle\langle\varphi,\psi\rangle\rangle(y,z)$, so the removal of the transitions leading to x will not affect these values.

M. Lemma 6: $\sim^{\varphi}$ is well defined and is an equivalence relation

The argument for the convergence of the computation of $\sim^{\varphi}$ is the same as for $\langle\langle\psi,\varphi\rangle\rangle$. The reflexivity and symmetry of $\sim^{\varphi}$ are quite straightforward from the definition. Let us show that if we have $\sim^{\varphi}(x,y) \wedge \sim^{\varphi}(y,z)$ then we have $\sim^{\varphi}(x,z)$.

Case B: We have $B@x \Leftrightarrow B@y$ and $B@y \Leftrightarrow B@z$, which gives us $B@x \Leftrightarrow B@z$.

Case $\varphi_1\wedge\varphi_2$: The transitivity of $\sim^{\varphi_1}$ and of $\sim^{\varphi_2}$ propagates to $\sim^{\varphi_1\wedge\varphi_2}$.

Case $\langle l\rangle\varphi$: Let us suppose $\neg\mathsf{Fail}^{\langle l\rangle\varphi}(x)$, this means that we have
$\bigwedge_{x\xrightarrow{l}x'} \bigvee_{y\xrightarrow{l}y'} \sim^{\varphi}(x',y')$. This also imply that $\mathsf{Fail}^{\langle l\rangle\varphi}(y)$ is not verified, and thus that we have $\bigwedge_{y\xrightarrow{l}y'} \bigvee_{z\xrightarrow{l}z'} \sim^{\varphi}(y',z')$.
From there we can conclude $\bigwedge_{x\xrightarrow{l}x'} \bigvee_{z\xrightarrow{l}z'} \sim^{\varphi}(x',z')$. The same can be done for the symmetrical conditions.

N. Theorem 6: compositionality of $\approx^{\psi}$

The formulas which are part of $\mathsf{Form}(x_1)$ and $\mathsf{Form}(x_2)$ verify the compositionality property, *i.e.* $\mathsf{Form}((x_1,x_2)) \subseteq \mathsf{Form}(x_1) \cap \mathsf{Form}(x_2)$. It remains to show that for any formula φ, we have $(\sim^{\varphi}(x_1,y_1) \wedge \sim^{\varphi}(x_2,y_2)) \Rightarrow \sim^{\varphi}((x_1,y_1),(x_2,y_2))$. We define the property $\mathcal{P}(f_1,f_2,f_3)$ as the following: for any x_1,y_1,x_2,y_2,

$$f_1(x_1,y_1) \Rightarrow f_2(x_2,y_2) \Rightarrow f_3((x_1,x_2),(y_1,y_2)).$$

We then show that the rules used to build $\sim^{\varphi}$ preserves this property.

Case B: $B@x_1 \Leftrightarrow B@y_1$ and $B@x_2 \Leftrightarrow B@y_2$ so $B@x_1@x_2 \Leftrightarrow B@y_2@y_2$

Case $\wedge$: If f_1,f_2,f_3 verify $\mathcal{P}(f_1,f_2,f_3)$ and f_1',f_2',f_3' verify $\mathcal{P}(f_1',f_2',f_3')$ then we have $\mathcal{P}(f_1\wedge f_1', f_2\wedge f_2', f_3\wedge f_3')$.

Case $\langle l\rangle\varphi$: We have shown the compositionality of Fail in the proof of lemma 2. Let us consider (wlog) the case where l is local to the module 1. Suppose $\bigwedge_{x_1\xrightarrow{l}x_1'} \bigvee_{y_1\xrightarrow{l}y_1'} \sim^{\varphi}(x_1',y_1')$ and $\sim^{\varphi}(x_2,y_2)$.
We have $\bigwedge_{(x_1,x_2)\xrightarrow{l}(x_1,x_2)'} \bigvee_{(y_1,y_2)\xrightarrow{l}(y_1,y_2)'} \sim^{\varphi}((x_1,x_2)',(y_1,y_2)')$
$= \bigwedge_{x_1\xrightarrow{l}x_1'} \bigvee_{y_1\xrightarrow{l}y_1'} \sim^{\varphi}((x_1',x_2),(y_1',y_2))$.

O. Theorem 7: $\approx^{\psi}$ preserves Pass

Let us show that if $x \approx^{\psi} y$ then for any $\varphi \in \mathsf{Form}(x)$, $\mathsf{Pass}^{\varphi}(x) \Leftrightarrow \mathsf{Pass}^{\varphi}(y)$.

Case $\mu X \varphi$: Exists $n \in \mathbb{N}$ such that $\sim^{\mu X \varphi} = \sim^{\varphi^n(\bot)}$ and $\mathsf{Pass}^{\mu X \varphi} = \mathsf{Pass}^{\varphi^n(\bot)}$.

Case $\nu X \varphi$: Exists $n \in \mathbb{N}$ such that $\sim^{\nu X \varphi} = \sim^{\varphi^n(\top)}$ and $\mathsf{Pass}^{\nu X \varphi} = \mathsf{Pass}^{\varphi^n(\top)}$.

Case B: We have $\sim^{B} (x,y) = B@a \Leftrightarrow B@y, \mathsf{Pass}^{B}(x) = B@x$ and $\mathsf{Pass}^{B}(y) = B@y$.

Case $\varphi_1 \wedge \varphi_2$: Let us suppose $\sim^{\varphi_1} (x,y) \wedge \sim^{\varphi_2} (x,y)$ and $\mathsf{Pass}^{\varphi_1}(x) \wedge \mathsf{Pass}^{\varphi_2}(x)$.
The induction hypothesis gives us $\mathsf{Pass}^{\varphi_1}(y) \wedge \mathsf{Pass}^{\varphi_2}(y)$.

Case $\varphi_1 \vee \varphi_2$: As before we suppose $\sim^{\varphi_1} (x,y) \wedge \sim^{\varphi_2} (x,y)$ as well as $\mathsf{Pass}^{\varphi_1}(x) \vee \mathsf{Pass}^{\varphi_2}(x)$. Which gives us $\mathsf{Pass}^{\varphi_1}(y) \vee \mathsf{Pass}^{\varphi_2}(y)$.

Case $\langle l \rangle \varphi$: We have

$$\sim^{\langle l \rangle \varphi} (x,y) = \mathsf{Fail}^{\langle l \rangle \varphi}(x) \vee \bigwedge_{x \xrightarrow{l} x'} \bigvee_{y \xrightarrow{l} y'} \sim^{\varphi} (x', y')$$

$$\bigwedge \mathsf{Fail}^{\langle l \rangle \varphi}(y) \vee \bigwedge_{y \xrightarrow{l} y'} \bigvee_{x \xrightarrow{l} x'} \sim^{\varphi} (y', x')$$

and $\mathsf{Pass}^{\langle l \rangle \varphi}(x) = \bigvee_{x \xrightarrow{l} x'} \mathsf{Pass}^{\varphi}(x')$.

If $\mathsf{Fail}^{\langle l \rangle \varphi}$ is verified then $\mathsf{Pass}^{\langle l \rangle \varphi}$ is not and the implication holds. So let us suppose $\bigwedge_{x \xrightarrow{l} x'} \bigvee_{y \xrightarrow{l} y'} \sim^{\varphi} (x', y')$ and $\bigvee_{x \xrightarrow{l} x'} \mathsf{Pass}^{\varphi}(x')$.
This gives us $\bigvee_{y \xrightarrow{l} y'} \mathsf{Pass}^{\varphi}(y')$.

Cases $\langle s \rangle \varphi$ and $\langle e \rangle \varphi$: $\mathsf{Pass}^{\langle s \rangle \varphi}(x)$ and $\mathsf{Pass}^{\langle e \rangle \varphi}(x)$ both are equal to $\bot$ for any x.

P. Theorem 8: the quotienting preserves the equivalence

We have $\mathsf{Form}(x) = \mathsf{Form}([x])$. Let us show that for any state x, for any $\varphi \in \mathsf{Form}(x), x \sim^{\varphi} [x]$.

Case B: x and $[x]$ have the same label.

Case $\varphi_1 \vee \varphi_2$: $\sim^{\varphi_1 \wedge \varphi_2} (x, [x]) = \sim^{\varphi_1} (x, [x]) \wedge \sim^{\varphi_2} (x, [x])$, which is true according to the IH.

Case $\langle l \rangle \varphi$:

$$\sim^{\langle l \rangle \varphi} (x, [x]) = \mathsf{Fail}^{\langle l \rangle \varphi}(x) \vee \bigwedge_{x \xrightarrow{l} x'} \bigvee_{[x] \xrightarrow{l} [x]'} \sim^{\varphi} (x', [x]') = \top$$

because for any x' there exist $[x]'$ such that $[x]' = [x']$,

and if x has no successor then $\mathsf{Fail}^{\langle l \rangle \varphi}(x) = \top$.

$$\sim^{\langle l \rangle \varphi} ([x], x) = \mathsf{Fail}^{\langle l \rangle \varphi}([x]) \vee \bigwedge_{[x] \xrightarrow{l} [x]'} \bigvee_{x \xrightarrow{l} x'} \sim^{\varphi} ([x]', x') = \top$$

because for any x' there exist $[x]'$ such that $[x]' = [x']$,

and if x has no successor then $\mathsf{Fail}^{\langle l \rangle \varphi}(x) = \top$.

Cases $\langle e \rangle \varphi$ and $\neg \varphi$: We can apply the induction hypothesis directly.