Modal Abstractions in µCRL

Jaco van de Pol, Miguel Valero Espada

REPORT SEN-R0401 FEBRUARY 15, 2004

ISSN 1386-369X

Modal Abstractions in μCRL

ABSTRACT
We describe a framework to generate modal abstract approximations from process algebraic specifications, written in the language μCRL. We introduce a new format for processes specification called *Modal Linear Process Equation* (MLPE). An MLPE represents all the possible interleavings of the parallel composition of a number of processes. Every transition step may lead to a set of abstract states labelled with a set of abstract actions. We use MLPEs to characterize abstract interpretations of systems and to generate *Modal Labelled Transition Systems*, in which transitions may have two modalities *may* and *must*. We prove that the abstractions are sound for the full action-based μ-calculus. The main parts of the theory have been formalized and proved using the automatic theorem prover PVS. The set of definitions, theorems and proofs composes a reusable framework to formally analyze process specifications.

2000 Mathematics Subject Classification: 68Q60;68Q65;68Q85
1998 ACM Computing Classification System: D.2.1;D.3.1
Keywords and Phrases: Process Algebra; Abstract Interpretation; Modal Transition Systems

Modal Abstractions in μCRL[⋆]

Jaco van de Pol and Miguel Valero Espada

Centrum voor Wiskunde en Informatica,
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
{Jaco.van.de.Pol, Miguel.Valero.Espada}@cwi.nl

Abstract. We describe a framework to generate modal abstract approximations from process algebraic specifications, written in the language μCRL. We introduce a new format for processes specification called *Modal Linear Process Equation* (MLPE). An MLPE represents all the possible interleavings of the parallel composition of a number of processes. Every transition step may lead to a set of abstract states labelled with a set of abstract actions. We use MLPEs to characterize abstract interpretations of systems and to generate *Modal Labelled Transition Systems*, in which transitions may have two modalities *may* and *must*. We prove that the abstractions are sound for the full action-based μ-calculus. The main parts of the theory have been formalized and proved using the automatic theorem prover PVS. The set of definitions, theorems and proofs composes a reusable framework to formally analyze process specifications.

1 Introduction

The theory of abstract interpretation [6, 18] denotes a classical framework for program analysis. It extracts program approximations by eliminating uninteresting information. Computations over concrete universes of data are performed over smaller abstract domains. The application of abstract interpretation to the verification of systems is suitable since it allows to formally transform possibly infinite instances of specifications into smaller and finite ones. By loosing some information we can compute a desirable view of the analysed system that preserves some interesting properties of the original. Abstract Interpretation has been successfully used to perform *control* and *data flow analysis* and to deal with the state space explosion problem, for a comprehensive introduction see [5]. A typical example of the technique is the so-called "rule of signs" used to determine the sign of arithmetic expressions by performing the computation only over the signs of the operators, i.e. the expression $-5 * 10$ is abstracted to $neg * pos$ which preserves the sign of the result.

The achievement of this paper is to enhance existing process algebraic verification tools (e.g. LOTOS, μCRL) with state-of-the-art abstract interpretation

⋆ Partially supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, grant CES.5009.

techniques that exist for state-based reactive systems. These techniques are based on homomorphisms [4, 10] (easier to use) or Galois Connections [22, 8, 17, 14] (more precise abstractions). The latter are sound for safety as well as liveness properties. A three-valued logic ensures that the theory can be used for proofs and refutations of temporal properties. We transpose this to a process algebra setting, allowing abstraction of states and action labels, and treating homomorphisms and Galois Connections in a uniform way. A preliminary step was already taken in [11]; those authors show how process algebras can benefit from abstract interpretation *in principle*. To this end they work with a basic LOTOS language and a simple temporal logic; their abstractions preserve linear-time safety properties only.

Semantically, our method is based on *Modal Labelled Transition Systems* [21, 20]. MLTSs are *mixed* transition systems in which transitions are labelled with actions and with two modalities: *may* and *must*. They are appropriate structures to define abstraction/refinement relations between processes. *May* transitions determine the actions that possibly occur in all refinements of the system while *must* transitions denote the ones that necessarily happen. On the one hand, the *may* part corresponds to an over-approximation that preserves *safety* properties of the concrete instance and on the other hand the *must* part under-approximates the model and reflects *liveness* properties. We define approximations and prove that they are sound for all properties in the full (action-based) μ-calculus [19], including negation. We had to extend existing theory by allowing abstraction and information ordering of action labels, which is already visible in the semantics of μ-calculus formulas.

This theory is applied to μCRL specifications, which (as in LOTOS) consist of an ADT part defining data operations, and a process specification part, specifying an event-based reactive system. Processes are defined using a.o. sequential and parallel composition, non-deterministic choice and hiding. Furthermore, atomic actions, conditions and recursion are present, and may depend on data parameters. The μCRL toolset transforms specifications to *linear process equations* (LPE), by eliminating parallel composition and hiding efficiently. The μCRL language and tool set have been used in numerous verifications of communication and security protocols and standards, distributed algorithms and industrial embedded systems.

We implement abstract interpretation as a transformation of LPEs to MLPEs (modal LPEs). MLPEs capture the extra non-determinism arising from abstract interpretation. They allow a simple transition to lead to a *set* of states with a *set* of action labels. We show that the MLTS generated from an MLPE is a proper abstraction of the LTS generated from the original LPE. This implies soundness for μ-calculus properties. Section 4 is devoted to this part. MLPEs can be represented by LPEs. Thus our method integrates perfectly with the existing transformation and state space generation tools of the μCRL toolset [2, 3]. Also, the three valued model checking problem can be rephrased as the usual model checking problem, along the lines of [14]. This enables the reuse of the model checkers in the CADP toolset [12]. However, the latter two transformations are not detailed in the current report.

Our approach differs from the classical on the fact that instead of giving the abstract semantics of the original model we symbolically generate a new specification that captures the abstract behavior. Furthermore, we consider that the abstract transformation of the concrete system to the MLPE format is important because it permits to apply other symbolic process transformation techniques and tools to the abstract systems [2].

The main part of theory mentioned above has been defined and proved correct in an elegant way using the computer assisted theorem prover PVS [24]. The use of the theorem prover gives extra confidence about the correctness of the theory. Furthermore, the definitions and proofs can be reused to easily extend the theory, to prove other transformations, or to apply the same techniques to another specification language. Also, this prover could be used to prove the safety conditions generated for user-specified abstractions.

To improve usability, we have predefined a few standard abstractions. Of course, the user can define specific abstractions, which in general lead to the generation of safety conditions. Finally, thanks to the uniform treatment, the tool can automatically lift a (predefined or user specified) homomorphism to a Galois Connection, thus combining ease with precision.

The report is organized as follows, first we present the main results about MLTSs and the semantic abstraction of a *Labelled Transition Systems* (LTS) to the mixed one following different approaches. Then we introduce the logical characterization of the abstractions. The next section is dedicated to the explanation of how to construct the abstraction directly from the μCRL specification. Section 5 is dedicated present the set of PVS theories that model the introduced frameworks.

2 Transition Systems

The set of techniques, used to translate a concrete system to a "safe" abstract instance of it, is normally called abstract interpretation and it has been already studied for many years. These techniques have their roots in the seminal papers of Abstract Interpretation[1] by Cousot and Cousot [6, 7]. The main idea is to provide a relation between the concrete data domain and an abstract version of it in such a way that the interpretation of the system over the abstract domain preserves and/or reflects some properties of the original. Although, part of results included in this section are well known in the field, we adapt classical frameworks for generating safe abstract approximations, by doing a non-trivial extension of them in order to allow the explicit abstraction of action labels which will permit to manipulate infinitely branching systems. Furthermore, we integrate in a uniform theory two broadly used approaches, the one based on homomorphic

[1] We use abstract interpretation when we speak about the general framework and Abstract Interpretation (with capitals) to refer to Cousots' work.

mappings between concrete and abstract domains and the one based on Galois Connections introduced by Cousots.

We start by presenting a small example that will be used as illustration of the techniques. The system is composed by two processes that communicate by sending natural numbers through a channel described as a FIFO buffer of size N, see Figure below. The system has two sources of infinity: the size of the buffer that can be arbitrarily big and the value of the data which values may belong to an infinite domain.

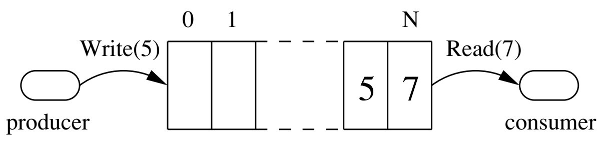


Fig. 1. Simple Buffer of size N

In order to verify properties of the system such as *"If the buffer is full, the producer cannot write anything"* we do not need to know neither how many items are in the buffer (we only have to know whether is full or not) nor what is the exact value of the stored data. But, if we decide to apply model checking techniques to prove the correctness of the system we realize the impossibility of generating the full state space due to the two sources of infiniteness. Therefore, we may consider an abstract and finite version of the system in which all these irrelevant details are omitted and that preserves the properties we are interested in. On the one hand, the content of the transferred items can be removed keeping only the information about the type of action performed (read or write) and, on the other hand, the FIFO list can be abstracted to a set of values determining the state of the buffer: *empty*, *full*, or anything in between *middle*. The abstract representation of the system, although loose some information of the original model, allows to check some interesting properties, as the above presented.

Now, let us define some general concepts and then we will continue by introducing the different abstraction techniques. The semantics of a system can be defined by a *Labelled Transition System.* We assume a non-empty set S of states, together with a non-empty set of transition labels A:

Definition 1 *A transition is a triple $s \xrightarrow{a} s'$ with $a \in A$ and s,$s' \in S$. We define a* Labelled Transition System *(LTS) as tuple $(S, A, \rightarrow, s_0)$ in which S and A are defined as above and $\rightarrow$ is a possibly infinite set of transitions and s_0 in S is the initial state.*

Figure 2 illustrates the LTS corresponding to the example above introduced. Actions R and W denote respectively read and write operations.

To model abstractions we are going to use a different structure that allows to represent approximations of the concrete system in a more suitable way. As

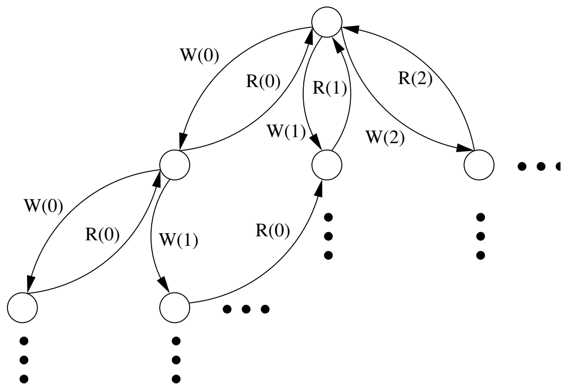


Fig. 2. LTS of a buffer system

introduced before, in *Modal Labelled Transition Systems* transitions have two modalities *may* and *must* which denote the possible and necessary steps in the refinements. This concept was introduced by Larsen and Thomsen [21]. Let us see the definition:

Definition 2 *A* Modal Labelled Transition System *(MLTS) or* may/must *labelled transition system (*may/must-*LTS) is a tuple* $(S, A, \rightarrow_{\Diamond}, \rightarrow_{\Box}, s_0)$ *where* S, A *and* s_0 *are as in the previous definition and* $\rightarrow_{\Diamond}, \rightarrow_{\Box}$ *are possibly infinite sets of (may or must) transitions of the form* $s \overset{a}{\rightarrow}_x s'$ *with* s,$s' \in S$, $a \in A$ *and* $x \in \{\Diamond, \Box\}$. *We require that every* must-*transition is a* may-*transition* ($\overset{a}{\rightarrow}_{\Box} \subseteq \overset{a}{\rightarrow}_{\Diamond}$).

MLTSs are suitable structures for stepwise refinements and abstractions. A refinement step of a system is done by preserving or extending the existing *must*-transitions and by preserving or removing the *may*-transitions. Abstraction is done the other way around. To every LTS corresponds a trivially equivalent MLTS constructed by labelling all transitions with both modalities; we will call it the corresponding *concrete* MLTS.

2.1 Homomorphism

The first approach to extract abstract *Modal Labelled Transition Systems* from LTSs that we are going to present is based on homomorphisms that relate concrete states and action labels with the their abstract versions. This theory was introduced by Clarke and Long [4], it is intuitively simple and it allows strong reductions of the state spaces.

Having a set of states S and a set of action labels A with their corresponding abstract sets, denoted by $\widehat{S}$ and $\widehat{A}$, we define a homomorphism H as a pair of total and surjective functions $\langle h_S, h_A \rangle$, where h_S is a mapping from states to abstract states, i.e., $h_S : S \to \widehat{S}$, and h_A maps action labels to abstract action labels, i.e., $h_A : A \to \widehat{A}$. The abstract state $\widehat{s}$ corresponds to all the states s for which $h_S(s) = \widehat{s}$, and the abstract action label $\widehat{a}$ corresponds to all the actions a for which $h_A(a) = \widehat{a}$.

Definition 3 *Given a* concrete *MLTS P $(S, A, \to_\diamond, \to_\square, s_0)$ and a homomorphism H, we say that $\widehat{P}$ defined by $(\widehat{S}, \widehat{A}, \to_\diamond, \to_\square, \widehat{s}_0)$ is the* minimal may/must$_H$-abstraction *(denoted by $\widehat{P} = min_H(P)$) if and only if $h_S(s_0) = \widehat{s}_0$ and the following conditions hold:*

- $\widehat{s} \xrightarrow{\widehat{a}}_\diamond \widehat{r} \iff \exists\, s, r, a.\, h_S(s) = \widehat{s} \wedge h_S(r) = \widehat{r} \wedge h_A(a) = \widehat{a} \wedge s \xrightarrow{a}_\diamond r$
- $\widehat{s} \xrightarrow{\widehat{a}}_\square \widehat{r} \iff \forall\, s.h_S(s) = \widehat{s}.\, (\exists\, r, a.\, h_S(r) = \widehat{r} \wedge h_A(a) = \widehat{a} \wedge s \xrightarrow{a}_\square r)$

This definition gives the most accurate abstraction of a concrete system by using a homomorphism, in other words the one that preserves most information of the original system.

Figure[2] below shows the minimal abstraction of the buffer model in which only the states have been abstracted, action labels remain as in the original. H is equal to the pair $\langle h_S, Id_A \rangle$ in which h_S is defined as follows: it maps the initial state to the abstract state e, which means *empty*, the states in which there are N entries in the buffer to f, which represents *full*, and the rest of the states to m, which means something in the *middle*.

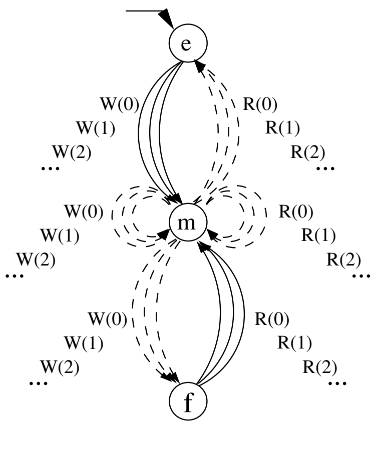


Fig. 3. Abstract Buffer size N; abstraction of states

We see that the system cannot be completely represented, even if the state space is finite, because the is infinitely branching. Therefore, we need also to

[2] For clarity when there is a *must* transition we do not include the corresponding *may* one.

apply abstraction on action labels in order to be able to use model checking techniques over the abstract system. We define h_A as follows: it maps all the write actions to $\widehat{w}$ and all the read actions to $\widehat{r}$.

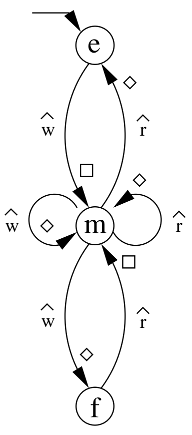


Fig. 4. Abstract Buffer size N; abstraction of states and action labels

In the final system, by the combination of both abstractions, we have removed all the information about the values that are in the buffer and the transferred data, only preserving the information about whether the buffer is empty, full or none of them. This abstraction allows to have a small finite model which keeps some information about the original. The example clearly illustrates the importance of the abstraction of action labels.

2.2 Galois Connection

Instead of using mappings between concrete and abstract domains we can define relations. The other classical approach we present is based on Galois Connections between domains, and it was introduced in the late seventies by Cousot and Cousot [6], see also [8, 9] for a good introduction. Two functions α and γ over two partially ordered sets $(P, \subseteq)$ and $(Q, \preccurlyeq)$ such that $\alpha : P \rightarrow Q$ and $\gamma : Q \rightarrow P$ form a Galois Connection if and only if the following conditions hold:

1. α and γ are total and monotonic.
2. $\forall p : P, p \subseteq \gamma \circ \alpha(p)$.
3. $\forall q : Q, \alpha \circ \gamma(q) \preccurlyeq q$.

α is the lower adjoint and γ is the upper adjoint of the Galois Connection, and they uniquely determine each other. Galois Connections enjoy suitable properties to perform abstractions. Let us consider $\mathcal{P}(S)$ and $\mathcal{P}(A)$ being partially ordered sets ordered by the set inclusion operator and the abstract $\widehat{S}$ and $\widehat{A}$ both being posets equipped with some order $\preccurlyeq$. The order gives a relation of the precision of the information contained in the elements of the domain. We define a pair G of Galois Connections: $G = \langle(\alpha_S, \gamma_S), (\alpha_A, \gamma_A)\rangle$. α is usually called the abstraction function and γ the concretization function.

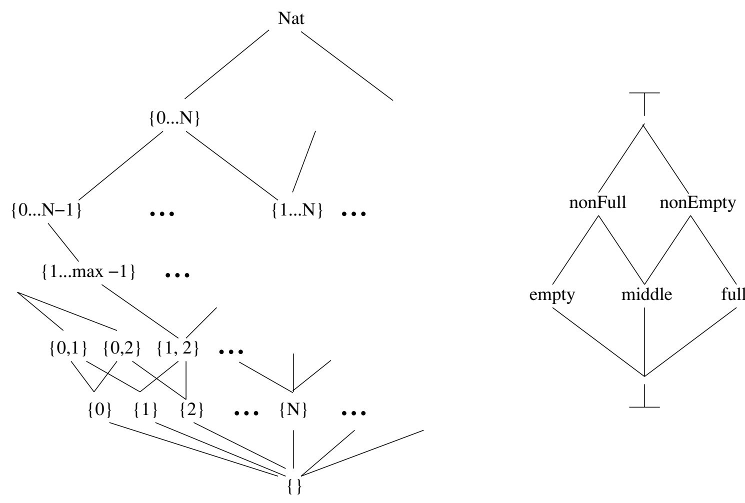


Fig. 5. Concrete and Abstract lattices of Naturals

Figure 5 shows two lattices corresponding to the representation of the naturals that we use to model the current size of the buffer. The concrete lattice is built over the power sets of naturals, the upper and lower bounds are represented by the full set (*Nat*) and the empty set ($\{\}$). The abstract domain is the previously used set of abstract naturals *empty, middle and full* extended with two new values *nonEmpty* and *nonFull* and with $\top$ and $\bot$ in order to complete the lattice, we abbreviate the names as $\{\bot, e, m, f, nE, nF, \top\}$. The definition of $\alpha(S)$ for S being a set of concrete values is:

- **if** $S = \{0\}$ **then** *empty*
- **else if** $\forall s \in S. 0 < s < N$ **then** *middle*
- **else if** $S = \{N\}$ **then** $full$
- **else if** $\forall s \in S. 0 < s \leq N$ **then** *nonEmpty*
- **else if** $\forall s \in S. s < N$ **then** $nonFull$
- **else if** $S = \{\}$ **then** $\bot$
- **otherwise** $\top$

We define $\gamma(\widehat{s})$ as:

- **if** $\widehat{s} = \bot$ **then** $\{\}$

– **if** $\widehat{s} = \mathit{empty}$ **then** $\{0\}$
– **if** $\widehat{s} = \mathit{middle}$ **then** $\{1, ..., N-1\}$
– **if** $\widehat{s} = \mathit{full}$ **then** $\{N\}$
– **if** $\widehat{s} = \mathit{nonEmpty}$ **then** $\{1, ..., N\}$
– **if** $\widehat{s} = \mathit{nonFull}$ **then** $\{0, ..., N-1\}$
– **if** $\widehat{s} = \top$ **then** $\widehat{s} = \top$ Nat

We can also define the abstraction of the action labels, the abstract lattice is presented in Figure 6. α_A of a set of action labels B only composed by write actions will be equal to $\widehat{w}$, $\alpha(B)$ with B composed by read actions will be equal to $\widehat{r}$, and if there are read and write actions in B then $\alpha(B)$ will be $\top$ and $\alpha(\{\})$ will be $\perp$. γ is trivially defined according to α.

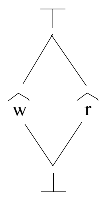


Fig. 6. Galois Connection of Actions Labels

As in the case of the homomorphism we define the minimal abstraction, as follows:

Definition 4 *Given two systems P and $\widehat{P}$ defined as in definition 3 and a pair of Galois Connections G, $\widehat{P}$ is the* minimal may/must$_G$-abstraction *(denoted by $\widehat{P} = min_G(P)$) if and only if $s_0 \in \gamma_S(\widehat{s}_0)$ and the following conditions hold:*

– $\widehat{s} \xrightarrow{\widehat{a}}_{\Diamond} \widehat{r} \iff \exists\, s \in \gamma_S(\widehat{s}),\, r \in \gamma_S(\widehat{r}),\, a \in \gamma_A(\widehat{a}).\, s \xrightarrow{a}_{\Diamond} r$
– $\widehat{s} \xrightarrow{\widehat{a}}_{\Box} \widehat{r} \iff \forall\, s \in \gamma_S(\widehat{s}).\, (\exists\, r \in \gamma_S(\widehat{r}),\, a \in \gamma_A(\widehat{a}).\, s \xrightarrow{a}_{\Box} r)$

Next figure presents part of the minimal abstraction of the buffer system[3].

The order defined over the abstract lattices gives a relation about the precision of the information contained in the values. For example, *empty* has more accurate information than *nonFull* and the latter more than $\top$. Furthermore, the order induces a precision relation between transitions, for example the Figure 8 shows different abstract write transitions; *empty* $\rightarrow$ *middle* is the most precise because the destination state is the smallest of all possible continuations. The other two transitions have less information. The same holds for the order over the action labels, for example *empty* $\xrightarrow{\widehat{w}}$ *middle* is more precise than *empty* $\xrightarrow{\top}$ *middle*.

[3] For readability, we separate write transitions: $\xrightarrow{\widehat{w}}$ and read transitions $\xrightarrow{\widehat{r}}$ and we do not include transitions to and from $\top$, or labelled with it.

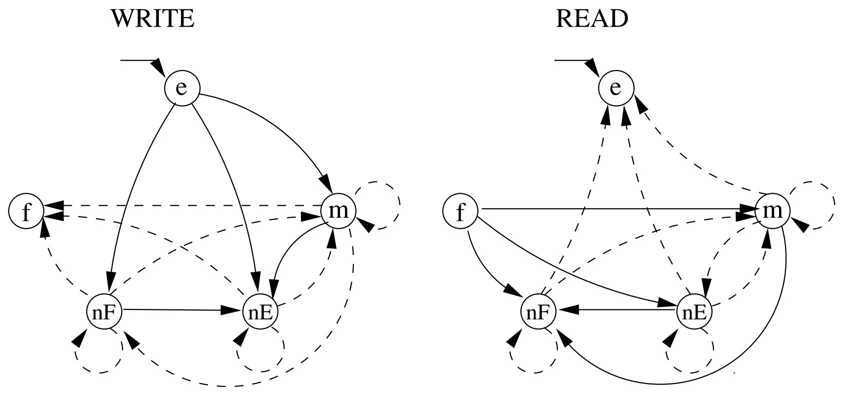


Fig. 7. Abstract Buffer size N (Galois Connection)

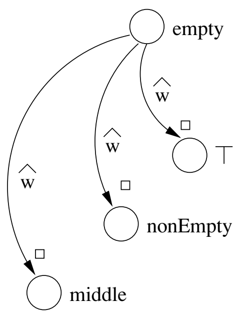


Fig. 8. Precision Relation

Due to the order over the abstract states and actions, the minimal system defined by the above presented definition is saturated of *may* and *must* transitions, i.e. there are transitions that do not add any extra information. We can easily see in the previous figure that the *must* part is saturated for example, the transition $e \xrightarrow{\widehat{w}}_{\square} nE$ does not add any information because we have $e \xrightarrow{\widehat{w}}_{\square} m$ which is more precise. We can restrict our definition by requiring that the abstract transitions are performed only between the most precise descriptions of the concrete transitions, as done in Dams' theory [8]. To do so, first we define:

– For every $\widehat{s}$, let M_{may} be equal to the set of pairs of (R, B) such that $\exists\, s \in \gamma_S(\widehat{s})$, $r \in R$, $a \in B.s \xrightarrow{a}_{\diamond} r$ is in P. M_{may} represents all sets of possible continuations of the concrete values related with $\widehat{s}$.
– M^{min}_{may} as the set of minimal elements in M_{may}:
$M^{min}_{may} = \{(R_m, B_m) \in M_{may} \mid \forall\, (R, B) \in M_{may}.\neg R \subset R_m \wedge \neg B \subset B_m\}$.
Among all possible sets of continuations we select the minimal ones.

– And $\widehat{M}_{may}$ as $\{(\widehat{r}, \widehat{a}) \mid \exists\,(R_{min}, B_{min}) \in M^{min}_{may} \wedge \widehat{r} = \alpha(R_{min}) \wedge \widehat{a} = \alpha(B_{min})\}$. $\widehat{M}_{may}$ is the abstraction of the minimal sets.

Then, we keep only the transitions $\widehat{s} \xrightarrow{\widehat{a}}_{\diamond} \widehat{r}$ iff $(\widehat{r}, \widehat{a}) \in \widehat{M}_{may}$. This rule eliminates the redundant *may* transitions. Observe that M^{min}_{may} will be always composed by singleton sets. To remove redundant *must* ones we proceed in the same way, the only change is done in the first definition:

– For every $\widehat{s}$, the set M_{must} equals to the set of pairs of (R, B) such that $\forall\, s \in \gamma_S(\,\widehat{s}\,),$ implies $\exists\, r \in R,\ a \in B.s \xrightarrow{a}_{\square} r$ is in P.

In the *must* case M^{min}_{must} is not necessarily compose by only singleton sets. We called to this minimal system *restricted* and it is denoted by $\widehat{P}{\downarrow}$. In general the condition $\xrightarrow{a}_{\square} \subseteq \xrightarrow{a}_{\diamond}$ will not hold anymore so add an extra rule to preserve the property. This rule simply requires that for every *must* transition the system will have also a *may* one. Note that no information is removed because only unprecise transitions are removed.

Let us compute the *restricted* set of transitions for the example in case $\widehat{s}$ equals *empty*:

– The following pairs will be in M_{may}[4]:
  - $(\{s_{1,0}\}, \{w(0)\})$
  - $(\{s_0, s_{1,0}\}, \{w(1)\})$
  - $(\{s_{1,2}\}, \{w(0), w(1)\})$
  - $(\{s_{1,0}, s_{N,1}\}, \{w(0), r(0)\})$
  - ...

  Therefore, M_{may} will be composed by all (R, B) such that there is at least one '$s_{1,x}$' in R and $w(d)$ for at least one d is in B.
– The set of the minimal pairs M^{min}_{may} will be:
  $\{(\{s_{1,x}\}, \{w(0)\}), (\{s_{1,x}\}, \{w(1)\}), \ldots\}$ for any x.
– Then $\widehat{M}_{may}$ will be equal to $\{(\mathit{middle}, \widehat{w})\}$.
– Therefore, we just keep the transition *empty* $\xrightarrow{\widehat{w}}_{\diamond}$ *middle* and we remove all the rest of outgoing transitions.

For the complete system, the following transitions are removed: $e \xrightarrow{\widehat{w}}_{\diamond,\square} nF$, $e \xrightarrow{\widehat{w}}_{\diamond,\square} nE$, $m \xrightarrow{\widehat{w}}_{\diamond} nF$, $m \xrightarrow{\widehat{r}}_{\diamond} nE$, $f \xrightarrow{\widehat{r}}_{\diamond,\square} nF$, $f \xrightarrow{\widehat{r}}_{\diamond,\square} nE$, $nF \xrightarrow{\widehat{w},\widehat{r}}_{\diamond} nF$ and $nE \xrightarrow{\widehat{w},\widehat{r}}_{\diamond} nE$. Figure 9 shows the result of the restriction.

In general we would not compute the minimal *restricted* abstraction, but just approximations of it. This form will be useful in order to characterize the information preserved by the abstractions as we will see in following sections.

[4] We denote by s_0 the initial state of figure 2, by $s_{1,x}$ the concrete states of the first row, after the insertion of one entry, by $s_{2,x}$ the sates of the second row etc...

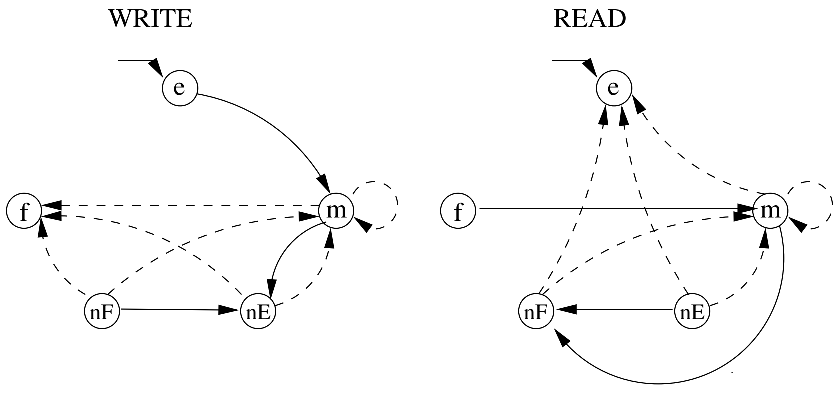


Fig. 9. Minimal *restricted* abstract buffer

2.3 Lifted homomorphism

In some cases, it would be interesting to define a Galois Connection from an homomorphism. In general, it is more intuitive to think in terms of mappings than of Connections and we will see in the next part of the report that the lifting of homomorphisms will save some effort to define symbolic abstractions. To define a Galois Connection from an homomorphism H we proceed as follows:

If we have the concrete domain C and the abstract $\widehat{A}$, then we build the abstract lattice as the power set of abstract values $\mathcal{P}(\widehat{A}\,)$ ordered by the set inclusion operator. Furthermore, we define $\alpha : 2^C \rightarrow 2^{\widehat{A}}$ and $\gamma : 2^{\widehat{A}} \rightarrow 2^C$ as:

- $\alpha(S) = \{H(s) \,|\, s \in S\}$
- $\gamma(\widehat{S}\,) = \{s \,|\, \exists\, \widehat{s} \in \widehat{S} \wedge H(s) = \widehat{s}\}$

Figure 10 displays the abstract lattice of naturals, and the functions as follows:

2.4 MLTSs approximation

In previous sections, we have seen the definition of the minimal abstractions using either an homomorphism or a Galois Connection. However, we can have, as well, non minimal abstractions; let us formalize the approximation relation between MLTSs:

Definition 5 *Given two MLTSs P $(S, A, \rightarrow_\Diamond, \rightarrow_\Box, s_0)$ and Q $(S, A, \rightharpoonup_\Diamond, \rightharpoonup_\Box, s_0)$ built over the same sets of states $\langle S, \preccurlyeq_S\rangle$ and actions $\langle A, \preccurlyeq_A\rangle$; Q is an abstraction of P, denoted by $P \sqsubseteq_\preccurlyeq Q$, if the following conditions hold:*

- $\forall\, s, a, r, s'.\, s \xrightarrow{a}_\Diamond r \wedge s \preccurlyeq_S s' \implies \exists\, a', r'.\, s' \overset{a'}{\rightharpoonup}_\Diamond r' \wedge r \preccurlyeq_S r' \wedge a \preccurlyeq_A a'$
- $\forall\, s', a', r', s.\, s' \overset{a'}{\rightharpoonup}_\Box r' \wedge s \preccurlyeq_S s' \implies \exists\, a, r.\, s \xrightarrow{a}_\Box r \wedge r \preccurlyeq_S r' \wedge a \preccurlyeq_A a'$

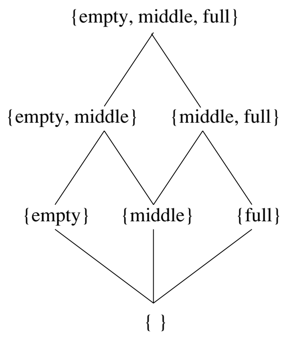


Fig. 10. Lifted Abstract Naturals

$P \sqsubseteq_{\preccurlyeq} Q$ means that Q is more abstract than P and it preserves all the information of the *may*-transitions of P and at least all *must* transitions present in Q are reflected in P. The *may* part of Q is an over-approximation of P and the *must* part is an under-approximation. The refinement relation is the dual of the abstraction.

Note that for the homomorphism approach there is no order defined between states or actions so we substitute $\preccurlyeq$ by $=$. in this case abstractions are done simply by preserving or adding more *may* transitions and by preserving or removing some *must* transitions.

3 Logical Characterization

To express properties about systems we are going to adapt the highly expressive temporal logic (action-based) μ-calculus [19], see also [26], which is defined by the following syntax, where a is an action in A:

$$\varphi ::= T \mid F \mid \neg\,\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [a]\varphi \mid \langle a\rangle\varphi \mid Y \mid \mu Y.\varphi \mid \nu Y.\varphi$$

Formulas are assumed to be monotonic. Following [17], a given formula is interpreted dually over an MLTS, i.e. there will be two sets of states that satisfy it. A set of states that necessarily satisfy the formula and a set of states that possibly satisfy it. Thus, the semantics of the formulas are given by $[\![\varphi]\!] \in 2^S \times 2^S$ and the projections $[\![\varphi]\!]^{nec}$ and $[\![\varphi]\!]^{pos}$ give the first and the second component, respectively. We show below the simultaneous recursive definitions of the evaluation of a state formula. In the state formulas, the propositional context $\rho : Y \rightarrow 2^S \times 2^S$ assigns state sets to propositional variables, and the $\oslash$ operator denotes context overriding. Note that the precision order between action labels plays an important role in the definition of the semantics of the modalities.

$$\begin{array}{rl}
[\![F]\!]_\rho & = \langle \emptyset, \emptyset \rangle \\
[\![T]\!]_\rho & = \langle S, S \rangle \\
[\![\neg\varphi]\!]_\rho & = \langle S \backslash [\![\varphi]\!]_\rho^{pos}, S \backslash [\![\varphi]\!]_\rho^{nec} \rangle \quad \text{(Note the switch of } \mathit{pos} \text{ and } \mathit{nec}\text{)} \\
[\![\varphi_1 \wedge \varphi_2]\!]_\rho & = \langle [\![\varphi_1]\!]_\rho^{nec} \cap [\![\varphi_2]\!]_\rho^{nec}, [\![\varphi_1]\!]_\rho^{pos} \cap [\![\varphi_2]\!]_\rho^{pos} \rangle \\
[\![\varphi_1 \vee \varphi_2]\!]_\rho & = \langle [\![\varphi_1]\!]_\rho^{nec} \cup [\![\varphi_2]\!]_\rho^{nec}, [\![\varphi_1]\!]_\rho^{pos} \cup [\![\varphi_2]\!]_\rho^{pos} \rangle \\
[\![[a]\varphi]\!]_\rho & = \langle \{ s \mid \forall\, r, a'.\, a \preccurlyeq a' \wedge s \xrightarrow{a'}_\Diamond r \Rightarrow r \in [\![\varphi]\!]_\rho^{nec} \}, \\
 & \quad\; \{ s \mid \forall\, r, a'.\, a' \preccurlyeq a \wedge\ s \xrightarrow{a'}_\Box r \Rightarrow r \in [\![\varphi]\!]_\rho^{pos} \} \rangle \\
[\![\langle a \rangle\varphi]\!]_\rho & = \langle \{ s \mid \exists\, r, a'.\, a' \preccurlyeq a \wedge s \xrightarrow{a'}_\Box r \wedge r \in [\![\varphi]\!]_\rho^{nec} \}, \\
 & \quad\; \{ s \mid \exists\, r, a'.\, a \preccurlyeq a' \wedge s \xrightarrow{a'}_\Diamond r \wedge r \in [\![\varphi]\!]_\rho^{pos} \} \rangle \\
[\![Y]\!]_\rho & = \rho(Y) \\
[\![\mu Y.\varphi]\!]_\rho & = \langle \cap \{ S' \subseteq S \mid \Phi_\rho^{nec}(S') \subseteq S' \}, \\
 & \quad\; \cap \{ S' \subseteq S \mid \Phi_\rho^{pos}(S') \subseteq S' \} \rangle \\
[\![\nu Y.\varphi]\!]_\rho & = \langle \cup \{ S' \subseteq S \mid S' \subseteq \Phi_\rho^{nec}(S') \}, \\
 & \quad\; \cup \{ S' \subseteq S \mid S' \subseteq \Phi_\rho^{pos}(S') \} \rangle \\
 & \quad\;\; \text{where } \Phi_\rho^x : 2^S \to 2^S \text{ with } x \in \{nec, pos\},\ \Phi_\rho^x(S') = [\![\varphi]\!]^x_{(\rho \oslash [S'/Y])}
\end{array}$$

We say that a state s necessarily satisfies a formula φ, denoted by $s \models^{nec} \varphi$, iff $s \in [\![\varphi]\!]^{nec}$ and dually s possibly satisfies a formula φ, denoted by $s \models^{pos} \varphi$, iff $s \in [\![\varphi]\!]^{pos}$. We remark that from the semantics of the negation follows:

- s necessarily satisfies $\neg\varphi$ iff s not possibly satisfies φ.
- s possibly satisfies $\neg\varphi$ iff s not necessarily satisfies φ.

We say that a state s necessarily satisfies a formula φ, denoted by $s \models^{nec} \varphi$, iff $s \in [\![\varphi]\!]^{nec}$ and dually s possibly satisfies a formula φ, denoted by $s \models^{pos} \varphi$, iff $s \in [\![\varphi]\!]^{pos}$. We remark that from the semantics of the negation follows:

- s necessarily satisfies $\neg\varphi$ iff s not possibly satisfies φ.
- s possibly satisfies $\neg\varphi$ iff s not necessarily satisfies φ.

It is not difficult to see that if s necessarily satisfies a formula φ then also s possibly satisfies φ. This follows from the fact that every *must*-transition is also a *may*-transition. Without this condition we would be able to prove $s \models^{nec} \varphi$ and $s \models^{nec} \neg\varphi$ for some φ which will leads to an *inconsistent* logic. In fact, it cannot be proved for any φ $s \models^{nec} \varphi$ and also $s \models^{nec} \neg\varphi$, i.e. the necessarily interpretation is consistent and it is always possible to prove $s \models^{pos} \varphi$ or $s \models^{pos}$ $\neg\varphi$ which means that the possibly interpretation is complete. The semantics gives a three valued logic:

- s necessarily satisfies φ.
- s possibly satisfies φ but not necessarily satisfies φ.
- s not possibly satisfies φ.

Note that for any *concrete* MLTS $[\![\varphi]\!]^{nec}$ will be equal to $[\![\varphi]\!]^{pos}$, although if the formula is interpreted over an abstract system the set of states that possibly satisfy the formula but do not necessarily do represents the loss of information caused by the abstraction.

3.1 Property Preservation of MLTSs Approximations

If a system *necessarily* satisfies a property then the formula will hold for all refinements of the system. On the other hand, if a system *possibly* satisfies a property then there exists some refinements for which the formula holds, furthermore all abstractions of the system will also *possibly* satisfy it. This idea is stated in the following lemma:

Lemma 6 *Given two MLTSs P and Q, over the same sets of states and labels S and A, with $P \sqsubseteq_{\preccurlyeq} Q$ then for all p and q in S such that $p \preccurlyeq q$ and for all formula φ then:*

– $q \models^{nec}_{Q} \varphi \Rightarrow p \models^{nec}_{P} \varphi$

– $q \not\models^{pos}_{Q} \varphi \Rightarrow p \not\models^{pos}_{P} \varphi$

This result is useful because, by performing symbolic abstractions (see next section) we generate approximations of the minimal abstraction, so the lemma states that we can still infer the satisfaction/refutation of the properties from the approximation to the original.

Proof:

We present the main part of the proof for the *necessarily* part, the rest of the cases either are trivial or are analogous to the ones included below, all the proof but the part related with the fixpoints has been checked using PVS. The proof is done by induction over the structure of formula and over the propositional context ρ[5]. We denote with $\rightarrow$ the transitions of P and $\rightharpoonup$ the ones of Q.

Universal Case

1. We have to prove that for all p and q such that $p \preccurlyeq q$ if $q \models^{nec}_{Q} [a]\,\varphi$ then $p \models^{nec}_{P} [a]\,\varphi$
2. By Induction Hypothesis we have that for all q and p such that $p \preccurlyeq q$ if $q \models^{nec}_{Q} \varphi$ then $p \models^{nec}_{P} \varphi$
3. $q \models^{nec}_{Q} [a]\,\varphi$ implies $\forall\, q', a'.\, a \preccurlyeq a' \wedge q \xrightharpoonup{a'}_{\diamond} q' \Rightarrow q' \models^{nec}_{Q} \varphi$
4. We want to prove that $\forall\, p', a'.\, a \preccurlyeq a' \wedge p \xrightarrow{a'}_{\diamond} p' \Rightarrow p' \models^{nec}_{P} \varphi$
5. Let's assume a transition $p \xrightarrow{a'}_{\diamond} p'$ and $a \preccurlyeq a'$ then we have to prove $p' \models^{nec}_{P} \varphi$
6. By definition of approximation there exists b and r such that $q \xrightharpoonup{b}_{\diamond} r$ and $p' \preccurlyeq r \wedge a' \preccurlyeq b$
7. We instantiate (3) with r and b and by monotonicity of $\preccurlyeq$ follows that $a \preccurlyeq b$, therefore $r \models^{nec}_{Q} \varphi$
8. The Induction Hypothesis instantiated with p' and r proves the case.

[5] Note that the context is only used in the proofs of the fixpoint operators, therefore, in the rest of the proofs, we do not explicitly represent it.

□ (Case)

Existential Case

1. We have to prove that for all p and q such that $p \preccurlyeq q$ if $q \models^{nec}_{Q} \langle a \rangle\, \varphi$ then $p \models^{nec}_{P} \langle a \rangle\, \varphi$
2. The Induction Hypothesis is the same of the previous case
3. $q \models^{nec}_{Q} \langle a \rangle\, \varphi$ implies $\exists\, q', a'.\, a' \preccurlyeq a \wedge q \xrightarrow{a'}_{\Box} q' \wedge q' \models^{nec}_{Q} \varphi$
4. We want to prove that $\exists\, p', a'.\, a' \preccurlyeq a \wedge p \xrightarrow{a'}_{\Box} p' \wedge p' \models^{nec}_{P} \varphi$
5. Let's assume a transition $q \xrightarrow{a'}_{\Box} q'$ with $a' \preccurlyeq a$ and $q' \models^{nec}_{Q} \varphi$
6. By definition of approximation there exists b and r such that $p \xrightarrow{b}_{\Box} r$ and $r \preccurlyeq q' \wedge b \preccurlyeq a'$
7. We instantiate (4) with r and b and by monotonicity of $\preccurlyeq$ follows that $b \preccurlyeq a$ therefore it rests to prove $r \models^{nec}_{P} \varphi$
8. The Induction Hypothesis instantiated with r and q' proves the case.

□ (Case)

The modal operators for the *possibly* fragment are done in a identical way. We could have defined one operator in function of the other using the standard equivalence: $[a]\varphi$ equals $\neg\langle a \rangle \neg\, \varphi$, however we have included these two proofs because are more illustrative. Now we proceed by presenting one proof for the fixpoint operators, also for the *necessary* part. Remember that $q \models \varphi$ iff $q \in [\![\, \varphi \,]\!]_{\rho}$. And note that, from now on, in order to simplify the notation of the proofs, we are going to use the following shortcut: instead of writing $[\![\, \varphi \,]\!]_{\rho \oslash [S/Y]}$ we will use $[\![\, \varphi \,]\!]_{S}$.

Greatest Fixpoint

1. We have to prove that for all p and q such that $p \preccurlyeq q$, and for all propositional context ρ, if $q \in [\![\, \nu\, Y.\, \varphi \,]\!]_{Q,\rho}$ then $p \in [\![\, \nu\, Y.\, \varphi \,]\!]_{P,\rho}$.
2. $p \in [\![\, \nu\, Y.\, \varphi \,]\!]_{P,\rho}$ implies $p \in \cup\{S \mid S \subseteq [\![\, \varphi \,]\!]_{P,S}\}$ which is equivalent to $\exists\, S.\, S \subseteq [\![\, \varphi \,]\!]_{P,S} \wedge p \in S$
3. By Induction Hypothesis we have for all p and q such that $p \preccurlyeq q$ and for all valuation of the variables if $q \in [\![\, \varphi \,]\!]_{Q,\rho}$ then $p \in [\![\, \varphi \,]\!]_{P,\rho}$
4. Let F be the greatest fixpoint of φ in Q, i.e. $F = [\![\, \nu\, Y.\, \varphi \,]\!]_{Q,\rho}$ then $[\![\, \varphi \,]\!]_{Q,F} = F$
5. By Induction Hypothesis we have:
   (a) $q \in [\![\, \varphi \,]\!]_{Q,F} \implies q \in [\![\, \varphi \,]\!]_{P,F}$ which implies $F = [\![\, \varphi \,]\!]_{Q,F} \subseteq [\![\, \varphi \,]\!]_{P,F}$ and,
   (b) $q \in [\![\, \varphi \,]\!]_{Q,F} \implies p \in [\![\, \varphi \,]\!]_{P,F}$
6. $q \in F$ implies that $q \in [\![\, \varphi \,]\!]_{Q,F}$ therefore by (b) follows $p \in [\![\, \varphi \,]\!]_{P,F}$
7. By monotonicity of $[\![\, \varphi \,]\!]_{P}$, from (a) follows $[\![\, \varphi \,]\!]_{P,F} \subseteq [\![\, \varphi \,]\!]_{P,[\![\, \varphi \,]\!]_{P,F}}$
8. We instantiate S in (2) with $[\![\, \varphi \,]\!]_{P,F}$ then by (6) and (7) the case is proved.

□ (Case)

The rest of the proofs for the fixpoint operators are analogous. But, it will enough to prove one of the operators because the other may be constructed using the negation, i.e, $\mu Y.\, \varphi$ equals $\neg \nu Y. \neg\, \varphi$.

3.2 Property Preservation of MLTSs Abstractions

Since the abstraction of a system preserves some information of the original one, the idea is to prove properties on the abstract and then to infer the result for the original. Since action labels occur in μ-calculus formulas, formulas over A are distinct from formulas over $\widehat{A}$. Therefore, we need to define the meaning of the satisfaction relation of an abstract formula on a concrete state: $[\![\widehat{\varphi}]\!]_\xi$ where ξ is either h_A or α_A depending on whether we use a homomorphism or a Galois Connection. $[\![\widehat{\varphi}]\!]_\xi$ gives the set of concrete states that (*necessarily* or *possibly*) satisfy an abstract formula. An extract of the *necessarily* semantics is given below, note that for the rest of the cases (T, F, $\wedge$, $\vee$ and fixpoints) the definition does not change, and the *possibly* semantics are dual:

$$[\![[\,\widehat{a}\,]\widehat{\varphi}]\!]^{nec}_\xi = \{s \mid \forall\, r, a.\, \widehat{a} \preccurlyeq \xi(\{a\}) \wedge s \xrightarrow{a}_\diamond r \Rightarrow r \in [\![\widehat{\varphi}]\!]^{nec}_\xi\}$$
$$[\![\langle\,\widehat{a}\,\rangle\widehat{\varphi}]\!]^{nec}_\xi = \{s \mid \exists\, r, a.\, \xi(\{a\}) \preccurlyeq \widehat{a} \wedge s \xrightarrow{a}_\Box r \wedge r \in [\![\widehat{\varphi}]\!]^{nec}_\xi\}$$

A concrete state s necessarily satisfies the abstract formula $\widehat{\varphi}$, denoted by $s \models^{nec}_\xi \widehat{\varphi}$, iff $s \in [\![\widehat{\varphi}]\!]^{nec}_\xi$. And dually, $s \models^{pos}_\xi \widehat{\varphi}$, iff $s \in [\![\widehat{\varphi}]\!]^{pos}_\xi$. Now we can give the property preservation result:

Theorem 7 *Let P be the MLTS $(S, A, \rightarrow_\diamond, \rightarrow_\Box, s_0)$, X be either a homomorphism $H = \langle h_S, h_A\rangle$ between (S, A) and $(\widehat{S}, \widehat{A})$ [in which case ξ stands for h] or a Galois Connection $G = \langle(\alpha_S, \gamma_S), (\alpha_A, \gamma_A)\rangle$ between $(\mathcal{P}(S), \mathcal{P}(A))$ and $(\widehat{S}, \widehat{A})$ [in which case ξ stands for α] and let $\widehat{P}\!\downarrow$ (over $\widehat{S}$ and $\widehat{A}$) be the minimal (restricted) abstraction of P. And finally let $\widehat{\varphi}$ be a formula over $\widehat{A}$, then for all $p \in S$ and $\widehat{p} \in \widehat{S}$ such that $\xi(\{p\}) \preccurlyeq \widehat{p}$:*

- $\widehat{p} \models^{nec} \widehat{\varphi} \Rightarrow p \models^{nec}_\xi \widehat{\varphi}$
- $\widehat{p} \not\models^{pos} \widehat{\varphi} \Rightarrow p \not\models^{pos}_\xi \widehat{\varphi}$

The proof follows from the fact that every *may* trace of P is mimicked on $\widehat{P}$ by some related states and, on the other hand, every *must* trace of $\widehat{P}$ is present in P. Next, we present the proof the Galois Connection approach, the homomorphism case is simpler. As in the previous proofs we use induction over the structure of the formula and propositional context:

Proof:

Negation *Necessary* part

1. We have to prove $\widehat{p} \in [\![\neg\,\widehat{\varphi}]\!]^{nec}_{\widehat{\rho}} \implies p \in [\![\neg\,\widehat{\varphi}]\!]^{nec}_{\rho,\alpha}$.
2. $\widehat{p} \in [\![\neg\,\widehat{\varphi}]\!]^{nec}_{\widehat{\rho}} \iff \widehat{p} \in \widehat{S}/[\![\,\widehat{\varphi}]\!]^{pos}_{\widehat{\rho}}$

3. $p \in [\![\neg \widehat{\varphi}]\!]^{nec}_{\rho,\alpha} \iff p \in S/[\![\widehat{\varphi}]\!]^{pos}_{\rho,\alpha}$
4. By Induction Hypothesis we have $p \in [\![\widehat{\varphi}]\!]^{pos}_{\rho,\alpha} \implies \widehat{p} \in [\![\widehat{\varphi}]\!]^{pos}_{\widehat{\rho}}$ which implies $p \notin S/[\![\widehat{\varphi}]\!]^{pos}_{\rho,\alpha} \implies \widehat{p} \notin \widehat{S}/[\![\widehat{\varphi}]\!]^{pos}_{\widehat{\rho}}$
5. From (4) follows $\widehat{p} \in \widehat{S}/[\![\widehat{\varphi}]\!]^{pos}_{\widehat{\rho}} \implies p \in S/[\![\widehat{\varphi}]\!]^{pos}_{\rho,\alpha}$ which trivially proves the case

□ (Case)

Universal operator *Necessary* part

1. We have to prove $\widehat{p} \models [\widehat{a}]\,\widehat{\varphi}$ then $p \models_\alpha [\widehat{a}]\,\widehat{\varphi}$ for all p such that $\alpha(\{p\}) \preccurlyeq \widehat{p}$
2. By Induction Hypothesis we have that for all $\widehat{p}$ and p such that $\alpha(\{p\}) \preccurlyeq \widehat{p}$ if $\widehat{p} \models \widehat{\varphi}$ then $p \models_\alpha \widehat{\varphi}$
3. $\widehat{p} \models [\widehat{a}]\,\widehat{\varphi}$ implies $\widehat{p} \in \{\widehat{s} \mid \forall\,\widehat{r}, \widehat{a}'.\,\widehat{a} \preccurlyeq \widehat{a}' \wedge \widehat{s} \xrightarrow{\widehat{a}'}_\diamond \widehat{r} \Rightarrow \widehat{r} \models \widehat{\varphi}\}$
4. From above follows $\forall\,\widehat{r}, \widehat{a}'.\,\widehat{a} \preccurlyeq \widehat{a}' \wedge \widehat{p} \xrightarrow{\widehat{a}'}_\diamond \widehat{r} \Rightarrow \widehat{r} \models \widehat{\varphi}$
5. We want to prove that $p \in \{s \mid \forall\, r, a.\,\widehat{a} \preccurlyeq \alpha(\{a\}) \wedge s \xrightarrow{a}_\diamond r \Rightarrow r \models_\alpha \widehat{\varphi}\}$
6. Let us assume the following transition $p \xrightarrow{a}_\diamond r$ and $\widehat{a} \preccurlyeq \alpha(\{a\})$ then we have to prove $r \models_\alpha \widehat{\varphi}$
7. If there exists the transition $\widehat{p} \xrightarrow{\alpha(\{a\})}_\diamond \alpha(\{r\})$ then:
   - From (4) and (6) we will have $\alpha(\{r\}) \models \widehat{\varphi}$
   - By Induction Hypothesis we have $r \models_\alpha \widehat{\varphi}$ which would finish the proof.
8. Therefore, we need to prove the existence of such transition, i.e. $\widehat{p} \xrightarrow{\alpha(\{a\})}_\diamond \alpha(\{r\})$ is in the minimal *restricted* system, in other words it is enough to prove that the pair $(\{r\}, \{a\})$ is in M^{min}_{may}
9. From $p \xrightarrow{a}_\diamond r$ follows that $(\{r\}, \{a\})$ is in M_{may}.
10. There is not any pair (R', B') in M_{may} with $R' \subset \{r\}$ or $B' \subset \{a\}$ which implies that $(\{r\}, \{a\}) \in M^{min}_{may}$ and this proves the case.

□ (Case)

Existential operator *Necessary* part

1. We have to prove $\widehat{p} \models \langle\widehat{a}\rangle\,\widehat{\varphi}$ then $p \models_\alpha \langle\widehat{a}\rangle\,\widehat{\varphi}$ for all p such that $\alpha(\{p\}) \preccurlyeq \widehat{p}$
2. The Induction Hypothesis is equal to one of the previous case.
3. $\widehat{p} \models \langle\widehat{a}\rangle\,\widehat{\varphi}$ implies $\widehat{p} \in \{\widehat{s} \mid \exists\,\widehat{r}, \widehat{a}'.\,\widehat{a}' \preccurlyeq \widehat{a} \wedge \widehat{s} \xrightarrow{\widehat{a}'}_\Box \widehat{r} \wedge \widehat{r} \models \widehat{\varphi}\}$
4. Let us assume the transition $\widehat{p} \xrightarrow{\widehat{a}'}_\Box \widehat{r}$ with $\widehat{a}' \preccurlyeq \widehat{a}$ and $\widehat{r} \models \widehat{\varphi}$
5. We want to prove that $\exists\, r, a.\,\alpha(\{a\}) \preccurlyeq \widehat{a} \wedge p \xrightarrow{a}_\Box r \wedge r \models_\alpha \widehat{\varphi}$
6. From (4) and by definition of minimal *restricted* abstraction there exists a pair (R, B) in M^{min}_{must} with $\alpha(R) = \widehat{r}$ and $\alpha(B) = \widehat{a}$
7. (R, B) in M^{min}_{must} implies (R, B) in M_{must}
8. By definition of M_{must} and from (7) follows that it exists r and a with $p \xrightarrow{a}_\Box r$ and $r \in R$ and $a \in B$.
9. By monotonicity of α follows that $\alpha(\{a\}) \preccurlyeq \alpha(B) = \widehat{a}$ and $\alpha(\{r\}) \preccurlyeq \alpha(R) = \widehat{r}$

10. We instantiate (5) with a and r and applying the Induction Hypothesis concludes the proof

$\Box$ (Case)

The proofs for the *possibly* part are analogous to the ones presented. Furthermore the operators $T, F, \wedge$ and $\vee$ are trivial. In order to proceed with the rest, we are going to introduce some extra definitions that will make easier the proofs for the fixpoint operators. First, let us define the following two functions $\dot{\alpha} : 2^S \rightarrow 2^{\widehat{S}}$ and $\dot{\gamma} : 2^{\widehat{S}} \rightarrow 2$ defined as follows:

- $\dot{\alpha}(S)$ as the the application of α to the elements of S, i.e. $\{\alpha(s) \mid s \in S\}$
- $\dot{\gamma}\,(\widehat{S})$ as the union of the application of γ to the elements of $\widehat{S}$, i.e. $\cup\{\gamma(\widehat{s}) \mid \widehat{s} \in \widehat{S}\}$

The following proposition presents the properties of the functions that are used during the proof:

Proposition 8 *The pair $(\langle\dot{\alpha}, \subseteq\rangle, \langle\dot{\gamma}, \subseteq\rangle)$ has the following properties:*

- $\dot{\alpha}$ *monotonic*
- $\dot{\gamma}$ *monotonic*
- $\dot{\alpha}$ *distributes over* $\cup$
- $\dot{\gamma}$ *distributes over* $\cup$

Proofs: (proposition)

- $\dot{\alpha}$ monotonic
  1. To Prove: if $S \subseteq S'$ then $\dot{\alpha}(S) \subseteq \dot{\alpha}(S')$
  2. For all s if $s \in S$ implies $s \in S'$ and $\alpha(\{s\}) \in \dot{\alpha}(S)$ implies $\alpha(\{s\}) \in \dot{\alpha}(S')$ which proves the property.
- $\dot{\gamma}$ monotonic
  1. To Prove: if $\widehat{S} \subseteq \widehat{S}'$ then $\dot{\gamma}(\widehat{S}) \subseteq \dot{\gamma}(\widehat{S}')$
  2. For all $\widehat{s}$ if $\widehat{s} \in \widehat{S}$ implies $\widehat{s} \in \widehat{S}'$ and $\gamma(\widehat{s}) \in \dot{\gamma}(\widehat{S})$ implies $\gamma(\widehat{s}) \in \dot{\gamma}(\widehat{S}')$ which proves the condition.
- $\dot{\alpha}$ distributes over $\cup$
  1. To prove $\dot{\alpha}(\cup_i S_i) = \cup_i\{\dot{\alpha}(S_i)\}$
  2. $\widehat{s} \in \dot{\alpha}(\cup_i S_i)$ if and only if $\exists\, s.\alpha(\{s\}) = \widehat{s} \wedge s \in \cup_i S_i$
  3. $s \in \cup_i S_i$ if and only if $\exists\, S.s \in S$
  4. Therefore $\widehat{s} \in \dot{\alpha}(\cup_i S_i) \iff \exists\, S, s.s \in S \,\wedge\, \alpha(\{s\}) = \widehat{s}$
  5. $\widehat{s} \in \cup_i\{\dot{\alpha}(S_i)\}$ if and only if $\exists\, S.\, \dot{\alpha}(S) \in \cup_i\{\dot{\alpha}(S_i)\} \,\wedge\, \widehat{s} \in \dot{\alpha}(S)$
  6. $\widehat{s} \in \dot{\alpha}(S)$ if and only if $\exists\, s \in S.\alpha(\{s\}) = \widehat{s}$
  7. Therefore $\widehat{s} \in \cup_i\{\dot{\alpha}(S_i)\} \iff \exists\, S, s.s \in S \,\wedge\, \alpha(\{s\}) = \widehat{s}$
  8. The property follows from (4) and (7).
- $\dot{\gamma}$ distributes over $\cup$
  1. To prove $\dot{\gamma}\,(\cup_i \widehat{S}_i) = \cup_i\{\dot{\gamma}\,(\widehat{S}_i)\}$
  2. $s \in \dot{\gamma}\,(\cup_i \widehat{S}_i)$ if and only if $\exists\, \widehat{s}, \widehat{S}.\widehat{s} \in \widehat{S} \,\wedge\, s \in \gamma(\widehat{s})$

3. $s \in \cup_i\{\dot{\gamma}\,(\widehat{S}_i)\}$ if and only if $\exists\, \widehat{S}.s \in \dot{\gamma}\,(\widehat{S}) \,\wedge\, \dot{\gamma}\,(\widehat{S}) \subseteq \cup_i\{\dot{\gamma}\,(\widehat{S}_i)\}$
4. $s \in \dot{\gamma}\,(\widehat{S})$ if and only if $\exists\, \widehat{s} \in \widehat{S} \,\wedge\, s \in \gamma\widehat{s}$
5. The property follows from (2) and (4).

□ (proposition)

We are going to use the following auxiliary lemma.

Lemma 9 *Considering the same definitions of the previous theorem. If we define $\widehat{\rho}(Y)$ with $Y \mapsto \widehat{S}$ as $\{\widehat{r} \mid \exists\, \widehat{s} \in \widehat{S}.\widehat{r} \preccurlyeq \widehat{s}\}$ and $\rho\,(Y)$ as $\dot{\gamma}\,(\widehat{\rho}\,(Y))$. Then for all p and for all valuation of the variables:*

– $\dot{\gamma}([\![\, \widehat{\varphi}'\,]\!]^{nec}_{\widehat{\rho}}) \subseteq [\![\, \widehat{\varphi}'\,]\!]^{nec}_{\rho,\alpha}$

– $\dot{\alpha}([\![\, \widehat{\varphi}'\,]\!]^{pos}_{\rho,\alpha}) \subseteq [\![\, \widehat{\varphi}'\,]\!]^{pos}_{\widehat{\rho}}$

implies

– $\alpha(\{p\}) \in [\![\, \widehat{\varphi}'\,]\!]^{nec}_{\widehat{\rho}} \Rightarrow p \in [\![\, \widehat{\varphi}'\,]\!]^{nec}_{\rho,\alpha}$

– $p \in [\![\, \widehat{\varphi}'\,]\!]^{pos}_{\rho,\alpha} \Rightarrow \alpha(\{p\}) \in [\![\, \widehat{\varphi}'\,]\!]^{pos}_{\widehat{\rho}}$

Proof: (lemma)

Necessarily part

– $\alpha(\{p\}) \in [\![\, \widehat{\varphi}'\,]\!]^{nec}_{\widehat{\rho}}$ implies $p \in \gamma(\alpha(\{p\})) \subseteq \dot{\gamma}\,([\![\, \widehat{\varphi}'\,]\!]^{nec}_{\widehat{\rho}})$
– $\dot{\gamma}([\![\, \widehat{\varphi}'\,]\!]^{nec}_{\widehat{\rho}}) \subseteq [\![\, \widehat{\varphi}'\,]\!]^{nec}_{\rho,\alpha}$ implies $p \in [\![\, \widehat{\varphi}'\,]\!]^{nec}_{\rho,\alpha}$ which proves the case.

Possibly part

– $p \in [\![\, \widehat{\varphi}'\,]\!]^{pos}_{\rho,\alpha}$ implies $\alpha(\{p\}) \in \dot{\alpha}([\![\, \widehat{\varphi}'\,]\!]^{pos}_{\rho,\alpha}) \subseteq [\![\, \widehat{\varphi}'\,]\!]^{pos}_{\widehat{\rho}}$ which proves the case.

□ (lemma)

Lemma 9 only implies Theorem 7 for the case that $\widehat{p} = \alpha(\{p\})$. However we can use this Lemma in combination with the previous result (Lemma 6) to prove the full Theorem. Therefore, we only need to prove the requirements of Lemma 9 to finish the rest of the proof. Let us present first the simple case:

Variable Valuation *Necessary* part

1. We have to prove $\dot{\gamma}([\![Y]\!]_{\widehat{\rho}}) \subseteq [\![Y]\!]_{\rho,\alpha}$ where Y is the set of propositional variables
2. $[\![Y]\!]_{\widehat{\rho}} = \widehat{\rho}\,(Y)$
3. $[\![Y]\!]_{\rho,\alpha} = \dot{\gamma}(\widehat{\rho}\,(Y))$ which trivially proves the case.

□ (Case)

Variable Valuation *Possibly* part

1. We have to prove $\dot{\alpha}([\![Y]\!]^{pos}_{\rho,\alpha}) \subseteq [\![Y]\!]^{pos}_{\widehat{\rho}}$
2. $[\![Y]\!]_{\rho,\alpha} = \dot{\gamma}(\widehat{\rho}(Y))$
3. $[\![Y]\!]_{\widehat{\rho}} = \widehat{\rho}(Y)$ then we have to prove: $\dot{\alpha}(\dot{\gamma}(\widehat{\rho}(Y))) \subseteq \widehat{\rho}$
4. $\dot{\alpha}(\dot{\gamma}(\widehat{\rho}(Y))) = \{\alpha(\gamma(\widehat{s})) \mid \widehat{s} \in \widehat{\rho}(Y)\})$)
5. By definition of $\widehat{\rho}$, for all $\widehat{r}$ such that $\widehat{r} \preccurlyeq \widehat{s}$, if $\widehat{s} \in \widehat{\rho}(Y)$ then $\widehat{r} \in \widehat{\rho}(Y)$ therefore considering that $\alpha(\gamma(\widehat{s})) \preccurlyeq \widehat{s}$ it if $\widehat{s} \in \widehat{\rho}(Y)$ then $\alpha(\gamma(\widehat{s})) \in \widehat{\rho}(Y)$, hence: $\{\alpha(\gamma(\widehat{s})) \mid \widehat{s} \in \widehat{\rho}(Y)\} \subseteq \{\widehat{s} \mid \widehat{s} \in \widehat{\rho}(Y)\} = \widehat{\rho}(Y)$ which proves the case.

$\square$ (Case)

We only present the proofs for the greatest fixpoint operator; $\mu Y.\varphi$ can always be replaced by $\neg(\nu.Y \neg\varphi)$ therefore it is enough to prove one the operators:

Greatest Fixpoint: *Necessary* part

1. Let us consider the case that $\dot{\gamma}\,([\![\nu Y.\, \widehat{\varphi}']\!]_{\widehat{\rho}}) \subseteq [\![\nu Y.\, \widehat{\varphi}']\!]_{\rho,\alpha}$.
2. The Induction Hypothesis, for all valuation of the variables: $\dot{\gamma}\,([\![\widehat{\varphi}']\!]_{\widehat{\rho}}) \subseteq [\![\widehat{\varphi}']\!]_{\rho,\alpha}$
3. If $\widehat{S} \subseteq [\![\widehat{\varphi}']\!]_{\widehat{S}}$ then $\dot{\gamma}\,(\widehat{S}) \subseteq \dot{\gamma}\,([\![\widehat{\varphi}']\!]_{\widehat{S}})$
4. Le us instantiate the Induction Hypothesis with $\widehat{S}$, then we have:
   $\dot{\gamma}\,([\![\widehat{\varphi}']\!]_{\widehat{S}}) \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}\,(\widehat{S}),\alpha}$
5. By transitivity, we have:
   $\dot{\gamma}\,(\widehat{S}) \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}\,(\widehat{S}),\alpha}$
6. For all $\widehat{S}$ if $\widehat{S} \subseteq [\![\widehat{\varphi}']\!]_{\widehat{S}}$ then $\dot{\gamma}\,(\widehat{S}) \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}\,(\widehat{S}),\alpha}$ implies:
   $[\![\nu Y.\, \widehat{\varphi}']\!]_{\widehat{\rho}} = \cup\{\widehat{S} \mid \widehat{S} \subseteq [\![\widehat{\varphi}']\!]_{\widehat{S}}\} \subseteq \cup\{\widehat{S} \mid \dot{\gamma}\,(\widehat{S}) \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}\,(\widehat{S}),\alpha}\}$
7. $\dot{\gamma}\,(\cup\{\widehat{S} \mid \dot{\gamma}\,(\widehat{S}) \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}\,(\widehat{S}),\alpha}\}) \subseteq \cup\{\dot{\gamma}\,(\widehat{S}) \mid \dot{\gamma}\,(\widehat{S}) \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}\,(\widehat{S}),\alpha}\})$
8. $\cup\{\dot{\gamma}\,(\widehat{S}) \mid \dot{\gamma}\,(\widehat{S}) \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}\,(\widehat{S}),\alpha}\}) \subseteq \cup\{S \mid S \subseteq [\![\widehat{\varphi}']\!]_{S,\alpha}\}$
9. By Transitivity: $\dot{\gamma}\,([\![\nu Y.\, \widehat{\varphi}']\!]_{\widehat{\rho}} \subseteq \cup\{S \mid S \subseteq [\![\widehat{\varphi}']\!]_{S,\alpha}\} = [\![\nu Y.\, \widehat{\varphi}']\!]_{\rho,\alpha}$ which proves the case.

$\square$ (Case)

Greatest Fixpoint: *Possibly* part

1. Let us consider the case that $\dot{\alpha}([\![\nu Y.\, \widehat{\varphi}']\!]_{\rho,\alpha}) \subseteq [\![\nu Y.\, \widehat{\varphi}']\!]_{\widehat{\rho}}$.
2. The Induction Hypothesis:
   For all valuation of the variables: $\dot{\alpha}([\![\widehat{\varphi}']\!]_{\rho,\alpha}) \subseteq [\![\widehat{\varphi}']\!]_{\widehat{\rho}}$
3. If $S \subseteq [\![\widehat{\varphi}']\!]_{S,\alpha}$ then by monotonicity of $[\![\widehat{\varphi}']\!]$ and considering that $S \subseteq \dot{\gamma}(\dot{\alpha}(S))$ we have that:
   $S \subseteq [\![\widehat{\varphi}']\!]_{\dot{\gamma}(\dot{\alpha}(S)),\alpha}$
4. By monotonicity of $\dot{\alpha}$ we have:
   $\dot{\alpha}\,(S) \subseteq \dot{\alpha}\,([\![\widehat{\varphi}']\!]_{\dot{\gamma}(\dot{\alpha}(S)),\alpha})$

5. Le us instantiate the Induction Hypothesis with $\dot{\alpha}(S)$, then we have:
   $\dot{\alpha}([\![\,\widehat{\varphi}'\,]\!]_{\dot{\gamma}(\dot{\alpha}(S)),\alpha}) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}$
6. By transitivity, we have:
   $\dot{\alpha}\,(S) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}$
7. For all S if $S \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\rho,\alpha}$ then $\dot{\alpha}\,(S) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}$ implies:
   $[\![\,\nu\, Y.\,\widehat{\varphi}'\,]\!]_{\rho,\alpha} = \cup\{S \mid S \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\rho,\alpha}\} \subseteq \cup\{S \mid \dot{\alpha}\,(S) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}\}$
8. By monotonicity of $\dot{\alpha}$:
   $\dot{\alpha}([\![\,\nu\, Y.\,\widehat{\varphi}'\,]\!]_{\rho,\alpha}) \subseteq \dot{\alpha}(\cup\{S \mid \dot{\alpha}\,(S) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}\})$
9. By distributivity of $\dot{\alpha}$ over $\cup$ we have:
   $\dot{\alpha}(\cup\{S \mid \dot{\alpha}\,(S) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}\}) = \cup\{\{\dot{\alpha}(S)\} \mid \dot{\alpha}\,(S) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}\})$
10. $\cup\{\{\dot{\alpha}(S)\} \mid \dot{\alpha}\,(S) \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\dot{\alpha}\,(S)}\}) \subseteq \cup\{\widehat{S} \mid \widehat{S} \subseteq [\![\,\widehat{\varphi}'\,]\!]_{\widehat{S}}\} = [\![\,\nu\, Y.\,\widehat{\varphi}'\,]\!]_{\widehat{\rho}}$
11. By transitivity we prove the case.

□ (Case)

□ (Theorem)

The main part of the proofs has been checked using PVS, the parts not included are the ones concerning the fixpoint operators.

The theorem states that we can infer the satisfaction of a formula on a concrete system if it is *necessarily* satisfied on the abstract. Furthermore, if the formula is not *possibly* satisfied on the abstract it will not hold on the concrete either, so it can be refuted. For example, the two presented abstractions (by homomorphism and by Galois Connection) prove: *"It is possible to write something if the buffer is empty"* expressed as $e \models^{nec} \langle\widehat{w}\rangle T$, which means that in the concrete system s_0 either satisfies $\langle w(0)\rangle T$ or $\langle w(1)\rangle T$ or $\langle w(2)\rangle T$ or ... Furthermore, we can prove on the abstract $f \not\models^{pos} \langle\widehat{w}\rangle T$ which means that in the concrete, all the corresponding concrete states satisfy neither $\langle w(0)\rangle T$ nor $\langle w(1)\rangle T$ nor $\langle w(2)\rangle T$ nor ...

In general, abstractions produced by using Galois Connections preserve more information than the ones generated by homomorphisms, however the state space reduction is stronger in the latter case. For example, the Galois Connection abstraction can prove: *"After every data inserted in the buffer it is possible to read something back"*, so every state *necessarily* satisfies:

$$[\widehat{w}]\langle\widehat{r}\rangle T$$

The property states that after every *may* write transition the system gets in a state in which a *must* read transition is possible. With this property we can infer the following behaviors to the concrete:

– $s \overset{W(0)}{\rightarrow} s' \overset{R(0)}{\rightarrow}$ ... OR $s \overset{W(0)}{\rightarrow} s' \overset{R(1)}{\rightarrow}$ ... OR ...

AND

– $s \overset{W(1)}{\rightarrow} s' \overset{R(0)}{\rightarrow}$ ... OR $s \overset{W(1)}{\rightarrow} s' \overset{R(1)}{\rightarrow}$ ... OR ...

AND

– ...

Another important property that can be proved with the Galois Connection technique but not with the proposed mapping is the *absence of deadlock* since there is a *must* transition from every state that *may* be reached, however the abstraction done by the homomorphism is not able to prove the property for there is no *must* transition from the state *middle* so we cannot infer the existence of some transition from the related states in the concrete system. This property is expressed as:

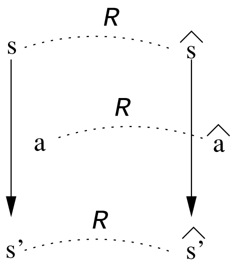
$$\nu X.[T]X \wedge \langle T\rangle T$$

3.3 Abstractions as simulation relations

We have adapted two widely used abstraction techniques for transition systems but remark that the general notion behind both methods is that of simulation relation as is shown, a.o, in [25]. First let us define the relation for LTSs:

Definition 10 *Given two labelled transition systems $P = (S, A, \rightarrow, s_0)$ and $Q = (\widehat{S}, \widehat{A}, \rightarrow, \widehat{s_0})$, a simulation is a binary relation $\mathcal{R}$ on states and actions labels, written $P \lesssim Q$ if $s_0 \mathcal{R} \widehat{s_0}$ and for all $s \in S$ and $\widehat{s} \in \widehat{S}$:*

– *if $s \mathcal{R} \widehat{s}$ and exists a state $r \in S$ and an action label $a \in A$ such that $s \xrightarrow{a} r$ then there exists $\widehat{r} \in \widehat{S}$ and $\widehat{a} \in \widehat{A}$ such that $\widehat{s} \xrightarrow{\widehat{a}} \widehat{r}$ and $s' \mathcal{R} \widehat{r}$ and $a \mathcal{R} \widehat{a}$*



The intuition of the simulation relation is that every transition in P can be mimicked by a transition in Q. Let us consider, now, the *Modal Labelled Transition Systems* $C = (S, A, \rightarrow_\diamond, \rightarrow_\Box, s_0)$ and $A = (\widehat{S}, \widehat{A}, \rightarrow_\diamond, \rightarrow_\Box, \widehat{s_0})$, which is an abstract version of the first one ($C \sqsubseteq_\preccurlyeq A$). Then we can prove following relations:

– Let A_{must} be $(\widehat{S}, \widehat{A}, \rightarrow_\Box, \widehat{s_0})$ and C_{must} be $(S, A, \rightarrow_\Box, s_0)$ then $A_{must} \lesssim C_{must}$.

– Let A_{may} be $(\widehat{S}, \widehat{A}, \rightarrow_\diamond, \widehat{s_0})$ and C_{may} be $(S, A, \rightarrow_\diamond, s_0)$ then $C_{may} \lesssim A_{may}$.

If the abstraction is done using an homomorphism then the simulation relation $\mathcal{R}$ is given by:

– For all $s \in S$ and $\widehat{s} \in \widehat{S}$, $s\,\mathcal{R}\,\widehat{s}$ if and only if $h_S(s) = \widehat{s}$.
– For all $a \in A$ and $\widehat{a} \in \widehat{A}$, $a\,\mathcal{R}\,\widehat{a}$ if and only if $h_A(a) = \widehat{a}$.

In the same way we can construct as well a simulation relation $\mathcal{R}$ from a Galois Connection. In this case $\mathcal{R}$ is given by:

– For all $s \in S$ and $\widehat{s} \in \widehat{S}$, $s\,\mathcal{R}\,\widehat{s}$ if and only if $s \in \gamma(\widehat{s}\,)$.
– For all $a \in A$ and $\widehat{a} \in \widehat{A}$, $a\,\mathcal{R}\,\widehat{a}$ if and only if $a \in \gamma(\widehat{a}\,)$.

These results match with the idea that the *must* part of the abstraction is an under approximation of the concrete system and the *may* part an over approximation.

We conclude the introduction to the abstraction of *transition systems*, the following section is dedicated to explain how to compute abstract approximations directly from μCRL models.

4 Process Abstraction

μCRL [15] is a combination of process algebra [1, 13] and abstract data types. Data is represented by an *algebraic specification* $\Omega = (\Sigma, E)$, in which Σ denotes a many-sorted *signature* (S, F), where is S the set of sorts and F the set of functions, and E a set of Σ-*equations*, see [16]. All specifications must include the boolean data type with the constants true and false (`T` and `F`). From process algebra μCRL inherits the following operators:

– `p.q` perform p and then perform q;
– `p + q` perform arbitrarily either p or q;
– $\sum_{d:D}$ `p(d)` perform $p(d)$ with an arbitrarily chosen d of sort D;
– `p` $\lhd$ `b` $\rhd$ `q` if b is true, perform p, otherwise perform q;
– `p || q` run processes p and q in parallel.
– δ stands for deadlock.

Atomic actions may have data parameters. The operator $|$ (`comm`) allows synchronous parameterized communication.

If two actions are able to synchronize we can force that they occur always in communication using the operator **encap** (∂_H). The operator τ_I hides enclosed actions by renaming into τ actions. The initial behavior of the system can be specified with the keyword **init** followed by a process term:

System $= \tau_I \partial_H(p_0 \parallel p_1 \parallel ...)$
init System

The following μCRL code correspond to a buffer of size N, value stored in the global variable. We see that the specification is composed by three simple processes representing a writer a producer and the consumer. Note that we simplify the system by no representing the value of the written data, the buffer process is just a counter determining the amount of entries currently stored. We assume the correct definition of the *booleans* and the *naturals* in which `S` corresponds to the successor, `P` corresponds to the predecessor, `lt` and `gt` to $<$ and $>$ respectively.

act Write
Read

map N:→Nat

proc buffer(n:Nat) =
Write.buffer(S(n)) $\lhd$ lt(n, N) $\rhd$ δ +
Read.buffer(P(n)) $\lhd$ gt(0, n) $\rhd$ δ
init buffer(0)

Every μCRL system can be transformed to a special format, called *Linear Process Equation* or *Operator*. An LPE (see definition below) is a single μCRL process which represents the complete system and from which communication and parallel composition operators have been eliminated.

$$X(d:D) = \sum_{i \in I} \sum_{e_i:E_i} a_i(f_i[d,e_i]).X(g_i[d,e_i]) \lhd c_i[d,e_i] \rhd \delta \qquad (1)$$

In the definition, d denotes a vector of parameters d of type D that represents the state of the system at every moment. We use the keyword *init* to declare the initial vector of values of d. The process is composed by a finite number I of summands, every summand i, has a list of local variables e_i, of possibly infinite domains, and it is of the following form: a condition $c_i[d,e_i]$, if the evaluation of the condition is true the process executes the action a_i with the parameter $f_i[d,e_i]$ and will move to a new state $g_i[d,e_i]$, which is a vector of terms of type D. $f_i[d,e_i], g_i[d,e_i]$ and $c_i[d,e_i]$ are terms built recursively over variables $x \in [d,e_i]$, applications of function over terms $t = f(t')$ and vectors of terms. To every LPE specification corresponds a labelled transition system. The semantics of the system described by an LPE are given by the following rules:

To every LPE specification corresponds a labelled transition system. The semantics of the system described by an LPE are given by the following rules:

- $s_0 = \mathrm{init}_{lpe}$
- $s \xrightarrow{a} s'$ iff exists $i \in I$ and exists $e : E_i$ such that $c_i[s,e] = \mathtt{T}$, $a_i(f_i[s,e]) = a$ and $g_i[s,e] = s'$

Terms are interpreted over the universe of values $\mathcal{D}$. The LTS corresponding to the above LPE can be generated for any finite `N`. Note that the simplification avoids the infinitely branching behavior presented in the previous section. The system may be arbitrarily big depending of the size of the buffer.

In order to abstract μCRL specifications we may define the relations between concrete and abstract values by means of a mapping $H : \mathcal{D} \to \widehat{\mathcal{D}}$ or a Galois Connection $(\alpha : \mathcal{P}(\mathcal{D}) \to \widehat{\mathcal{D}}, \gamma : \widehat{\mathcal{D}} \to \mathcal{P}(\mathcal{D}))$. This idea is explained below.

4.1 Abstract Interpretation of data domains

We start by introducing an abstract operator to relate the concrete data specification and the abstract version: "$\widehat{\ }$": list(Σ) $\to$ list(Σ). In order to keep the data abstraction as general as possible we define the operator from lists of data specifications to abstract counterparts, the only restriction we need to add to maintain the semantics of the process condition is that the booleans are not modified, i.e, $\widehat{Bool} = Bool$.

We overload the syntactic operator "$\widehat{\ }$" to denote also the abstraction of data terms, but before presenting the formal definition let us give some intuition about it. For example, in our buffer specification we may have a function S which computes the successor of a natural number. The abstract version of S may be defined as follows:

- For the homomorphism:
  $\widehat{S}(empty) = middle$, $\widehat{S}(middle) = middle$ or $\widehat{S}(middle) = full$. It will be undefined for $full$.
- For the Galois Connection approach:
  $\widehat{S}(empty) = middle$, $\widehat{S}(middle) = nonEmpty$, $\widehat{S}(nonFull) = nonEmpty$, $\widehat{S}(nonEmpty) = nonEmpty$, $\widehat{S}(full) = \top$, $\widehat{S}(\perp) = \perp$ and $\widehat{S}(\top) = \top$

Abstract interpretation of functions may add non-determinism to the system, for example $\widehat{S}(middle)$ in the homomorphism case may return different values ($middle$ and $full$). Furthermore, not all the sorts of a specification need to be abstracted, for example, a predicate $\widehat{empty?}$ applied to $empty$ will return true however applied to $nonFull$ can be either true or false. To deal with these two considerations, we lift abstract functions to return sets of values. We introduce a new set $\widehat{X}$ of variables of type $\mathcal{P}(\widehat{D})$. So we get $\widehat{\ } : \mathcal{T}_{\vec{D}}(\Sigma, X) \to \mathcal{T}_{\vec{\mathcal{P}}(\widehat{D})}(\Sigma, \widehat{X})$. In the example, for the homomorphism $\widehat{S}(\{middle\}) = \{middle, full\}$ and for the Galois: $\widehat{S}(\{empty\}) = \{nonEmpty\}$ and $\widehat{empty?}(nonFull) = \{T, F\}$.

We could have defined the abstract type of booleans with $\{\perp, \widehat{T}, \widehat{F}, \top\}$ being $\widehat{empty?}(nonFull)$ equals to $\top$, but we wanted to avoid the redefinition of the semantics of the conditions of the processes for the abstract values.

Returning to μCRL, remember that data is represented by an *algebraic specification* that consists of:

- A many-sorted *signature* $\Sigma = (S, F)$, where S is a set of types or sort names, and F a set of function symbols. A function f of some sort S, and with arity n, is typed by $f : S_0 \times \cdots \times S_{n-1} \to S_n$, where $S_0, \ldots, S_{n-1}$ are the sorts of the arguments of f, and S_n the sort of f.

– A set E of *Σ-equations*, which are expressions of the form $s = t$ where s and t are equally typed terms constructed from variables and function symbols in the usual way.

For example, the specification of naturals used on the buffer system is done as follows:

```
sort  Nat
func  0:→Nat
      S:Nat→Nat
map   eq:Nat×Nat→Bool
      P:Nat→Nat
      lt:Nat×Nat→Bool
      gt:Nat×Nat→Bool
var   x,y:Nat
rew   eq(x,x) = T
      eq(0,S(x)) = F
      eq(S(x),0) = F
      eq(S(x),S(y)) = eq(x,y)
      P(S(x)) = x
      lt(x, 0) = F
      lt(0, S(x)) = T
      lt(S(x), S(y)) = lt(x, y)
      gt(x, y)= lt(y,x)
```

The keyword `func` denotes the *constructor* function symbols and `map` is used to declare additional functions for a sort. We define equations using variables (declared after `rew`) to specify the function symbols. The following fragment specifies the mapping H from naturals to abstract naturals.

```
sort  abs_Nat
func  empty, full, middle, undefined:→abs_Nat
map   H:Nat→abs_Nat
var   x:Nat
rew   H(x) = if(eq(x, 0), empty,
        if(eq(x, N), full,
          if(lt(x, N), middle, undefined)))
```

Let us now show a part of the abstract specification of the naturals, note that we assume the existence of the well defined sorts of sets of booleans and set of abstract naturals.

```
map   abs_0:→setOf_abs_Nat
      abs_N:→setOf_abs_Nat
      eq:abs_Nat×abs_Nat→Bool
      S:abs_Nat→setOf_abs_Nat
      P:abs_Nat→setOf_abs_Nat
      lt:abs_Nat×abs_Nat→setOf_Bool
```

```
        gt:abs_Nat×abs_Nat→setOf_Bool
var     x,y:abs_Nat
rew     abs_0 = {empty}
        abs_N = {full}

        S(empty) = {middle}
        S(middle) = {middle, full}
        S(full) = {undefined}

        P(middle) = {middle, empty}
        P(full) = {middle}

        lt(empty,empty) = {F}
        lt(empty,middle) = {T}
        lt(empty,full) = {T}

        lt(middle,empty) = {F}
        lt(middle,middle) = {F, T}
        lt(middle,full) = {T}

        lt(full,x) = {F}

        gt(x, y) = lt(y, x)
```

Not all possible abstract interpretations are correct; in order to generate *safe* abstractions the data terms involved in the specification and their abstract versions have to satisfy a formal requirement, usually called *safety condition*. The condition for the homomorphism function is expressed as follows:

– for all d in $D.\, H(t[\,d\,]) \in \widehat{t}\,[\,\{H(d)\}\,]$

For the Galois Connection case is similar (note that we consider that t point-wisely applies to sets):

– for all $\widehat{d}$ in $\widehat{D}.\, \widehat{t}\,[\,\widehat{d}\,] \succeq \alpha(t\,[\,\gamma(\,\widehat{d}\,)\,]\,)$

In our example we will have to check that all the functions and constants involved in our system satisfy the safety criteria. The functions we must check are:

1. `0`
2. `N`
3. `S(x)`
4. `lt(x, y)`
5. `P(x)`
6. `gt(x, y)`

Let us check some cases for the homomorphic approach:

– $H(0) \overset{?}{\in} \widehat{0}$
  - $H(0)$ equals *empty*

  - $\widehat{0}$ equals $\{empty\}$ implies $\surd$

- $H(S(x)) \overset{?}{\in} \widehat{S}(H(x))$, we proceed by cases:
  - Case x equals 0:
    - $S(0) = 1$ and $H(1) = middle$
    - $H(0) = empty$ and $\widehat{S}(empty) = \{middle\}$
    - $middle \in \{middle\} \Rightarrow \surd$
  - Case $0 < x < N$:
    - $S(x) = y$ and $0 < y \leq N$
    - $H(y) = middle$ or $full$
    - $H(x) = middle$ and $\widehat{S}(middle) = \{middle, full\}$
    - $middle \in \{middle, full\}$ and $full \in \{middle, full\} \Rightarrow \surd$
  - Case x equals N:
    - $S(N) = N + 1$ and $H(N + 1) = undefined$
    - $H(N) = full$ and $\widehat{S}(full) = \{undefined\}$
    - $- \in \{undefined\} \Rightarrow \surd$
  - Case $x > N$:
    - $S(x) = y$ and $y > N$ and $H(x) = undefined$
    - $H(y) = undefined$ and $\widehat{S}(undefined) = \{undefined\}$
    - $undefined \in \{undefined\} \Rightarrow \surd$

The other conditions can be proved in a similar way.

□

4.2 Modal Linear Process Equation

We present now a new format, the *Modal Linear Process Equation* that will be used to represent the symbolic abstraction of an LPE. An MLPE has the following form:

$$X(d : \mathcal{P}(D)) = \sum_{i \in I} \sum_{e_i : E_i} a_i(F_i[d, e_i]).X(G_i[d, e_i]) \lhd C_i[d, e_i] \rhd \delta \qquad (2)$$

The definition is similar to the one of *Linear Process Equation*, the difference is that the state is represented by a list of power sets of abstract values and for every i: C_i returns a non empty set of booleans, G_i a non empty set of states and F_i also a non empty set of action parameters. Actions are parameterized with sets of values, as well. From an MLPE we can generate a *Modal Labelled Transition System* following these semantic rules:

- $S_0 = \text{init}_{mlpe}$
- $S \xrightarrow{A}_{\Box} S'$ if and only if exists $i \in I$ and exists $e \in E_i$ such that $\mathtt{F} \notin C_i[S, e]$, $A = a(F_i[S, e])$ and $S' = G_i[S, e]$
- $S \xrightarrow{A}_{\Diamond} S'$ if exists $i \in I$ and exists $e \in E_i$ such that $\mathtt{T} \in C_i[S, e]$, and $A = a(F_i[S, e])$ and $S' = G_i[S, e]$

In case we use a plain homomorphism (without lifting it to a Galois Connection), we restrict the rules by letting S_0, S, A and S' be only singleton sets. To compute an abstract interpretation of a linear process, we define the operator " $\bar{\ }$ ": $LPE \rightarrow MLPE$ that pushes the abstraction through the process operators till the data part:

$p = X(t)$ then $\bar{p} = X(\widehat{t})$ being X a process name
$p = a(t)$ then $\bar{p} = \bar{a}(\widehat{t})$ being a an action label
$p = p_0 + ... + p_n$ then $\bar{p} = \bar{p_0} + ... + \bar{p_n}$
$p = \delta$ then $\bar{p} = \delta$
$p = p_0.p_1$ then $\bar{p} = \bar{p_0}.\bar{p_1}$
$p = p_l \lhd t_c \rhd p_r$ then $\bar{p} = \bar{p_l} \lhd \widehat{t_c} \rhd \bar{p_r}$
$p = \sum_{e:E} p$ then $\bar{p} = \sum_{\widehat{e}:\widehat{E}} \bar{p}$

MLPEs allow to capture in a uniform way both approaches: Galois Connection and Homomorphism as well as the combination of both consisting in the lifting of a mapping to a Galois Connection. In the example, the abstract values *nonEmpty* and *nonFull* will be captured by $\{middle, full\}$ and $\{empty, middle\}$ respectively. The successor of *nonFull* will be the union of the successor of *empty* and *middle*: $\{middle, full\}$. In this example, the lifting of the homomorphism saves the extra effort of defining abstract functions.

The MLPE below models the buffer example ($\widehat{S}$, $\widehat{P}$ stand for the abstract versions of successor, predecessor and $\widehat{N}$ for the maximal size of the buffer):

proc $\mathsf{X}(\mathsf{n}{:}\mathcal{P}(\mathsf{abs\_Nat})) =$
$\mathsf{W}.\ \mathsf{X}(\widehat{S}(\mathsf{n})) \lhd \widehat{lt}(\mathsf{n},\widehat{N}) \rhd \delta\ +$
$\mathsf{R}.\ \mathsf{X}(\widehat{P}(\mathsf{n})) \lhd \widehat{gt}(\widehat{0},\ \mathsf{n}) \rhd \delta$

Just considering that all functions are pointwisely extended in order to deal with sets of values, the above MLPE can be used equally for any kind of relation between the data domains: homomorphisms, arbitrary Galois Connections and lifted homomorphisms. The following theorem asserts that the abstract interpretation of a linear process produces an abstract approximation of the (restricted) minimal abstraction of the original.

Theorem 11 *Given a Linear Process Equation* lpe *(as defined in (1)), a Modal Linear Process Equation* mlpe *(as defined in (2)) and an abstraction relation* X *between their data domains (where* X *is either a homomorphism or a Galois Connection). Then if:*

1. mlpe *is the abstract interpretation of* lpe (mlpe $= \bar{\text{lpe}}$),
2. mlts *is the* concrete *MLTS generated from* lpe
3. $\widehat{\text{mlts}\downarrow}$ *the (restricted) minimal w.r.t X of* mlts
4. *All pairs (f,F), (g,G) and (c,C) satisfy the safety condition.*

– *Then, the MLTS* ($\widehat{\text{mlts}}$) *generated from* mlpe *is an abstraction of* $\widehat{\text{mlts}\downarrow}$, *i.e*, ($\widehat{\text{mlts}\downarrow} \sqsubseteq_{\preccurlyeq} \widehat{\text{mlts}}$)

$$\begin{tikzcd}
\text{lpe} \arrow[rr] \arrow[d] & & \text{mlts} \arrow[d] \\
\bar{\text{lpe}} \arrow[r] & \widehat{\text{mlts}} \arrow[r, phantom, "\sqsupseteq_{\preccurlyeq}"] & \widehat{\text{mlts}\downarrow}
\end{tikzcd}$$

The proof is done by checking that every *may* transition generated by the abstract *Modal Linear Process Equation* has at least one more precise counterpart in the (restricted) minimal abstraction of the concrete system (and the other way around for the *must* transitions). We show below the proof corresponding to the Galois Connection approach (the homomorphic case is simpler).

Proof: (may part)

1. Let $\widehat{s} \overset{\widehat{a}}{\rightarrow}_{\diamond} \widehat{r}$ be a *may* transition of the minimal restricted abstraction $\widehat{\text{mlts}\downarrow}$.
2. By definition of minimal *restricted* abstraction we will have a transition $s \overset{a}{\rightarrow}_{\diamond} r$ in *lts* with $r \in R$ and $a \in B$ and $\widehat{r} = \alpha(R)$ and $\widehat{a} = \alpha(B)$ with the pair (R, B) in M^{min}_{may}
3. Therefore, we will have a summand in *lpe* that generates the above transition, i.e., exists i and e such that:
   – $T = c_i(s, e)$
   – $a = f_i(s, e)$
   – $r = g_i(s, e)$
4. We want to prove that exists $\widehat{a}'$ and $\widehat{r}'$ such that:
   (a) $\widehat{s} \overset{\widehat{a}'}{\rightarrow}_{\diamond} \widehat{r}'$ can be generated by *mple*
   (b) $\widehat{r} \preccurlyeq \widehat{r}'$
   (c) $\widehat{a} \preccurlyeq \widehat{a}'$
5. Let $\widehat{r}'$ be equal to $G_i(\widehat{s}, \widehat{e})$ and $\widehat{a}'$ be $F_i(\widehat{s}, \widehat{e})$ for one $\widehat{e}$
6. By the safety condition we have: $\widehat{f}\,(\,\widehat{D}\,) \succcurlyeq \alpha(f\,(\,\gamma(\,\widehat{D}\,)\,)\,)$
7. Let:
   – $\widehat{D}$ be equal to the pair $(\widehat{s}, \widehat{e})$
   – $\cup c_i$ be equal to $\{c_i(s, e) | \exists\, s, e.s \in \gamma(\widehat{s})\}$. $\cup c_i$ denotes the possible values of the ith condition.
   – $\cup g$ be equal to $\{g_i(s, e) | \exists\, s, e, i.s \in \gamma(\widehat{s})\}$. $\cup g$ denotes the possible continuations from the states related with $\widehat{s}$
   – $\cup f$ be equal to $\{a_i(f_i(s, e)) | \exists\, s, e, i.s \in \gamma(\widehat{s})\}$. $\cup f$ denotes the possible action labels of the transitions from the states related with $\widehat{s}$
8. In order to prove (a) we need to prove $T \in C_i(\widehat{s}, \widehat{e})$ therefore:
   – $T \in c_i(s, e)$ implies $T \in \cup c_i$ and by the safety condition $\alpha(\cup c_i) \preccurlyeq C_i(\widehat{s}, \widehat{e})$ which implies $T \in C_i(\widehat{s}, \widehat{e})$ with proves the requirement of (a).
9. To prove (b) we proceed as follows:

  - Let $\cup M$ be the union of all sets of M^{min}_{may}
  - Let $\cup R$ bet the first component of $\cup M$
  - Every r in *lts* such that there is a transition $s \rightarrow_{\Diamond} r$ with $s \in \gamma(\widehat{s})$ is in $\cup f$. Therefore, every element of $\cup R$ is in $\cup f$.
  - From above follows that $R \subseteq \cup f$.
  - By the safety condition and the properties of the Galois Connections, we have that $\widehat{r} = \alpha(R) \preccurlyeq \alpha(\cup f) \preccurlyeq F_i(\widehat{s}, \widehat{e}) = \widehat{r}'$ which implies $\surd$
10. To prove (c) we proceed in the same way.

□ (Case)

Proof: (must part)

1. Let $\widehat{s} \xrightarrow{\widehat{a}}_{\Box} \widehat{r}$ be a *must* transition generated by *mlpe*, i.e. exists one i and $\widehat{e}$ such that:
   - $F \notin C_i(\widehat{s}, \widehat{e})$
   - $\widehat{a} = F_i(\widehat{s}, \widehat{e})$
   - $\widehat{r} = G_i(\widehat{s}, \widehat{e})$
2. We have to prove that exists one $\widehat{s} \xrightarrow{\widehat{a}'}_{\Box} \widehat{r}'$ in the minimal *restricted* abstraction such that $\widehat{r}' \preccurlyeq \widehat{r}$ and $\widehat{a}' \preccurlyeq \widehat{a}$
3. In other words, we need to prove that exists (R, B) in M^{min}_{must} such that:
   (a) $\widehat{s} \xrightarrow{\alpha(B)}_{\Box} \alpha(R)$ in the minimal restricted abstraction
   (b) $\alpha(R) \preccurlyeq \widehat{r}$
   (c) $\alpha(B) \preccurlyeq \widehat{a}$
4. Let:
   - $\cup M$ the union of all sets of M^{min}_{must}
   - $\cup R$ and $\cup B$ the first and the second component of $\cup M$
5. By monotonicity of α for every R it holds that $\alpha(R) \preccurlyeq \alpha(\cup R)$. The same holds for B. Then, we are going to prove:
   - $\alpha(\cup R) \preccurlyeq \widehat{r}$
   - $\alpha(\cup B) \preccurlyeq \widehat{a}$
6. By definition, for all (r, b) in $\cup M$ there exists $s \xrightarrow{b}_{\Box} r$ generated by *lpe*
7. As in the *may* case, let $\cup f$ be the union of all f such there is a transition $s \rightarrow_{\Box} f$ generated by *lpe* then by (6) follows that $\alpha(\cup R)) \preccurlyeq \alpha(\cup f)$
8. By the safety condition follows that $\alpha(\cup f) \preccurlyeq \widehat{r}$ therefore $\alpha(\cup R) \preccurlyeq \widehat{r}$.
9. By (5) and (8) follows (b)
10. We repeat steps (7), (8) and (9) in order to prove (c)
11. (a) follows trivially by definition which implies $\surd$

□ (Case)

□ (Theorem)

This proof has been completely checked with PVS. By Theorem 7 and Lemma 6, we can prove (refute) properties for *lpe* by considering *mlpe* directly.

4.3 Abstraction of Data Terms

So far, we have discussed the general framework to extract a safe abstraction from an LPE giving the conditions that the functions involved have to satisfy. We present a particular case in which we push further the abstraction through the data terms. We recall, that a data term is constructed recursively by variables, vector of terms and applications of functions. We can define the abstract interpretation operator over data as follows:

$$\begin{array}{l} t = x \text{ then } \widehat{t} = \widehat{x} \text{ where } x \text{ is a variable} \\ t = [t_0, ..., t_n] \text{ then } \widehat{t} = [\widehat{t_0}, ..., \widehat{t_n}] \\ t = f(t') \text{ then } \widehat{t} = \widehat{f}(\widehat{t'}) \end{array}$$

For every function $f : D \to D$ we need to specify $\widehat{f} : \widehat{D} \to \mathcal{P}(\widehat{D})$, furthermore we have to provide also the pointwise lifting of the functions, i.e, $\widehat{f} : \mathcal{P}(\widehat{D}) \to \mathcal{P}(\widehat{D})$ with $\widehat{f}(\widehat{X})$ equals to $\cup\{\widehat{f}(\widehat{x}) \,|\, \widehat{x} \in \widehat{X}\}$. One of the benefits of the Galois Connections is that they permit to easily define the most precise functions, as follows:

- $\widehat{f}(\widehat{d}) = \alpha(f(\gamma(\widehat{d})))$

Instead proving the safety criteria for every guard, action and next in a process specification we can prove in general that the abstract data specification satisfies the "safety conditions" and then infer that any particular system does as well. This would allow to reuse abstract specifications of the data into different systems and create libraries of abstractions. The following lemma presents this idea:

Lemma 12 *For every algebraic specification $\Omega = (\Sigma, E)$ with $\Sigma = (D, F)$ and for every abstract algebraic specification $\widehat{\Omega} = (\widehat{\Sigma}, \widehat{E})$ with $\widehat{\Sigma} = (\widehat{D}, \widehat{F})$ and a mapping $\mathcal{H} : D \to \mathcal{P}(\widehat{D})$. If all functions $f : D \to D$ and its abstract version $\widehat{f} : \widehat{D} \to \widehat{D}$ satisfy that for every $d \in D. \mathcal{H}(f(d)) \in \widehat{f}(\{\mathcal{H}(d)\})$ then every term $t[x]$ and its abstract version $\widehat{t}[\widehat{x}]$ satisfies the "safety condition", i.e. $H(t[x]) \in \widehat{t}[\{H(x)\}]$*

A similar lemma holds for Galois Connections. Now, we present the proof which is done by structural induction over data terms. The application of induction gives three cases:

Proof

1. If t is a variable x then we have to prove: $H(x) \in \widehat{x}$, which by definition is equal to $\{H(x)\}$ which trivially satisfy the definition.
2. if t is a vector $[t_0, ..., t_n]$ then we have to prove: $H([t_0, ..., t_n]) \in \widehat{[t_0, ..., t_n]}$

  - By definition $\widehat{[t_0, ..., t_n]} = [\widehat{t_0}, ..., \widehat{t_n}]$
  - By Induction Hypothesis we have that for every t_i. $H(t_i) \in \widehat{t_i}$ therefore: $H([t_0, ..., t_n]) = [H(t_0), ..., H(t_n)] \in [\widehat{t_0}, ..., \widehat{t_n}]$ which trivially proofs the case.
3. if t is a function $f(t')$ then we have to prove: $H(f(t')) \in \widehat{f(t')}$
  - By definition $\widehat{f(t')} = \widehat{f}(\widehat{t'})$
  - By Induction Hypothesis we have that $H(t') \in \widehat{t'}$.
  - By Lemma's conditions for all $H(x) \in \widehat{t'}.H(x) \in \widehat{f}(\{H(x)\})$ which trivially proofs the case.

$\Box$ (Lemma)

5 PVS

Even if the parts of framework exposed in this paper belongs to the classical theories of abstract interpretation, the interest of using the computer assisted theorem prover is big. First, the set of basic definitions and proofs are implemented they can be reused, easily, to extend the framework. In our case we use the framework to extend the classical theories in order to deal with explicit abstraction of action labels. In the same way we also prove correct the symbolic abstraction for a specific modeling language, μCRL. Another code transformations may also be proved following the same methodology with a limited effort. The use of a theorem prover gives always an extra confidence of the correctness of the theory. In this section we give an overview of the main definitions described in PVS. Basic references to PVS can be found in the web page: `http://pvs.csl.sri.com/`

In the present section we present the PVS framework corresponding to the abstraction theory using the homomorphic approach which is easier to understand. In the appendix we include the Galois Connection theories, but remember that the former can be embed in the later us we have done in the previous sections.

Labelled Transition Systems can be viewed in PVS as a record composed by the initial state and a predicate (`step`) that is satisfied if there exists a transition between two given states labelled by a given action name, we define the structure in a parameterized theory:

```
lts[D: TYPE+, Act: TYPE+]: THEORY
BEGIN
   LTS: TYPE = [# init: D, step: [D, Act, D -> bool] #]
END lts
```

As example, we can now define a PVS inductive predicate to determine whether a state is reachable or not (the variable `l` is a LTS defined as above, `d` and `d2` are states and `a` an action label).

```
reachable(l)(d): INDUCTIVE bool =
   d = l`init OR
   (EXISTS d2, a: reachable(l)(d2) AND l`step(d2, a, d))
```

In PVS, *Modal Transition Systems* are defined by means of a type with a predicate modeling the condition which determines that every *must*-transition is a *may*-transition:

```
MODAL_LTS: TYPE =
   {M : [# init: D,
      may_step: [D, A, D -> bool],
      must_step: [D, A, D -> bool]
   #] | subset?(M‘must_step, M‘may_step) }
```

As in the labelled transition system, we can define two reachability predicates as follows:

```
possible(ml)(d):  INDUCTIVE bool =
   ml‘init = d OR
   (EXISTS d2, a: possible(ml)(d2) AND ml‘may_step(d2, a, d))

necessary(ml)(d):  INDUCTIVE bool =
   ml‘init = d OR
   (EXISTS d2, a: necessary(ml)(d2) AND ml‘must_step(d2, a, d))
```

Note that in the definition *ml* denotes an MLTS. The following function returns the *concrete*-MLTS from a given LTS:

```
lts2mlts(lts:LTS): MODAL_LTS =
   (# init := lts‘init,
      may_step:= {(d1, a, d2) | lts‘step(d1, a ,d2)},
      must_step:= {(d1, a, d2) | lts‘step(d1, a, d2)}
   #)
```

The following function computes the minimal may/must$_H$-abstraction of an MLTS as defined in 3:

```
mlts_min_Habs(ml)(hs, ha): MODAL_LTS[aD, aA] =
   (# init:= hs(ml‘init),
      may_step:={(ad1, aA, ad2) | EXISTS cd1, cd2, cA:
                hs(cd1) = ad1 AND
                hs(cd2) = ad2 AND
                ha(cA) = aA AND
                ml‘may_step(cd1, cA, cd2)},
      must_step:={(ad1, aA, ad2) | (FORALL cd1:
                 hs(cd1) = ad1 IMPLIES
                 EXISTS cd2, cA:
                 hs(cd2) = ad2 AND
                 ha(cA) = aA AND
                 ml‘must_step(cd1, cA, cd2)))}
   #)
```

Now we present the abstraction order between MLTSs and its dual (the refinement). The definition implies $mlc \sqsubseteq mla$, being *mlc* and *mla* two MLTSs over the same set of states and action labels. Note that as far as we are considering only mapping functions there is not precision order among states and action labels. Therefore, the definition is simpler than the one presented in definition 5:

```
abstraction?(mla)(mlc): bool =
   subset?(mla`must_step, mlc`must_step) AND
   subset?(mlc`may_step, mla`may_step)

refinement?(mla)(mlc): bool =
   abstraction?(mlc)(mla)
```

Below, we present the syntax and semantics of the logic. The fixpoint operators are not included.

```
state_form[A: TYPE+]: DATATYPE
BEGIN
  mTT: TT?
  mFF: FF?
  mNot(f1:state_form): mNot?
  mAnd(f1: state_form, f2: state_form): mAND?
  mOr(f1: state_form, f2: state_form): mOR?
  mForall(a: A, f: state_form): mForall?
  mExists(a: A, f: state_form): mExists?
END state_formAbs

eval(ml, f): RECURSIVE [setof[D], setof[D]] =
   CASES f OF
      mFF: (emptyset[D], fullset[D]),
      mTT: (fullset[D], emptyset[D]),
      mNot(f1):
       (difference(fullset[D], eval(ml, f1)`2),
        difference(fullset[D], eval(ml, f1)`1)),
      mAnd(f1, f2):
       (intersection(eval(ml, f1)`1, eval(ml, f2)`1),
        intersection(eval(ml, f2)`2, eval(ml, f2)`2)),
      mOr(f1, f2):
       (union(eval(ml, f1)`1, eval(ml, f2)`1),
        union(eval(ml, f2)`2, eval(ml, f2)`2)),
      mForall(a, f1):
      ({d1 | FORALL d2: ml`may_step(d1, a, d2) IMPLIES
        member(d2, eval(ml, f1)`1)},
       {d1 | FORALL d2: ml`must_step(d1, a, d2) IMPLIES
        member(d2, eval(ml, f1)`2)}),
      mExists(a, f1):
      ({d1 | EXISTS d2: ml`must_step(d1, a, d2) AND
        member(d2, eval(ml, f1)`1)},
       {d1 | EXISTS d2: ml`may_step(d1, a, d2) AND
        member(d2, eval(ml, f1)`2)})
   ENDCASES
   MEASURE f BY <<

nec_satisfy(ml, d, f): bool = member(d, eval(ml, f)`1)
pos_satisfy(ml, d, f): bool = member(d, eval(ml, f)`2)
```

The next semantics corresponds to part of the interpretation of a abstract formula over a concrete system, the rest of the cases are trivial:

```
eval(ml, f)(ha): RECURSIVE [setof[cD], setof[cD]] =
   CASES f OF
      ...
      mForall(aA, f1):
       ({d1 | FORALL d2: FORALL cA: ha(cA) = aA AND
        ml`may_step(d1, cA, d2) IMPLIES
        member(d2, eval(ml, f1)(ha)`1)},
       {d1 | FORALL d2: FORALL cA: ha(cA) = aA AND
        ml`must_step(d1, cA, d2) IMPLIES
        member(d2, eval(ml, f1)(ha)`2)}),
      mExists(aA, f1):
       ({d1 | EXISTS cA: ha(cA) = aA AND
         EXISTS d2: ml`must_step(d1, cA, d2) AND
         member(d2, eval(ml, f1)(ha)`1)},
        {d1 | EXISTS cA: ha(cA) = aA AND
         EXISTS d2: ml`may_step(d1, cA, d2) AND
         member(d2, eval(ml, f1)(ha)`2)})
   ENDCASES
   MEASURE f BY <<

nec_satisfy(ml, d, f)(ha): bool = member(d, eval(ml, f)(ha)`1)
pos_satisfy(ml, d, f)(ha): bool = member(d, eval(ml, f)(ha)`2)
```

Now, we present the principal results of section 2. The first Theorem stands for the preservation of formulas by the minimal abstraction and the following lemma characterizes the approximation relation between MLTSs:

```
af: VAR state_formAbs[aA]
preserving_min: THEOREM
   FORALL cml, min_aml, hs, ha:
   min_aml = mlts_min_Habs(cml)(hs,ha) IMPLIES
    FORALL af, ad:(
      (nec_satisfy(min_aml, ad, af)
       IMPLIES
       (FORALL cd: hs(cd) = ad IMPLIES
        nec_satisfy(cml, cd, af)(ha)))
   AND
      (not (pos_satisfy(min_aml, ad, af))
       IMPLIES
       (FORALL cd: hs(cd) = ad IMPLIES
        not(pos_satisfy(cml, cd, af)(ha))))
   )

preserving_approx: LEMMA
   FORALL aml, cml,:
   abstraction?(aml)(cml) IMPLIES
    FORALL af, ad:(
      (nec_satisfy(aml, ad, af)
       IMPLIES
        nec_satisfy(cml, ad, af))
    AND
      (not (pos_satisfy(aml, ad, af))
       IMPLIES
        not (pos_satisfy(cml, cd, af)))
   )
```

So far, we have presented the PVS theory corresponding to the transition systems. Now, we start with the processes specification part. A *linear process operator* is defined as pair composed by the initial state and set of n summands. The theory that specifies LPOs.

```
lpo[Act, State, Local: TYPE+, n: nat]: THEORY
BEGIN
   sumrecord: TYPE = [# act: Act, guard: bool, next: State #]
   summand: TYPE = [State, Local -> sumrecord]
   lpo: TYPE = [# init: State, sums: [below(n) -> summand] #]

   L: VAR lpo
   i: VAR below(n)
   d: VAR State
   e: VAR Local

   act(L)(i)(d, e): Act = L`sums(i)(d, e)`act
   guard(L)(i)(d, e): bool = L`sums(i)(d, e)`guard
   next(L)(i)(d, e): State = L`sums(i)(d, e)`next
END lpo
```

The semantics of an LPO are as follows:

```
   step(s)(d1, a, d2): bool =
    EXISTS e: s(d1, e)`guard AND a = s(d1, e)`act
              AND d2 = s(d1, e)`next
   lpo2lts(l): LTS =
    (#
     init := init(l),
     step := LAMBDA d1, a, d2: EXISTS i:
                    step(l`sums(i))(d1, a, d2)
    #)
```

The following fragment of PVS specifies the *Modal Linear Process Equation*. Condition, states and action labels are lifted to sets:

```
modal_lpo[Act, State, Local: TYPE+, n: nat]: THEORY
BEGIN
  sumrecord: TYPE = [# act: setof[Act],
                    guard: setof[bool],
                    next: setof[State] #]
  summand: TYPE = [setof[State], Local -> sumrecord]
  modal_lpo: TYPE = [# init: setof[State],
                    sums: [below(n) -> summand] #]

  L: VAR modal_lpo
  i: VAR below(n)
  d: VAR setof[State]
  e: VAR Local

  act(L)(i)(d, e): setof[Act] = L`sums(i)(d, e)`act
  guard(L)(i)(d, e): setof[bool] = L`sums(i)(d, e)`guard
  next(L)(i)(d, e): setof[State] = L`sums(i)(d, e)`next

END modal_lpo
```

The semantics of an MLPO are described by the following functions:

```
  may_step(s)(d1, a, d2): bool =
      EXISTS e, A, D:
        member(TRUE, s(d1, e)`guard) AND
            singleton?(A)  AND subset?(A, s(d1, e)`act) AND
            singleton?(D) AND subset?(D, s(d1, e)`next)

  must_step(s)(d1, a, d2): bool =
      EXISTS e:
        (NOT member(FALSE, s(d1, e)`guard)) AND
            singleton?(s(d1, e)`act) AND
            singleton?(s(d1, e)`next)

   must_step?(l)(d1, a, d2): bool =
      EXISTS i: must_step(l`sums(i))(d1, a, d2)
   may_step?(l)(d1, a, d2): bool =
      EXISTS i: may_step(l`sums(i))(d1, a, d2)

   must_steps(l)(leS, leA): setof[[D, A, D]] =
   {x:[D, A, D] | must_step?(l)(x)}

   may_steps(l)(leS, leA): setof[[D, A, D]] =
   {x:[D, A, D] | may_step?(l)(x)}

  mlpo2mlts(l): MODAL_LTS =
      (# init := init(l),
         may_step
           := LAMBDA d1, a, d2: EXISTS i:
              may_step(l`sums(i))(d1, a, d2),
         must_step
           := LAMBDA d1, a, d2: EXISTS i:
              must_step(l`sums(i))(d1, a, d2) #)
```

The safety conditions for the homomorphic case may be specified as follows. *setNext* denotes all the possible continuations from the concrete states *cd1* that map to a given abstract state and *setAct* all the possible labels that can have the transitions from these states. In the predicate *safe?*, *Hs* and *Ha* stand for the pointwise set extension of *hs* and *ha*.

```
setNext(lpo)(ad1)(hs): setof[cD] =
   {cd2| EXISTS cd1, e, (i: below(n)):
         hs(cd1) = ad1 AND  lpo`sums(i)(cd1, e)`next = cd2}

setAct(lpo)(ad1)(hS): setof[cA] =
   {cA| EXISTS cd1, e, (i: below(n)):
         hs(cd1) = ad1 AND  lpo`sums(i)(cd1, e)`act = cA}

safe?(amlpo, lpo)(hs, Hs, Ha): bool =
   (FORALL ad1, ae, (i: below(n)):
      subset?(Hs(setNext(lpo)(ad1)(hs)),
         amlpo`sums(i)(ad1, ae)`next)
      AND
      subset?(Ha(setAct(lpo)(ad1)(hs)),
         amlpo`sums(i)(ad1, ae)`act)
   )
   AND
   (
   FORALL ad, ae, (i: below(n)), cd ,e:
      hs(cd) = ad IMPLIES
      member(guard(lpo)(i)(cd, e), guard(amlpo)(i)(ad, ae))
   )
```

To finish, we present the main result of the symbolic transformation of a *Linear Process Equation*. The following Theorem asserts that if an abstract *mlpo* is a *safe* abstraction of a concrete *lpo*, i.e., they satisfy the safety conditions then the *mlts* generated is an abstraction of the minimal:

```
mlpo_abstraction: THEOREM
   FORALL lpo, amlpo, hs, ha, min_amlts:
      min_amlts = mlts_min_Habs(lpo2mlpo(lpo))(hs, ha) AND
      safe?(amlpo, lpo2mlpo(lpo))(hs, lift(hs), lift(ha))
   IMPLIES
      abstraction?(mlpo2mlts(amlpo))(min_amlts)
```

The fragment of PVS presented in this section correspond to the most important parts of the specification of the homomorphic abstraction. In the appendix the complete theory of the Galois Connection framework is included.

6 Conclusion

In order to apply the abstract interpretation framework for reactive systems [8, 17] to μCRL processes, we extended it with the explicit abstraction of action labels. This required non-trivial changes in most definitions. A μCRL specification in LPE-form can be abstracted to a modal LPE; the state space of this MLPE corresponds to a reduced MLTS, approximating the original LTS. This approximation can be used to verify and refute safety as well as liveness formulas for the original system.

The resulting approach incorporates the homomorphism approach (which is easier to understand and use) and the Galois Connection approach (which preserves more properties, especially liveness properties). A user-defined homomorphism can also be lifted to a Galois Connection automatically, combining ease with precision.

We already have a working prototype implementation, which will be described in a separate paper. It is based on a projection of MLPEs to LPEs, in order to reuse existing state space generation tools. We will apply the techniques from [17] in order to translate the three-valued model checking problem to regular model checking, in order to also reuse a model checker for the modal μ-calculus, e.g. CADP [12, 23]. Another interesting question, optimality of abstractions, can in principle be addressed along the lines of [8].

Our theory allows the collection of standard data abstractions with accompanying safety criteria proofs. Such abstraction libraries can be freely used in various protocols, without additional proof obligations. This will enhance automatic verification of protocols in specialized domains.

Model checking becomes more and more crucial for the correctness of software, but in practice additional techniques, such as abstraction, are needed. This may affect the correctness and modularity of the resulting verification methodology and tools. We support modularity by implementing abstraction as an LPE to LPE transformation, which can be composed freely by other existing transformations [2]. We feel that it is important to provide a rigorous basis for verification technology, so we have checked the main part of the results in this paper in the theorem prover PVS (except Lemma 12, which would require a deep embedding of μCRL syntax in PVS).

References

[1] J.A. Bergstra, , and J.W. Klop. Algebra of communicating processes with abstraction. *TCS*, pages 77–121, 1985.

[2] S. Blom, W. Fokkink, J. F. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *Proc. of CAV*, LNCS, pages 250–254, 2001.

[3] S. Blom, J. F. G., I. van Langevelde, B. L., and J.C. van de Pol. New developments around the μCRL tool set. In *ENTCS*, 2003.

[4] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM*, pages 343–354, 1992.

[5] P. Cousot. Abstract interpretation based formal methods and future challenges. In *Informatics: 10 Years Back, 10 Years Ahead*, LNCS, pages 138 – 143, 2001.

[6] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *Proc. of the 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[7] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of the 6th ACM Symposium on Principles of Programming Languages*, pages 269–282, 1979.

[8] D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking.* PhD thesis, Eindhoven University of Technology, 1996.

[9] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, pages 253–291, 1997.

[10] D.E. Long. *Model Checking, Abstraction, and Compositional Verification.* PhD thesis, Carnegie Mellon University, 1993.

[11] A. Fantechi, S. Gnesi, and D. Latella. Towards automatic temporal logic verification of value passing process algebra using abstract interpretation. In *Concur*, LNCS, pages 562–578, 1996.

[12] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP – a protocol validation and verification toolbox. In *Proc. of CAV*, LNCS, pages 437–440, 1996.

[13] W. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science. Springer, 2000.

[14] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. *LNCS*, pages 426–440, 2001.

[15] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62, 1995.

[16] J.F. Groote and J.J. van Wamel. Algebraic data types and induction in μCRL. Technical report, Universiteit van Amsterdam, 1994.

[17] M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: a foundation for three-valued program analysis. *LNCS*, pages 155–169, 2001.

[18] N. D. Jones and F. Nielson. Abstract interpretation: A semantics-based tool for program analysis. In *Handbook of Logic in Computer Science*, pages 527–636. 1995.

[19] D. Kozen. Results on the propositional μ-calculus. In Mogens Nielsen and Erik Meineche Schmidt, editors, *ICALP*, LNCS, pages 348–359, 1982.

[20] K. G. Larsen. Modal specifications. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems: Proceedings*, LNCS, pages 232–246, 1989.

[21] K. G. Larsen and B. Thomsen. A modal process logic. In *Proc., 3rd Annual Symposium on Logic in Computer Science*, pages 203–210. IEEE Computer Society, 1988.

[22] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, pages 11–44, 1995.

[23] R. Mateescu. *Verification des proprietes temporelles des programmes paralleles.* PhD thesis, Institut National Polytechnique de Grenoble, 1998.

[24] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In *Proc. of CAV*, LNCS, pages 411–414, 1996.

[25] D. Schmidt. Binary relations for abstraction and refinement, 1999.

[26] C. Stirling. Modal and temporal logics. In S. Abramsky, Dov M. Gabbay, and T. S.Ẽ. Maibaum, editors, *Handbook of Logic in Computer Science. Volume 2. Background: Computational Structures*, pages 477–563. 1992.

A PVS theory for Galois Connection

```
galois_connection[D, A: TYPE+]: THEORY

% Definition of Galois Connection for the setting we are going to use.
% Some auxiliary predicates and two basic results are included.

BEGIN
   le: VAR pred[[A, A]]

   alpha: VAR [setof[D] -> A]
   gamma: VAR [A -> setof[D]]

   a, a1, a2: VAR A
   D, D1, D2: VAR setof[D]

   total?(alpha): bool =
      FORALL D: EXISTS a: alpha(D) = a

   total?(gamma): bool =
      FORALL a: EXISTS D: gamma(a) = D

   monotonic?(alpha)(le): bool =
      FORALL D1, D2:
      subset?(D1, D2) IMPlIES le(alpha(D1), alpha(D2))

   monotonic?(gamma)(le): bool =
      FORALL a1, a2:
         le(a1,a2) IMPlIES subset?(gamma(a1), gamma(a2))

   galois?(alpha, gamma)(le): bool =
      partial_order?(le)AND
      total?(alpha) AND monotonic?(alpha)(le) AND
      total?(gamma) AND monotonic?(gamma)(le) AND
      (FORALL D:subset?(D, gamma(alpha(D))))
      AND
      (FORALL a: le(alpha(gamma(a)), a))

   lemmaG1: LEMMA
      FORALL alpha, gamma, le: galois?(alpha, gamma)(le)
         IMPLIES FORALL D: alpha(gamma(alpha(D))) = alpha(D)

   lemmaG2: LEMMA
      FORALL alpha, gamma, le: galois?(alpha, gamma)(le)
         IMPLIES FORALL a: gamma(alpha(gamma(a))) = gamma(a)

END galois_connection


lts[D: TYPE+, A: TYPE+]: THEORY

% Definition of Labelled Transition System

BEGIN
   LTS: TYPE = [# init: D, step: [D, A, D -> bool] #]
END lts


modal_lts[D: TYPE+, A: TYPE+]: THEORY

% Definition of Modal Labelled Transition System and some related definitions
% Only the definition is used in the rest of the theories

BEGIN

   MODAL_LTS: TYPE =
   {M : [# init: D,
         may_step: [D, A, D -> bool],
         must_step: [D, A, D -> bool]
   #] | subset?(M`must_step, M`may_step) }

   ml: VAR MODAL_LTS
   d, d1, d2: VAR D
   a: VAR A

   possible(ml)(d):  INDUCTIVE bool =
   ml`init = d OR
   (EXISTS d2, a: possible(ml)(d2) AND ml`may_step(d2, a, d))

   necessary(ml)(d):  INDUCTIVE bool =
   ml`init = d OR
      (EXISTS d2, a: necessary(ml)(d2) AND ml`must_step(d2, a, d))

  Inclusion: LEMMA
     FORALL ml: necessary(ml)(d) IMPLIES possible(ml)(d)
END modal_lts


lts2mlts[D, A: TYPE+]: THEORY

% Basic transformation from a LTS to a concrete MLTS

BEGIN
   IMPORTING lts[D, A]
   IMPORTING modal_lts[D, A]

   l: VAR LTS
   ml: VAR MODAL_LTS
   d, d1, d2: VAR D
   a: VAR A

   lts2mlts(lts:LTS): MODAL_LTS =
   (# init := lts`init,
       may_step:= {(d1, a, d2) | lts`step(d1, a ,d2)},
       must_step:= {(d1, a, d2) | lts`step(d1, a, d2)} #)
END lts2mlts


state_formAbs[A: TYPE+]: DATATYPE

% The defintion of the state formulas. They are used over MLTSs

BEGIN
   mTT: TT?
   mFF: FF?
   mNot(f1:state_formAbs): mNot?
   mAnd(f1: state_formAbs, f2: state_formAbs): mAND?
   mOr(f1: state_formAbs, f2: state_formAbs): mOR?
   mForall(alpha: A, f: state_formAbs): mForall?
   mExists(alpha: A, f: state_formAbs): mExists?
END state_formAbs


mlts_state_formSemAbs[D, A: TYPE+]: THEORY

% The semantics of the state formulas over an MLTS
% There are two predicates of satisfaction (nec and pos)
% for the neccesarily satisfied and for the possibly

BEGIN
  IMPORTING state_formAbs[A]
  IMPORTING modal_lts[D, A]


   d, d1, d2: VAR D
   ml, ml1, ml2: VAR MODAL_LTS
   f, f1, f2: VAR state_formAbs
   alpha,a: VAR A
   le: VAR pred[[A, A]]

   eval(ml, f)(le): RECURSIVE [setof[D], setof[D]] =
    CASES f
      OF mFF: (emptyset[D], fullset[D]),
         mTT: (fullset[D], emptyset[D]),
         mNot(f1):
              (difference(fullset[D], eval(ml, f1)(le)`2),
               difference(fullset[D], eval(ml, f1)(le)`1)),
         mAnd(f1, f2):
              (intersection(eval(ml, f1)(le)`1, eval(ml, f2)(le)`1),
               intersection(eval(ml, f2)(le)`2, eval(ml, f2)(le)`2)),
         mOr(f1, f2):
              (union(eval(ml, f1)(le)`1, eval(ml, f2)(le)`1),
               union(eval(ml, f2)(le)`2, eval(ml, f2)(le)`2)),
         mForall(alpha, f1):
              ({d1 | FORALL d2: FORALL a:  le(alpha, a) AND
               ml`may_step(d1, a, d2) IMPLIES
               member(d2, eval(ml, f1)(le)`1)},
              {d1 | FORALL d2: FORALL a: le(a, alpha) AND
               ml`must_step(d1, a, d2) IMPLIES
               member(d2, eval(ml, f1)(le)`2)}),
         mExists(alpha, f1):
              ({d1 |EXISTS a: le(a, alpha) AND
               EXISTS d2: ml`must_step(d1, a, d2) AND
               member(d2, eval(ml, f1)(le)`1)},
              {d1 | EXISTS a: le(alpha, a) AND
               EXISTS d2: ml`may_step(d1, a, d2) AND
               member(d2, eval(ml, f1)(le)`2)})
         ENDCASES
     MEASURE f BY <<

   nec_satisfy(ml, d, f)(le): bool = member(d, eval(ml, f)(le)`1)
   pos_satisfy(ml, d, f)(le): bool = member(d, eval(ml, f)(le)`2)
END mlts_state_formSemAbs

mlts_state_formSemConcretization[cD, cA, aA: TYPE+]: THEORY

% Semantics of an abstract formula interpretated over a concrete system

BEGIN
  IMPORTING state_formAbs[aA]
  IMPORTING modal_lts[cD, cA]

   d, d1, d2: VAR cD
   ml, ml1, ml2: VAR MODAL_LTS
   f, f1, f2: VAR state_formAbs
   cA: VAR cA
   aA: VAR aA

   gamma: VAR [aA -> setof[cA]]
   alpha: VAR [setof[cA] -> aA]

   le: VAR pred[[aA, aA]]

   eval(ml, f)(alpha, gamma, le): RECURSIVE [setof[cD], setof[cD]] =
    CASES f
      OF mFF: (emptyset[cD], fullset[cD]),
         mTT: (fullset[cD], emptyset[cD]),
         mNot(f1):
              (difference(fullset[cD], eval(ml, f1)(alpha, gamma, le)`2),
               difference(fullset[cD], eval(ml, f1)(alpha, gamma, le)`1)),
         mAnd(f1, f2):
              (intersection(eval(ml, f1)(alpha, gamma, le)`1,
               eval(ml, f2)(alpha, gamma, le)`1),
               intersection(eval(ml, f2)(alpha, gamma, le)`2,
               eval(ml, f2)(alpha, gamma, le)`2)),
         mOr(f1, f2):
              (union(eval(ml, f1)(alpha, gamma, le)`1,
               eval(ml, f2)(alpha, gamma, le)`1),
               union(eval(ml, f2)(alpha, gamma, le)`2,
               eval(ml, f2)(alpha, gamma, le)`2)),
         mForall(aA, f1):
              ({d1 | FORALL cA: le(aA, alpha(singleton(cA)))
               IMPLIES FORALL d2: ml`may_step(d1, cA, d2) IMPLIES
               member(d2, eval(ml, f1)(alpha, gamma, le)`1)},
              {d1 | FORALL cA: le(alpha(singleton(cA)), aA)
               IMPLIES FORALL d2: ml`must_step(d1, cA, d2) IMPLIES
               member(d2, eval(ml, f1)(alpha, gamma, le)`2)}),
         mExists(aA, f1):
              ({d1 | EXISTS cA:  le(alpha(singleton(cA)), aA)  AND
               EXISTS d2: ml`must_step(d1, cA, d2) AND
               member(d2, eval(ml, f1)(alpha, gamma, le)`1)},
              {d1 | EXISTS cA: le(aA, alpha(singleton(cA)))  AND
               EXISTS d2: ml`may_step(d1, cA, d2) AND
               member(d2, eval(ml, f1)(alpha, gamma, le)`2)})
       ENDCASES
     MEASURE f BY <<

   nec_satisfy(ml, d, f)(alpha, gamma, le): bool =
      member(d, eval(ml, f)(alpha, gamma, le)`1)
   pos_satisfy(ml, d, f)(alpha, gamma, le): bool =
      member(d, eval(ml, f)(alpha, gamma, le)`2)
END mlts_state_formSemConcretization


mlts_min_Gabstraction[cD, cA, aD, aA:TYPE+]: THEORY

% Definitions of:
% * Minimal abstraction
% * Restriction. To facilitate the proofs we define the minimal restricted
%   abstraction in a simpler way:
% -- For the may part we make use of the observation that the minimal
%    set is composed by pairs of singleton elements.
% -- For the must part we over approximate the minimal set
%    presented in the previous sections.
%    This over appoximation is enough to prove the correctness of the resutls.
% * The abstraction relation between MLTSs
% And two Basic Preservation Results:
% -- preserving_min
% -- preserving_apprx
% The theory also includes some auxiliary functions and lemmas

BEGIN
   IMPORTING modal_lts[cD, cA]
   IMPORTING modal_lts[aD, aA]
   IMPORTING mlts_state_formSemConcretization[cD, cA, aA]
   IMPORTING mlts_state_formSemAbs[aD, aA]
   IMPORTING galois_connection[cD, aD]
   IMPORTING galois_connection[cA, aA]

   cml: VAR MODAL_LTS[cD, cA]
   aml, aml_res, amltsA, amltsC: VAR MODAL_LTS[aD, aA]

   cd, cd1, cd2, cd22: VAR cD
   ad, ad1, ad2: VAR aD
   ad11, ad22: VAR aD
   cA, cA1: VAR cA
   a, a1, a11, aA, aA1: VAR aA

   gammaS: VAR [aD -> setof[cD]]
   gammaA: VAR [aA -> setof[cA]]
   alphaS: VAR [setof[cD] -> aD]
   alphaA: VAR [setof[cA] -> aA]
```

```
mlts_min_Gabs?(aml, cml)(alphaS, alphaA): bool =
   aml`init  = alphaS(singleton(cml`init))
   AND
   (
   FORALL ad1, aA, ad2: aml`must_step(ad1, aA, ad2) IFF
       (FORALL cd1: alphaS(singleton(cd1)) = ad1 IMPLIES
       (EXISTS cd2, cA: (alphaS(singleton(cd2)) = ad2 AND
       alphaA(singleton(cA)) = aA AND
       cml`must_step(cd1, cA, cd2)))))
   AND
   (
   FORALL ad1, aA, ad2: aml`may_step(ad1, aA, ad2) IFF
       (EXISTS cd1, cd2, cA: (alphaS(singleton(cd1)) = ad1 AND
       alphaS(singleton(cd2)) = ad2 AND alphaA(singleton(cA)) = aA AND
       cml`may_step(cd1, cA, cd2)))
   )

leS: VAR pred[[aD, aD]]
leA: VAR pred[[aA, aA]]

A, B: VAR setof[[setof[cD], setof[cA]]]
C : VAR setof[[aD, aA]]

PP: VAR setof[[cD, cA]]
Pmin: VAR [setof[cD], setof[cA]]

dR, dRmin: VAR setof[cD]
aR, aRmin: VAR setof[cA]

setMinMay(cml, ad1)(gammaS, gammaA): setof[[setof[cD], setof[cA]]] =
   {(dR, aR) | EXISTS cd1, cA, cd2: gammaS(ad1)(cd1) AND
    cml`may_step(cd1, cA, cd2) AND
    dR = singleton(cd2) AND aR = singleton(cA)}

setMust(cml, ad1)(gammaS, gammaA): setof[[setof[cD], setof[cA]]]=
   {(dR, aR) | FORALL cd1: gammaS(ad1)(cd1) IMPLIES EXISTS cA, cd2:
    cml`must_step(cd1, cA, cd2) AND dR(cd2) AND aR(cA)}

minSetMust(cml, ad1)(gammaS, gammaA)(A): setof[[cD, cA]] =
   {(cd2, cA) | EXISTS dRmin, aRmin: aRmin(cA) AND
    dRmin(cd2) AND A(dRmin, aRmin) AND
    EXISTS cd1: gammaS(ad1)(cd1) AND cml`must_step(cd1, cA, cd2)}

minMust(PP): [setof[cD], setof[cA]] =
   ({cd2 | EXISTS cA: PP(cd2, cA)},
    {cA | EXISTS cd2:  PP(cd2, cA)})

setAlpha(B)(alphaS, alphaA): setof[[aD, aA]] =
   {(ad22, aA1) | EXISTS Pmin:
    B(Pmin) AND ad22 = alphaS(Pmin`1) AND aA1 = alphaA(Pmin`2)}


restriction?(aml_res, cml)(alphaS, gammaS, alphaA, gammaA): bool =
   (FORALL ad1, aA, ad2: aml_res`may_step(ad1, aA, ad2)
    IFF
    setAlpha(setMinMay(cml, ad1)(gammaS, gammaA))
            (alphaS, alphaA)((ad2, aA))
   )
   AND
   (FORALL ad1, aA, ad2: aml_res`must_step(ad1, aA, ad2)
   IFF
    nonempty?(setMust(cml, ad1)(gammaS, gammaA)) AND
    aA = alphaA(minMust(minSetMust(cml, ad1)(gammaS, gammaA)
              (setMust(cml, ad1)(gammaS, gammaA)))`2)
    AND
    ad2 = alphaS(minMust(minSetMust(cml, ad1)(gammaS, gammaA)
              (setMust(cml, ad1)(gammaS, gammaA)))`1)
```

```
      )

   abstraction?(amltsA, amltsC)(leS, leA): bool =
      (FORALL ad1, a1, ad2, ad11:
       amltsC`may_step(ad1, a1, ad2) AND  leS(ad1, ad11)
       IMPLIES EXISTS a11, ad22: leA(a1, a11) AND leS(ad2,ad22) AND
          amltsA`may_step(ad11, a11, ad22))
       AND
      (FORALL ad11, a11, ad22, ad1:
       amltsA`must_step(ad11, a11, ad22) AND  leS(ad1, ad11)
       IMPLIES EXISTS a1, ad2: leA(a1, a11) AND leS(ad2, ad22) AND
          amltsC`must_step(ad1, a1, ad2))

   lemma_singleton: LEMMA
    FORALL cml, aml, aml_res, gammaA, gammaS, alphaA, gammaS, leA, leS:
    restriction?(aml_res, cml)(alphaS, gammaS, alphaA, gammaA) AND
    galois?(alphaS, gammaS)(leS) AND galois?(alphaA, gammaA)(leA)
    IMPLIES(FORALL cd1, cA, cd2, ad1: gammaS(ad1)(cd1) AND
       cml`may_step(cd1, cA, cd2)
       IMPLIES
       aml_res`may_step(ad1, alphaA(singleton(cA)), alphaS(singleton(cd2))))

   af: VAR state_formAbs[aA]

   preserving_approx: LEMMA
    FORALL amltsA, amltsC, leA, leS:
    abstraction?(amltsA, amltsC)(leS, leA) AND
    galois?(alphaS, gammaS)(leS) AND galois?(alphaA, gammaA)(leA)
    IMPLIES (FORALL af, ad1, ad11: (
       nec_satisfy(amltsA, ad11, af)(leA) AND leS(ad1, ad11)
       IMPLIES
       nec_satisfy(amltsC, ad1, af)(leA))
      AND (
       not(pos_satisfy(amltsA, ad11, af)(leA)) AND leS(ad1, ad11)
       IMPLIES
       not(pos_satisfy(amltsC, ad1, af)(leA)))
    )

   preserving_min: LEMMA
    FORALL cml, aml, aml_res, gammaA, gammaS, alphaA, gammaS, leA, leS:
    restriction?(aml_res, cml)(alphaS, gammaS, alphaA, gammaA) AND
    galois?(alphaS, gammaS)(leS) AND galois?(alphaA, gammaA)(leA)
    IMPLIES FORALL ad, cd, af:(
       (gammaS(ad)(cd) AND
        nec_satisfy(aml_res, ad, af)(leA)
        IMPLIES
        nec_satisfy(cml, cd, af)(alphaA, gammaA, leA)
     )
     AND
       (gammaS(ad)(cd) AND
        not(pos_satisfy(aml_res, ad, af)(leA))
        IMPLIES
        not(pos_satisfy(cml, cd, af)(alphaA, gammaA, leA)))
     )
END mlts_min_Gabstraction


lpo[Act, State, Local: TYPE+, n: nat]: THEORY

% Defintion of Linear Process Operator (or Equation)

BEGIN
   sumrecord: TYPE = [# act: Act, guard: bool, next: State #]
   summand: TYPE = [State, Local -> sumrecord]
   lpo: TYPE = [# init: State, sums: [below(n) -> summand] #]

   L: VAR lpo
   i: VAR below(n)
   d: VAR State
   e: VAR Local

   act(L)(i)(d, e): Act = L`sums(i)(d, e)`act
   guard(L)(i)(d, e): bool = L`sums(i)(d, e)`guard
   next(L)(i)(d, e): State = L`sums(i)(d, e)`next
END lpo


lpo_semantics[Act, D, E: TYPE+, n: nat]: THEORY

% Semantics of an LPO

BEGIN
   IMPORTING lpo[Act, D, E, n], lts[D, Act]

   l: VAR lpo
   s: VAR summand
   d1, d2: VAR D
   e: VAR E
   a: VAR Act
   i: VAR below(n)

   step(s)(d1, a, d2): bool =
      EXISTS e: s(d1, e)`guard AND a = s(d1, e)`act AND d2 = s(d1, e)`next

   lpo2lts(l): LTS =
      (# init := init(l),
         step := LAMBDA d1, a, d2: EXISTS i: step(l`sums(i))(d1, a, d2) #)

END lpo_semantics

modal_lpo[Act, State, Local: TYPE+, n: nat]: THEORY

% Definition of a Modal Linear Process Operator
% Note that: States Parametes are not lifted to sets because
% in this file we only use the Galois Connection approach.

BEGIN

   sumrecord: TYPE = [# act: Act, guard: setof[bool], next: State #]
   summand: TYPE = [State, Local -> sumrecord]
   modal_lpo: TYPE = [# init: State,
                        sums: [below(n) -> summand] #]

   L: VAR modal_lpo
   i: VAR below(n)
   d: VAR State
   e: VAR Local

   act(L)(i)(d, e): Act = L`sums(i)(d, e)`act
   guard(L)(i)(d, e): setof[bool] = L`sums(i)(d, e)`guard
   next(L)(i)(d, e): State = L`sums(i)(d, e)`next

END modal_lpo


mlpo_semantics[A, D, E: TYPE+, n: nat]: THEORY

% Semantics of an MLPO

BEGIN
   IMPORTING modal_lpo[A, D, E, n], modal_lts[D, A]

   l: VAR modal_lpo
   s: VAR summand
   d1, d2, d: VAR D
   e: VAR E
   a, a1: VAR A
   i: VAR below(n)

   leS: VAR pred[[D, D]]
   leA: VAR pred[[A, A]]

   may_step(s)(d1, a, d2): bool =
      EXISTS e: member(TRUE, s(d1, e)`guard) AND
      a = s(d1, e)`act AND
      d2 = s(d1, e)`next

   must_step(s)(d1, a, d2): bool =
      EXISTS e: (NOT member(FALSE, s(d1, e)`guard)) AND
      (s(d1, e)`act = a) AND
      (s(d1, e)`next = d2)

   must_step?(l)(d1, a, d2): bool =
      EXISTS i: must_step(l`sums(i))(d1, a, d2)

   may_step?(l)(d1, a, d2): bool =
      EXISTS i: may_step(l`sums(i))(d1, a, d2)

   must_steps(l)(leS, leA): setof[[D, A, D]] =
      {x:[D, A, D] | must_step?(l)(x)}

   may_steps(l)(leS, leA): setof[[D, A, D]] =
      {x:[D, A, D] | may_step?(l)(x)}

  mlpo2mlts(l)(leS, leA): MODAL_LTS =
      (# init := init(l),
         may_step := LAMBDA d1, a, d2: EXISTS i:
            may_step(l`sums(i))(d1, a, d2),
         must_step := LAMBDA d1, a, d2: EXISTS i:
            must_step(l`sums(i))(d1, a, d2) #)

END mlpo_semantics

lpo2mlpo[Act, State, Local: TYPE+, n: nat]: THEORY

% Basic Transformation from a LPO to an concrete MLPO

BEGIN
   B: TYPE = {b: setof[bool] | nonempty?(b)}
   IMPORTING lpo[Act, State, Local, n],
                  modal_lpo[Act, State, Local, n]
   IMPORTING lpo_semantics[Act, State, Local, n],
                  mlpo_semantics[Act, State, Local, n],
                  lts2mlts[State, Act]

   lpo: VAR lpo
   mlpo: VAR modal_lpo

  s: VAR State
   l: VAR Local

   lpo2mlpo(lpo): modal_lpo =
   (# init:= lpo`init,
       sums:= LAMBDA(i: below(n)): LAMBDA(s, l):
       (#
        act:= act(lpo)(i)(s, l),
        guard:={b:bool| b=guard(lpo)(i)(s, l)},
        next:=next(lpo)(i)(s, l)
       #)
   #)

END lpo2mlpo

summand_abstraction[cA, aA, cD, aD, cE, aE: TYPE+, n:nat]:THEORY
```

```
% Definition of the Safety Conditions and Main Theorem "mlpo_abstraction"

BEGIN
   IMPORTING lpo[cA, cD, cE, n],
             modal_lpo[cA, cD, cE, n],
             modal_lpo[aA, aD, aE, n],
             lpo2mlpo[cA, cD, cE, n]
   IMPORTING galois_connection[cD, aD],
             galois_connection[cA, aA]
   IMPORTING mlpo_semantics[cA, cD, cE, n],
             mlpo_semantics[aA, aD, aE, n]
   IMPORTING mlts_min_Gabstraction[cD, cA, aD, aA]

   lpo: VAR lpo[cA, cD, cE, n].lpo
   cmlpo: VAR modal_lpo[cA, cD, cE, n].modal_lpo
   amlpo: VAR modal_lpo[aA, aD, aE, n].modal_lpo

   amlts, min_amlts, res_amlts: VAR  MODAL_LTS[aD, aA]

   cd: VAR cD
   ad: VAR aD
   e: VAR cE
   ae: VAR aE

   gammaD: VAR [aD -> setof[cD]]
   gammaA: VAR [aA -> setof[cA]]
   alphaD: VAR [setof[cD] -> aD]
   alphaA: VAR [setof[cA] -> aA]
   leD: VAR pred[[aD, aD]]
   leA: VAR pred[[aA, aA]]

   ad1, ad11: VAR aD
   ae1: aE
   ce1: cE
   cd1, cd2: VAR cD
   cA: VAR cA

   setNext(lpo)(ad1)(gammaD): setof[cD] =
      {cd2| EXISTS cd1, e, (i: below(n)):
       gammaD(ad1)(cd1) AND  lpo`sums(i)(cd1, e)`next = cd2}

   setAct(lpo)(ad1)(gammaD): setof[cA] =
      {cA| EXISTS cd1, e, (i: below(n)):
       gammaD(ad1)(cd1) AND  lpo`sums(i)(cd1, e)`act = cA}

   safe?(amlpo, lpo)(alphaD, gammaD, alphaA, gammaA, leD, leA): bool =
      (FORALL ad1, ae,(i: below(n)):
       leD(alphaD(setNext(lpo)(ad1)(gammaD)), amlpo`sums(i)(ad1, ae)`next)
       AND
       leA(alphaA(setAct(lpo)(ad1)(gammaD)), amlpo`sums(i)(ad1, ae)`act)
      )
      AND
      (FORALL ad, ae, (i: below(n)), cd ,e:
       gammaD(ad)(cd) IMPLIES
       member(guard(lpo)(i)(cd, e), guard(amlpo)(i)(ad, ae))
      )

    mlpo_abstraction: THEOREM
       FORALL lpo, amlts, amlpo, gammaD, gammaA, alphaD, alphaA, leA, leD:(
       galois?(alphaD, gammaD)(leD) AND galois?(alphaA, gammaA)(leA) AND
       restriction?(amlts, mlpo2mlts(lpo2mlpo(lpo))(=, =))
                   (alphaD, gammaD, alphaA, gammaA) AND
       safe?(amlpo, lpo)(alphaD, gammaD, alphaA, gammaA, leD, leA)
       )
       IMPLIES
       abstraction?(mlpo2mlts(amlpo)(leD, leA),  amlts)(leD, leA)
END summand_abstraction
```