Decomposition of Automata PDL and its Extension

Xinxin Liu and Bingtian Xue

***Abstract*—In this work we study the decomposition problem of automata PDL and one of its extension. We proved that automata PDL enjoys a good decomposition property under a very large class of process context, while this problem is more complicated for regular PDL, which has an exponential blow-up. We introduce proposition identifiers to automata PDL to obtain a language which has a good balance between expressiveness and ease of analysis. We prove that this extended specification language still has a good decomposition property for a large class of process contexts. After that we present a method to solve the weak bisimulation equations as an application of the extended language, by combining the decomposition property and the decision procedure proposed in [1].**

***Index Terms*—decomposition, propositional dynamic logic, fixed point, weak bisimulation, equation solving.**

I. Introduction

When practicing hierarchical and modular development methodologies within a specification formalism, you will inevitably face decomposition problems. Informally and in short, a decomposition problem is concerned with reducing a specification required of a combined system into (sufficient and necessary) specifications of the system's components.

Facing concurrent systems, the notion of *weakest inner property* is a generalization of that of weakest precondition and strongest postcondition for sequential systems [2]. For a given specification S in some specification formalism and a (one hole) process context C, the weakest inner property of S under C, written as $wip(S, C)$, is the weakest property for a process p such that the combined process $C(p)$ satisfies S. When developing a concurrent system with the specification S and after having decided to use the context C as part of the implementation, $wip(S, C)$ is obviously the weakest property for the rest of the system in order that the whole system satisfies S. Whether $wip(S, C)$ is always expressible as a specification in the same formalism as S certainly depends on the expressive power of the specification formalism, but an increase of expressive power does not necessarily imply that it is more likely to be able to express the weakest inner property: when a more expressive specification formalism is used, on one hand it becomes easier to express properties like $wip(S, C)$ if S is a specification from a less expressive specification formalism, on the other hand however, properties like $wip(S, C)$ are more likely to exceed the expressive power because S now ranges over more complicated specifications. So whether $wip(S, C)$ is always expressible as a specification in the same formalism as S should be examined individually.

One well known specification formalism is Propositional Dynamic Logic (PDL) [3]. PDL was introduced by Fisher and Ladner [4] in the late 1970s as a formalism for reasoning about programs. Soon afterwards the logic was outdated for that purpose through the introduction of the modal μ-calculus [5]. However, PDL has by now become a standard logic that is far from being outdated. PDL formulas are easy to understand and analyze, while μ-calculus formulas are often hard. In particular, some simple properties concerning repetition of traces often have to be encoded into complex recursive properties in μ-calculus. PDL can be used in program verification, to describe the dynamic evolution of agent-based systems, for planning or knowledge engineering, it has links to epistemic logics, it is closely related to description logics, etc. There has been a resurgence of interest in PDL in recent years. In [6], [7] and [8], many interesting problems of PDL with some extension have been studied.

(Regular) PDL uses regular expressions for programs in which limited recursive patterns of traces can be described. In this work we will study automata PDL[3], which uses automata to describe programs and is equivalent to (regular) PDL. The use of automata instead of regular expressions has similar effect as Pratt's *flowgraph* in his Propositional Flowgraph Logic (PFL) [9] and they both bring more succinctness into the expressions. We will show that automata PDL enjoys such a good decomposition property for a very large class of process contexts, while this problem is more complicated for (regular) PDL, which has an exponential blow-up.

There are two ways to enhance PDL: adding new operators on the formula level or on the program level. We introduce limited use of recursive propositions into automata PDL to obtain a language, which has a good balance between expressiveness and ease of analysis [1]. We demonstrate that this language still has a good decomposition property for a large class of process contexts. In [1] we present a decision procedure for the satisfiability of the formulas. The worst case time complexity of the decision procedure is polynomial in the size of the programs and exponential in the number of the sub-formulas. Combining the decision procedure and the decomposition property, we can use this language to solve weak (or branching) bisimulation equations of processes.

In the following section we define the syntax and semantics of APDL. In section 3 we study the decomposition problems of APDL. In section 4 we define the syntax and semantics of the extended language and present the decomposition property of this extended language. In section 5 we give an example of solving the bisimulation equations. In the last section we conclude our work, together with some future and related work. Missing proofs can be found in the appendix.

Manuscript received December 13, 2011; revised January 12, 2012. This work was supported by the Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences.

X. Liu is with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, P.O.Box 8718, 100190 Beijing, China, e-mail: xinxin@ios.ac.cn.

B. Xue is with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, and Graduate School of Chinese Academy of Sciences, P.O.Box 8718, 100190 Beijing, China, e-mail: xuebt@ios.ac.cn.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

II. Automata Propositional Dynamic Logic

This section presents the syntax and semantics of APDL, which is equivalent to regular PDL, because of the *Kleene Theorem* [10].

The language of PDL has expressions of two sorts: *propositions* or *formulas* $\varphi, \psi, \ldots$ and *programs* $\alpha, \beta, \ldots$ There are countably many atomic symbols of each sort. Atomic programs are denoted $a, b, \ldots$, which are also called *actions*, and the set of all atomic programs is denoted Act here. Atomic propositions here are tt, which makes the presentation here slightly different from but equivalent to that in [3]. APDL is proved to be equivalent to regular PDL, and it can bring more succinctness in expressivity, and facilitate development of decomposition. The set of all propositions is denoted Φ and the set of all programs is denoted Π. We define Φ and Π inductively as follows:

1) $\mathrm{tt} \in \Phi$;
2) if φ, $\psi \in \Phi$ then $\varphi \vee \psi \in \Phi$;
3) if $\varphi \in \Phi$, $\alpha \in \Pi$ then $\langle\alpha\rangle\varphi \in \Phi$;
4) if $\varphi \in \Phi$, $\neg\varphi \in \Phi$.

1) $Act \subseteq \Pi$;
2) $M \in \Pi$, where $M = (N, S, i, j, \delta)$ is a finite automaton: $N = 0, \ldots, n-1$: set of states; $S = Act \cup \{?\varphi \mid \varphi \in \Phi\}$: alphabet; $i, j \in N$: start and final states respectively; δ: $N \times S \times N$: transition relation.

The semantics of APDL is interpreted on a labeled transition system $\langle \mathbf{S}, Act, \{\xrightarrow{a} \mid a \in Act\}\rangle$, where $\mathbf{S}$ is a set of states, and each $\xrightarrow{a}$ is a transition relation $\xrightarrow{a} \subseteq \mathbf{S} \times \mathbf{S}$. The satisfaction relation $\models \subseteq \mathbf{S} \times \Phi$ and the transition relation $\Rightarrow \subseteq (\mathbf{S} \times \Pi) \times \mathbf{S}$ are defined inductively as follows:

1) $p \models \mathrm{tt}$ holds for all $p \in \mathbf{S}$;
2) $p \models \varphi \vee \psi$ in case that $p \models \varphi$ or $p \models \psi$;
3) $p \models \langle\alpha\rangle\varphi$ in case that there exists $q \in \mathbf{S}$ such that $(p, \alpha) \Rightarrow q$ and $q \models \varphi$;
4) $p \models \neg\varphi$ in case that $p \not\models \varphi$.

$$\frac{}{(p, a) \Rightarrow q} \quad p \xrightarrow{a} q \qquad \frac{}{(p, (N, S, i, i, \delta)) \Rightarrow p}$$

$$\frac{(p', (N, S, i', j, \delta)) \Rightarrow q}{(p, (N, S, i, j, \delta)) \Rightarrow q} \quad p \xrightarrow{a} p', (i, a, i') \in \delta$$

$$\frac{(p, (N, S, i', j, \delta)) \Rightarrow q}{(p, (N, S, i, j, \delta)) \Rightarrow q} \quad p \models \varphi, (i, ?\varphi, i') \in \delta$$

We write $p \not\models \varphi$ when $p \models \varphi$ does not hold.

III. Decomposition of APDL

Decomposition is a desirable property as a specification language, which relates a specification language to states (systems) with a structure. In this section we will show that APDL enjoys such a good decomposition property for a very large class of process context. For a given APDL formula φ with declaration D and a (one hole) context C, we can use an APDL formula ψ to express the specification which requires that a state p putting together with C satisfies $C(p) \models \varphi$.

In order to facilitate a general investigation of how contexts transform properties, we use the operational theory of contexts in terms of action transducers, which is introduced in [11] and applied in [12], [13]. Informally and in short, a context is semantically viewed as an object which consumes actions from its inner processes and produces actions for an external observer, thus acting as an interface between them. That is, in the behavior of the process $C(p)$ (as a state in LTS), the context C acts as an interface between an external observer and the internal process p, in the sense that C consumes actions produced by the internal process p in order to produce actions for the external observer. We allow transduction in which the context produces actions on its own without involving the inner process. Also, the context may change during transduction.

Definition 3.1: A context system *is a structure* $(K, Act, \longrightarrow)$*, where* K *is a set of contexts,* $\longrightarrow \subseteq K \times (Act_0 \times Act) \times K$ *is the transduction relation,* $Act_0 = Act \cup \{0\}$ *with 0 being a distinguished no-action symbol (i.e.* $0 \notin Act$*).*

For $(C, (a, b), C') \in \longrightarrow$ we shall adopt the notation $C \xrightarrow[a]{b} C'$ and interpret this as: "by consuming the action a the context C can produce the action b and change into C'". For $a = 0$ the production of b does not involve consumption of any action.

Semantically, the relationship between the semantics of process, context, and combined process satisfies the following: $C(p) \xrightarrow{b} q$ if and only if there exist C', p' such that $q = C'(p'), C \xrightarrow[a]{b} C', p \xrightarrow{a} p'$ or there exists C' such that $q = C'(p), C \xrightarrow[0]{b} C'$.

Now we will show for contexts which satisfy the above operational semantics, specifications in APDL are decomposable. What we will do is to introduce a (weakest) *property transformer*, $\mathcal{W}$, which - given the context C and the property φ - will construct the weakest inner property $\mathcal{W}(C, \varphi)$.

Definition 3.2: Let $(K, Act, \longrightarrow)$ *be a context system with* K *being finite. For APDL formula* φ *and program* α*, and contexts* $C, C' \in K$*, define APDL formula* $\mathcal{W}(C, \varphi)$ *inductively:*

$$\begin{aligned}
\mathcal{W}(C, \mathrm{tt}) &= \mathrm{tt}\\
\mathcal{W}(C, \varphi \vee \psi) &= \mathcal{W}(C, \varphi) \vee \mathcal{W}(C, \psi)\\
\mathcal{W}(C, \neg\varphi) &= \neg\mathcal{W}(C, \varphi)\\
\mathcal{W}(C, \langle a\rangle\varphi) &= \bigvee_{C \xrightarrow[b]{a} C'} \langle b\rangle\mathcal{W}(C', \varphi) \vee \bigvee_{C \xrightarrow{a} C'} \mathcal{W}(C', \varphi)\\
\mathcal{W}(C, \langle M\rangle\varphi) &= \langle M'\rangle\mathrm{tt}
\end{aligned}$$

$M' = (N', S', (i, C), n_f, \delta')$, *where* $M = (N, S, i, j, \delta)$

$N' = N \times K \cup \{n_f\}$

$S' = Act \cup \{?\mathcal{W}(C, \psi) \mid C \in K,\ ?\psi \in S\}$
$\quad \cup \{?\mathcal{W}(C, \varphi) \mid C \in K\} \cup \{\mathrm{tt}\}$

$\delta' = \{((k, C_1), ?\mathcal{W}(C_1, \psi), (l, C_1)) \mid (k, ?\psi, l) \in \delta,\ C_1 \in K\}$
$\quad \cup \{((k, C_1), b, (l, C_2)) \mid (k, a, l) \in \delta,\ C_1 \xrightarrow[b]{a} C_2\}$
$\quad \cup \{((k, C_1), ?\mathrm{tt}, (l, C_2)) \mid (k, a, l) \in \delta,\ C_1 \xrightarrow{a} C_2\}$
$\quad \cup \{((j, C_1), ?\mathcal{W}(C_1, \varphi), n_f) \mid C_1 \in K\}$

The reason of introducing the final state n_f is to make M' a deterministic automata. Of course we can built an APDL formula equivalent to $\langle M'\rangle\mathrm{tt}$ with a nondeterministic automata M'', which is a little easier to understand:

$$\mathcal{W}(C, \langle M\rangle\varphi) = \langle M''\rangle\mathcal{W}(C, \varphi)$$

$M'' = (N'', S'', (i, C), S''_f, \delta'')$, where $M = (N, S, i, j, \delta)$

$N'' = N \times K$

$S'' = Act \cup \{?\mathcal{W}(C, \psi) \mid C \in K,\ ?\psi \in S\} \cup \{\mathrm{tt}\}$

$S''_f = \{(j, C_1) \mid C_1 \in K\}$

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$$\delta'' = \{((k, C_1), ?\mathcal{W}(C_1, \psi), (l, C_1)) \mid (k, ?\psi, l) \in \delta,\ C_1 \in K\}$$
$$\cup \{((k, C_1), b, (l, C_2)) \mid (k, a, l) \in \delta,\ C_1 \xrightarrow[b]{a} C_2\}$$
$$\cup \{((k, C_1), ?\mathrm{tt}, (l, C_2)) \mid (k, a, l) \in \delta,\ C_1 \xrightarrow{a} C_2\}$$

It is easy to prove that $\forall p \in \mathbf{S}$, $p \models \langle M' \rangle \mathrm{tt}$ if and only if $p \models \langle M'' \rangle \mathcal{W}(C, \varphi)$

The following theorem shows that this transformer has the expected properties.

Theorem 3.3: Let $(K, Act, \longrightarrow)$ be a context system, φ be an APDL formula, and $C, C' \in K$ be contexts. Then for any $p \in \mathbf{S}$ the following equivalence holds:

$$C(p) \models \varphi \text{ if and only if } p \models \mathcal{W}(C, \varphi).$$

IV. APDL WITH RECURSION AND ITS DECOMPOSITION

We now study an extension of APDL - APDL with recursive propositions. First we present the syntax and semantics of this language, which arrows recursion on the properties. We refer to this language as recAPDL in this paper.

We allow *property identifiers* in recAPDL, which are denoted $X, Y, \ldots$, and the set of all property identifiers is denoted $\mathcal{V}$. The meaning of the property identifiers is determined by a recursive declaration D. The syntax of recAPDL programs is the same as APDL, and that of formulas only needs to add the following:

5. $X \in \Phi$;

The recursive declaration D is defined as follows:

Definition 4.1: A declaration *is a finite set D with elements of the form $X = \varphi$. Different elements in D have different left hand sides.*

The requirement that different elements of D have different left hand sides is to exclude multiple definition of identifiers.

A test $?\varphi$ is called a *positive* test if φ is a positive formula, and a *negative* test if negative. For an automaton M, if all tests in the alphabet S are positive (negative) tests, M is called a positive (negative) program. For formulas, whether it is positive or negative is defined as:

1) tt, X is positive;
2) φ and ψ are positive (negative), then $\varphi \vee \psi$ is positive (negative);
3) φ is positive (negative), α is positive (negative), then $\langle \alpha \rangle \varphi$ is positive;
4) φ is positive (negative), then $\neg\varphi$ is negative (positive).

$D = \{X_1 = \varphi_1, \ldots, X_m = \varphi_m\}$ is *well defined* if $\varphi_1, \ldots, \varphi_m$ are positive formulas w.r.t $X_1, \ldots, X_m$.

The semantics of formulas and programs under a given environment ρ are the same as APDL under a given environment ρ except for the property identifiers:

5. $p \models_\rho X$ in case that $p \in \rho(X)$.

$D = \{X_1 = \varphi_1, \ldots, X_m = \varphi_m\}$ defines an environment for the identifiers $X_1, \ldots, X_m$, which are the weakest properties they satisfy.

Proposition 4.2: Let ρ, ρ' be two environments such that $\rho(X) \subseteq \rho'(X)$, φ a positive formula, α a positive program, β a negative program w.r.t all the X defined in ρ and ρ', then:

1) *if $(p, \alpha) \Rightarrow_\rho q$ then $(p, \alpha) \Rightarrow_{\rho'} q$;*
2) *if $(p, \beta) \Rightarrow_{\rho'} q$ then $(p, \beta) \Rightarrow_\rho q$;*
3) *if $p \models_\rho \varphi$ then $p \models_{\rho'} \varphi$.*

The proof of this proposition only needs simple induction on the transition rule $\Rightarrow$ and the structure of φ. We will not do that because of the limitation of the length. Just be careful with the $\neg\psi$ case.

Since D is well defined and we have the monotonous property, there exists a unique maximal environment $\rho_{\max} = \bigcup\{\rho \mid \rho \text{ satisfies } D\}$ and $\rho_{\max}$ satisfies D.

Then we have the semantics of formulas under a declaration D:

$$p \models_D \varphi \text{ iff } p \models_{\rho_{\max}} \varphi$$
$$(p, \alpha) \Rightarrow_D q \text{ iff } (p, \alpha) \Rightarrow_{\rho_{\max}} q$$

Example 4.3:

1) fairness property:
   CTL*: $EGFp$
   μ-calculus: $\nu X.\mu Y.\langle\bullet\rangle((X \wedge p) \vee Y)$
   recAPDL: $X = \langle \bullet^*.?p \rangle X$
   $\cdot \longrightarrow \cdots\cdots \longrightarrow \cdot \longrightarrow \cdots\cdots \longrightarrow \cdot \longrightarrow \cdots\cdots\cdots$
   $X \qquad\qquad p, X \qquad\qquad p, X$
2) $X = \langle (\tau.?X)^*.a \rangle Y$
   $Y = \langle (\tau.?Y)^*.b \rangle X$
   $\cdot \xrightarrow{\tau} \cdots \xrightarrow{\tau} \cdot \xrightarrow{a} \cdot \xrightarrow{\tau} \cdots \xrightarrow{\tau} \cdot \xrightarrow{b} \cdot \xrightarrow{\tau} \cdots$
   $X \quad \cdots \quad X \quad Y \quad \cdots \quad Y \quad X \quad \cdots$
   $\xrightarrow{\tau} \cdot \xrightarrow{a} \cdot \xrightarrow{\tau} \cdots \xrightarrow{\tau} \cdot \xrightarrow{b} \cdot \xrightarrow{\tau} \cdots\cdots\cdots$
   $X \quad Y \quad \cdots \quad Y \quad X \quad \cdots\cdots$

In the remaining of this section we will show that recAPDL still has the decomposition property for a large class of process contexts. That is, for contexts which satisfy the operational semantics in *Section 3*, specifications in rAPDL are decomposable. We will introduce a (weakest) *property transformer*, $\mathcal{W}^r$, which is the same as $\mathcal{W}$ for APDL except for the property identifiers:

$$\mathcal{W}^r(C, X) \quad = \quad X^C$$

where X^C is an introduced proposition identifier.

Definition 4.4: Let $(K, Act, \longrightarrow)$ be a context system, φ be a formula with declaration D, and $C, C' \in K$ be contexts. We define a declaration D' as follows:

if $X = \varphi_X \in D$, $C \in K$ then $X^C = \mathcal{W}^r(C, \varphi_X) \in D'$.

The following lemma and theorem show that this transformer has the expected properties.

Lemma 4.5: Let $(K, Act, \longrightarrow)$ be a context system, φ be a recAPDL formula and $C, C' \in K$ be contexts. Let ρ, ρ' be two environment which satisfy the following: $C(p) \in \rho(X)$ if and only if $p \in \rho'(X^C)$. Then for any $p \in \mathbf{S}$ the following equivalences hold:

$$C(p) \models_\rho \varphi \text{ if and only if } p \models_{\rho'} \mathcal{W}^r(C, \varphi).$$

Theorem 4.6: Let $(K, Act, \longrightarrow)$ be a context system, φ be a recAPDL formula with declaration D, and $C, C' \in K$ be contexts. Let D' be constructed as Definition 4. *Then for any $p \in \mathbf{S}$ the following equivalence holds:*

$$C(p) \models_D \varphi \text{ if and only if } p \models_{D'} \mathcal{W}^r(C, \varphi).$$

V. SOLVING WEAK BISIMULATION EQUATIONS

Now as an application of recAPDL, we give a way to solve process equations of the form $C(x) \equiv p$ (finding out whether there is a process x which satisfies the equation), where $\equiv$ can any equivalence relation among strong bisimulation equivalence, weak bisimulation equivalence, branching bisimulation equivalence, $\frac{2}{3}$-bisimulation equivalence, n-nested bisimulation equivalence for any n.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

By combining the decomposition property of recAPDL and the decision procedure presented in [1], we show how to solve weak bisimulation equations $C(x) \approx p$. The following is how to do it.

Step 1 According to the result of [1], there is a recAPDL formula φ_p and a declaration D such that $C(x) \approx p$ if and only if $C(x) \models_D \varphi_p$.
We associate a proposition identifier X_p for each state p in the (finite) state space and for each relevant action $a \in Act$ (assuming only finitely many of them are relevant to the process) we also associate an automaton M_a with the definition:
- when $a = \tau$, $M_a = (\{i_a, i_\tau\}, Act, i_a, i_\tau, \delta_a)$ where $\delta_a = \{(i_a, a, i_\tau), (i_a, \tau, i_a), (i_\tau, \tau, i_\tau)\}$;
- when $a = \tau$, $M_\tau = (\{i_\tau\}, Act, i_\tau, i_\tau, \{(i_\tau, \tau, i_\tau)\}$

After that, we construct a declaration

$$D = \{X_p = \bigwedge_{p \xrightarrow{a} p'} \langle M_a \rangle X_{p'} \wedge \bigwedge_{a \in Act} [a](\bigvee_{p \xRightarrow{\hat{a}} p'} X_{p'}) | p \in \mathbf{S}\}$$

It has been proved in [1] that:
Lemma 5.1: Let $p \in \mathbf{S}$ be a finite state process, X_p and D constructed as above. Then for any process $q \in \mathbf{S}$, $p \approx q$ if and only if $q \models_D X_p$.

Step 2 By the decomposition property of recAPDL, there is a recAPDL formula $\mathcal{W}^r(C, \varphi_p)$ such that
$C(x) \models_D \varphi_p$ if and only if $x \models_D \mathcal{W}^r(C, \varphi_p)$
Thus there is an x with $C(x) \approx p$ if and only if $\mathcal{W}^r(C, \varphi_p)$ is satisfiable under D.

Step 3 And then we can use the decision procedure presented in [1] to check whether $\mathcal{W}^r(C, \varphi_p)$ is satisfiable under D.
Informally the iterative procedure is as follows:
Starting from the set $\mathcal{C}$ of sets of the sub-formulas of given formula φ, here is $\mathcal{W}^r(C, \varphi_p)$, do the following until $\mathcal{C}$ does not decrease: find out one set $\Gamma \in \mathcal{C}$ in which the formulas have contradiction - between the formulas in it or between it and other $\Gamma \in \mathcal{C}$. The former part tries to find out Γ which avoids the axioms of propositions, and the latter checks whether all the transitions required are allowed in $\mathcal{C}$.
Then we have:
Lemma 5.2: φ is satisfiable if and only if, upon termination there exists $\Gamma \in \mathcal{C}$ such that $\varphi \in \Gamma$.
The correctness of this algorithm has been proved in [1], which guarantees that upon termination, all formulas in $\mathcal{C}$ are satisfiable and all satisfiable Γ will not be deleted from $\mathcal{C}$ during the iteration. The worst case time complexity of the decision procedure is polynomial in the size of the programs and exponential in the number of the sub-formulas. □

In the same way, we can solve the branching bisimulation equations of processes and any bisimulation equations mentioned above. In [13], [14], [15], *Disjunctive Modal Transition Systems* (DMTS) were used to solve strong bisimulation equations of the form $C(x) \sim p$. Here by using recAPDL we can do what we could not do with DMTS.

VI. Conclusion

In this work we study the decomposition problem, which is a desirable property as a specification language, of automata PDL and one of its extension – recAPDL. Decomposition is always an interesting issue which relates a specification language to states (systems) with a structure. We prove that APDL enjoys a good decomposition property under a very large class of process context, while this problem is more complicated for regular PDL, which has an exponential blow-up. We introduce proposition identifiers to APDL to obtain a language - recAPDL, which has a good balance between expressiveness and ease of analysis. We prove that this extended specification language still has a good decomposition property for a large class of process contexts. After that we present a way to solve the weak (branching) bisimulation equations as an application of recAPDL, by combining the decomposition property and the decision procedure proposed in [1], which has a time complexity which is polynomial in the size of the programs in the formulas and exponential in the number of the sub-formulas.

By adding nesting to recAPDL [16], we can get a language which is more expressive than CTL and CTL*, and is still no more expressive than modal μ-calculus. We demonstrate that this extended recAPDL with nesting is also decomposable. This is beyond this paper, and we will present that in [16], which is in preparing.

According to the results of this paper, tools for decomposing recAPDL specifications can be built. Moreover, by combining the tools for deciding satisfiability of rPDL specifications and the tools for decomposing rPDL specifications, equatioin solver (EQ) can be built. EQ can either find a solution or provide a proof to explain why the equation system is not solvable. This work is in progressing.

Another interesting future direction is the decomposition problem for the contexts which have two or more holes (i.e. there are two or more components to be designed). In this case for each hole (missing component) we need to find a sub-specification, such that the sub-specification is sufficiently strong that the correctness of the overall system is guaranteed. On the other hand the sub-specifications should be as weak as possible in order not to restrict unnecessarily the implementations allowed for the components. The difficulty here is that in general this decomposition will not be unique: the requirements to one component may be loosened if in return the requirements to the others are strengthened. This problem for recADPL is being studied.

Appendix A
Proof of Theorem 3.3

Proof Induction on the structure of φ. The cases are quite simple except for $\langle M \rangle \varphi$.
For $\langle M \rangle \varphi$ we need to prove that:
$C(p) \models \langle M \rangle \varphi$ iff $p \models \mathcal{W}(C, \langle M \rangle \varphi)$
"⇒": $C(p) \models \langle M \rangle \varphi$ then $p \models \mathcal{W}(C, \langle M \rangle \varphi)$
$C(p) \models \langle M \rangle \varphi \Longrightarrow$
$\exists C', p'$ s.t. $(C(p), M) \Rightarrow C'(p')$ and $C'(p') \models \varphi$
If we can prove that:
$(C(p), M) \Rightarrow C'(p')$ & $C'(p') \models \varphi$ then $\exists q$ s.t. $(p, M') \Rightarrow q$, it is obvious that $p \models \langle M' \rangle \mathrm{tt}$
Then $p \models \mathcal{W}(C, \langle M \rangle \varphi)$
Now we prove that:
$(C(p), (N, S, i, j, \delta)) \Rightarrow C'(p')$ & $C'(p') \models \varphi$ then $\exists q$ s.t. $(p, (N', S', (i, C), n_f, \delta')) \Rightarrow q$
Induction on the transition rule of $\Rightarrow$:

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

1) $\dfrac{}{(p,(N,S,i,i,\delta)) \Rightarrow p}$

$(C(p),(N,S,i,j,\delta)) \Rightarrow C'(p')\ \&\ C'(p') \models \varphi$
$\Longrightarrow\quad i=j, C'=C, p'=p\ \&\ C'(p') \models \varphi$
$\Longrightarrow\quad i=j\ \&\ C(p) \models \varphi$
$\Longrightarrow\quad i=j\ \&\ p \models \mathcal{W}(C,\varphi)$
inductive hypothesis
And $\quad ((j,C), ?\mathcal{W}(C,\varphi), n_f) \in \delta'$
$\&\ (p,(N',S',n_f,n_f,\delta')) \Longrightarrow p$
definition of M'
$\Longrightarrow\quad (p,(N',S',(i,C),n_f,\delta')) \Longrightarrow p$

2) $\dfrac{(p',(N,S,i',j,\delta)) \Rightarrow q}{(p,(N,S,i,j,\delta)) \Rightarrow q} \quad p \xrightarrow{a} p', (i,a,i') \in \delta$

$(C(p),(N,S,i,j,\delta)) \Rightarrow C'(p')\ \&\ C'(p') \models \varphi$
$\Longrightarrow\quad C(p) \xrightarrow{a} C''(p''), (i,a,i') \in \delta\ \&$
$(C''(p''),(N,S,i',j,\delta)) \Rightarrow C'(p')\ \&$
$C'(p') \models \varphi$
$\Longrightarrow\quad C(p) \xrightarrow{a} C''(p''), (i,a,i') \in \delta\ \&$
$\exists q, (p'',(N',S',(i',C''),n_f,\delta')) \Rightarrow q$
inductive hypothesis
$\Longrightarrow\quad C \xrightarrow[b]{a} C', p \xrightarrow{b} p''$ or $C \xrightarrow{a} C', p''=p,$
$\&\ (i,a,i') \in \delta, \exists q,$
$(p'',(N',S',(i',C''),n_f,\delta')) \Rightarrow q$

– $C \xrightarrow[b]{a} C', p \xrightarrow{b} p''$
$C \xrightarrow[b]{a} C'', p \xrightarrow{b} p'', (i,a,i') \in \delta$
$\&\ \exists q, (p'',(N',S',(i',C''),n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad ((i,C),b,(i',C'')) \in \delta', p \xrightarrow{b} p''\ \&$
$\exists q, (p'',(N',S',(i',C''),n_f,\delta')) \Rightarrow q$
definition of δ'
$\Longrightarrow\quad \exists q, (p,(N',S',(i,C),n_f,\delta')) \Rightarrow q$

– $C \xrightarrow{a} C'', p = p''$
$C \xrightarrow{a} C'', p = p'', (i,a,i') \in \delta$
$\&\ \exists q, (p'',(N',S',(i',C''),n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad ((i,C),?\mathrm{tt},(i',C'')) \in \delta', p = p''\ \&$
$\exists q, (p'',(N',S',(i',C''),n_f,\delta')) \Rightarrow q$
definition of δ'
$\Longrightarrow\quad ((i,C),?\mathrm{tt},(i',C'')) \in \delta', p \models \mathrm{tt}\ \&$
$\exists q, (p,(N',S',(i',C''),n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad \exists q, (p,(N',S',(i,C),n_f,\delta') \Rightarrow q$

3) $\dfrac{(p,(N,S,i',j,\delta)) \Rightarrow q}{(p,(N,S,i,j,\delta)) \Rightarrow q} \quad p \models \psi, (i,?\psi,i') \in \delta$

$(C(p),(N,S,i,j,\delta)) \Rightarrow C'(p')\ \&\ C'(p') \models \varphi$
$\Longrightarrow\quad (C(p),(N,S,i',j,\delta)) \Rightarrow C'(p')\ \&$
$C(p) \models \psi, (i,?\psi,i') \in \delta\ \&\ C'(p') \models \varphi$
$\Longrightarrow\quad p \models \mathcal{W}(C,\varphi)\ \&$
outer inductive hypothesis
$((i,C),?\mathcal{W}(C,\psi),(i',C)) \in \delta'\ \&$
definition of δ'
$\exists q$ s.t. $(p',(N',S',(i',C),n_f,\delta')) \Rightarrow q$
inner inductive hypothesis
$\Longrightarrow\quad \exists q$ s.t. $(p,(N',S',(i,C),n_f,\delta')) \Rightarrow q$

"$\Leftarrow$": $p \models \mathcal{W}(C,\langle M\rangle\varphi)$ then $C(p) \models \langle M\rangle\varphi$
$p \models \mathcal{W}(C,\langle M\rangle\varphi) \Longrightarrow p \models \langle M'\rangle\mathrm{tt} \Longrightarrow \exists q$ s.t $(p,M') \Rightarrow q$ and $q \models \mathrm{tt}$
So we need to prove that $(p,(N',S',(i,C),n_f,\delta')) \Rightarrow q$ then $C(p) \models \langle(N,S,i,j,\delta)\rangle\varphi$
Induction on the transition rule of $\Rightarrow$

– $\dfrac{}{p,(N,S,i,i,\delta)) \Rightarrow p}$
$(p,(N',S',(i,C),n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad p=q, (N,S,i,j,\delta) = \emptyset$
$\&\ p \models \langle(N',S',(i,C),n_f,\delta')\rangle\mathrm{tt}$
$\Longrightarrow\quad (C(p),(N,S,i,j,\delta)) \Rightarrow C(p)$
$\&\ p \models \mathcal{W}(C,\langle(N,S,i,j,\delta)\rangle\varphi)$
$\Longrightarrow\quad (C(p),(N,S,i,j,\delta)) \Rightarrow C(p)$
$\&\ p \models \mathcal{W}(C,\varphi) \qquad (N,S,i,j,\delta) = \emptyset$
$\Longrightarrow\quad (C(p),(N,S,i,j,\delta)) \Rightarrow C(p)$
$\&\ C(p) \models \varphi$ outer inductive hypothesis
$\Longrightarrow\quad C(p) \models \langle(N,S,i,j,\delta)\rangle\varphi$

– $\dfrac{(p',(N,S,i',j,\delta)) \Rightarrow q}{(p,(N,S,i,j,\delta)) \Rightarrow q} \quad p \xrightarrow{b} p', (i,b,i') \in \delta$
$(p,(N',S',(i,C),n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad p \xrightarrow{b} p', ((i,C),b,(i',C')) \in \delta'\ \&$
$(p',(N',S',(i',C''),n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad p \xrightarrow{b} p',\ \&\ \exists C'$ s.t. $(i,a,i') \in \delta\ \&$
$C \xrightarrow[b]{a} C'$ definition of δ'
$\&\ C'(p') \models \langle(N,S,i',j,\delta)\rangle\varphi$
inductive hypothesis
$\Longrightarrow\quad C(p) \xrightarrow{a} C'(p'),\ \&\ (i,a,i') \in \delta\ \&\ \exists C'',$
p'' s.t. $(C'(p'),(N,S,i',j,\delta)) \Rightarrow C''(p'')$
$\&\ C''(p'') \models \varphi$
$\Longrightarrow\quad (C(p),(N,S,i,j,\delta)) \Rightarrow C''(p'')\ \&$
$C''(p'') \models \varphi$ transition rule of $\Rightarrow$
$\Longrightarrow\quad C(p) \models \langle(N,S,i,j,\delta)\rangle\varphi$ definition of $\models$

– $\dfrac{(p,(N,S,i',j,\delta)) \Rightarrow q}{(p,(N,S,i,n_f,\delta)) \Rightarrow q} \quad p \models \psi, (i,?\psi,i') \in \delta$
$(p,(N',S',(i,C),n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad p \models \psi, ((i,C),?\psi,n') \in \delta$ and
$(p,(N',S',n',n_f,\delta')) \Rightarrow q$
$\Longrightarrow\quad n' = (i',C'), (i,a,i') \in \delta, C \xrightarrow{a} C',$
$\psi = \mathrm{tt},$ or $n' = n_f, \psi = \mathcal{W}(C,\varphi)$
$\&\ p \models \psi,\ \&\ (p,(N',S',n',n_f,\delta')) \Rightarrow q$

$*\ n' = (i',C'), (i,a,i') \in \delta, C \xrightarrow{a} C', \psi = \mathrm{tt}$
$\Longrightarrow\quad (i,a,i') \in \delta\ \&\ C(p) \xrightarrow{a} C'(p)\ \&$
definition of context system

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$C'(p) \models \langle (N,S,i',j,\delta)\rangle\varphi$
inductive hypothesis
$\Longrightarrow$ $(i,a,i') \in \delta$ & $C(p) \xrightarrow{a} C'(p)$ &
$\exists C'', p'',$ s.t.
$(C'(p),(N,S,i',j,\delta)) \Rightarrow C''(p'')$
& $C''(p'') \models \varphi$ definition of $\models$
$\Longrightarrow$ $(C(p),(N,S,i,j,\delta)) \Rightarrow C''(p'')$
& $C''(p'') \models \varphi$ transition rule of $\Rightarrow$
$\Longrightarrow$ $C(p) \models \langle (N,S,i,j,\delta)\rangle\varphi$

$*$ $n' = n_f, \psi = \mathcal{W}(C,\varphi)$
$\Longrightarrow$ $p \models \mathcal{W}(C,\varphi)$ & $(N,S,i,j,\delta) = \emptyset$
$\Longrightarrow$ $C(p) \models \varphi$ and $(N,S,i,j,\delta) = \emptyset$
outer inductive hypothesis
$\Longrightarrow$ $C(p) \models \langle (N,S,i,j,\delta)\rangle\varphi$ □

APPENDIX B
PROOF OF LEMMA 4.5

Proof The proof of this lemma is the same as the proof for APDL decomposition property (*Theorem 3.3*) under given environments ρ and ρ' except for the proposition identifiers:

- X
  $C(p) \models_\rho X$
  $\Longleftrightarrow$ $C(p) \in \rho(X)$ definition of $\models_\rho$
  $\Longleftrightarrow$ $p \in \rho'(X^C)$ definition of ρ and ρ'
  $\Longleftrightarrow$ $p \models_{\rho'} X^C$ definition of $\models_{\rho'}$
  $\Longleftrightarrow$ $p \models_{\rho'} \mathcal{W}^r(C,X)$ definition of $\mathcal{W}^r$ □

APPENDIX C
PROOF OF THEOREM 4.6

Proof

$\Rightarrow$: $C(p) \models_D \varphi$ then $p \models_{D'} \mathcal{W}^r(C,\varphi)$
Define: $\rho(X^C) = \{p \mid C(p) \models_D X, p \in \mathbf{S}, C \in K\}$
We have $C(p) \models_D \varphi$ then $p \models_\rho \mathcal{W}^r(C,\varphi)$ holds by *Lemma 1*;
We can easily prove that ρ is a post-fix point of D' as follows:

- $X = \varphi_X \in D$, so $X^C = \mathcal{W}^r(C,\varphi_X) \in D'$
  $p \in \rho(X^C) \Longrightarrow C(p) \models_D X \Longrightarrow C(p) \models_D \varphi_X \Longrightarrow p \models_\rho \mathcal{W}^r(C,\varphi_X)$

So $p \models_\rho \varphi$ then $p \models_{D'} \varphi$;
Then $C(p) \models_D \varphi$ then $p \models_{D'} \mathcal{W}^r(C,\varphi)$.

$\Leftarrow$: $p \models_{D'} \mathcal{W}^r(C,\varphi)$ then $C(p) \models_D \varphi$
Define: $\rho(X) = \{C(p) \mid p \models_{D'} X^C, p \in \mathbf{S}, C \in K\}$
Then we have $p \models_{D'} \mathcal{W}^r(C,\varphi)$ then $C(p) \models_\rho \varphi$ holds by *Lemma 1*;
We can easily prove that ρ is a post-fix point of D as follows:

- $X = \varphi_X \in D$, so $X^C = \mathcal{W}^r(C,\varphi_X) \in D$
  $C(p) \in \rho(X) \Longrightarrow p \models_{D'} X^C \Longrightarrow p \models_{D'} \mathcal{W}^r(C,\varphi_X) \Longrightarrow C(p) \models_\rho \varphi_X$

So $p \models_\rho \varphi$ then $p \models_{D'} \varphi$;
Then $p \models_{D'} \mathcal{W}^r(C,\varphi)$ then $C(p) \models_D \varphi$. □

REFERENCES

[1] X. Liu and B. Xue, "Specification in pdl with recursion," submitted to Nasa Formal Methods Symposium, Norfolk, 2012.
[2] E. Dijkstra, *A Discipline of Programming*. Prentice–Hall, 1976.
[3] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*. MIT Press, 2000.
[4] M.J.Fischer and R.E.Ladner, "Propositional dynamic logic of regular programs," *J. Comput. System Sci.*, vol. 18, no. 2, 1979.
[5] D. Kozen, "Results on the propositional mu–calculus," *Lecture Notes In Computer Science, Springer Verlag*, vol. 140, 1982, in Proc. of International Colloquium on Algorithms, Languages and Programming 1982.
[6] M. Lange, "Model checking propositional dynamic logic with all extras," *Journal of applied logic*, vol. 4, 2006.
[7] D. Leivant, "Propositiona dynamic logic for recursive procedures," *Lecture Notes In Computer Science, Springer Verlag*, vol. 5295, 2008.
[8] C. Löding, C. Lutz, and O. Serre, "Propositional dynamic logic with recursive programs," *Journal logic and algebraic programming*, vol. 73, 2007.
[9] V. Pratt, "Using graphs to understand pdl," *Lecture Notes In Computer Science, Springer Verlag*, vol. 131, 1982.
[10] B. Khoussainov and A. Nerode, "Automata theory and its applications." Birkhauser Boston, 2001.
[11] K. Larsen, "Context–dependent bisimulation between processes," Ph.D. dissertation, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, 1986.
[12] K. Larsen and R. Milner, "Verifying a protocol using relativized bisimulation," *Lecture Notes In Computer Science, Springer Verlag*, vol. 267, 1987, in Proceedings of International Colloquium on Algorithms, Languages and Programming 1987.
[13] K. Larsen and X. Liu, "Equation solving using modal transition systems," in *Proceedings on Logic in Computer Science*, 1990.
[14] X. Liu, "Specification and decomposition in concurrency," Ph.D. dissertation, University of Aalborg, Fredrik Bajers Vej 7, DK 9220 Aalborg ø, Denmark, 1992.
[15] L. X. K.G. Larsen, "On equation solving," in *2nd NOrdic Workshop on Program Correctness*, K. Larsen and A. Skou, Eds., 1990.
[16] X. Liu and B. Xue, "Recursive pdl with nesting," in preparing.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)