# MODEL CHECKING OF AMBIENT CALCULUS SPECIFICATIONS AGAINST AMBIENT LOGIC FORMULAS

by

OZAN AKAR

BSc, in Computer Engineering, İstanbul Technical University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2009

MODEL CHECKING OF AMBIENT CALCULUS SPECIFICATIONS AGAINST AMBIENT LOGIC FORMULAS

APPROVED BY:

| | |
|---|---|
| Prof. Mehmet Ufuk Çağlayan<br>(Thesis Supervisor) | ..... |
| Prof. Emin Anarım | ..... |
| Assist. Prof. Fatih Alagöz | ..... |

DATE OF APPROVAL: 30.09.2009

# ACKNOWLEDGEMENTS

First of all, I thank my advisor Prof. Mehmet Ufuk Çağlayan for his support and guidance during my studies. He supported me from the beginning and gave me a big amount of his time for this thesis.

I would like to thank Prof. Emin Anarım and Assist. Prof. Fatih Alagöz for being a part of the committee.

I thank Devrim Ünal for discussions we had and for his feedback during my study. Many critical ideas about my thesis were sprouted during these discussions. His recommendations and criticisms made my arguments much more solid and richer.

I thank Gökhan Kabukcu for his support during my time at Boğaziçi University. He shared his time and knowledge with me and has always been there when I needed.

I thank Mete Geleş, Şerif Bahtiyar and Salih Bayar for thier supports and comments on this thesis. What I captured from those comments were very helpful for me. I have taken great pleasure from their company as well as I have learned a lot.

I thank the members of NETLAB for their discussions and feedbacks during my studies. I enjoyed being a member of their productive and joyful environment. I seized an opportunity to get acquainted with great people there.

I thank Murat Turan, Tolga Çadırcıoğlu, A. Turgay Yiğit and Murat Akzeybek for their great friendship and support while I were studying on this thesis.

Most important of all, I would like to express my deepest gratitude to my family, especially to my mother, for their endless love, patience and encouragement.

# ABSTRACT


## MODEL CHECKING OF AMBIENT CALCULUS SPECIFICATIONS AGAINST AMBIENT LOGIC FORMULAS

Formal methods are mathematical techniques applied in specification and verification of concurrent interactive systems. Model checking is a widely used formal method for formal verification of systems. In model checking, an exhaustive search is applied on a finite state model of the target system.

While there are model checkers to verify the only temporal behaviors of systems, two new notions of model checking analysis recently come into prominence, mobility and locations. Although there are various model checker proposals for modeling and verifying concurrent interactive systems with respect to mobility and locations, there are a few model checker tools able to perform such verifications. In this thesis, a new model checking methodology is proposed which is able to verify temporal and spatial properties of systems together. The proposed model checking methodology is able to perform more detailed verifications than existing tools.

In this thesis, ambient logic and ambient calculus are used as formal languages to express models and the properties of the systems. Ambient calculus is a process calculus derived from $\pi$-calculus. It is able to theorize about concurrent systems with respect to mobility and locations. Ambient logic is a modal logic able to express temporal and spatial properties of models. It is strictly based on ambient calculus. Proposed model checking methodology accepts models expressed with a fragment of ambient calculus and properties expressed with a fragment of ambient logic as inputs. It returns a success message or the states of the model which are violating properties. In the scope of this thesis, an implementation of proposed methodology is provided

which is the only tool using both ambient calculus and ambient logic.

In this thesis, performance of the proposed model checking methodology is shown over case studies. Security policies defined for multi-domain networks need to be formally checked against security breaches and ensured that they are consistent in a given network configuration. In case studies, a set of ambient calculus specifications modeling such networks are verified against ambient logic formulas with proposed model checking methodology.

# ÖZET

## ÇEVREL CEBİR TANIMLAMALARININ ÇEVREL MANTIK FORMÜLLERİ İLE MODEL DENETLEMESİ

Biçimsel metotlar koşut zamanlı sistemlerin tanımlanmasında ve doğrulanmasında yararlanılan matematiksel tekniklerdir. Biçimsel metotların sıkça kullanılan bir örneği de model denetlemedir. Model denetleme yönteminde hedef sistemin soyutlama ile oluşturulurmuş sonlu durumlu bir modeli üzerinde tüm durumların doğrulandığı kapsamlı bir inceleme yürütülür.

Sistemlerin zamana bağlı davranışlarını inceleyen model denetleyiciler mevcuttur. Yakın zamanda iki yeni kavram, konumlar ve konum değiştirme, model denetlemede önem kazanmıştır. Konum ve konum değiştirme kavramlarını kapsayan çeşitli model denetleme yöntemleri üzerine algoritmalar önerilmişse de, bu tarz bir denetleme yapabilen çok az model denetleme uygulaması gerçeklenmiştir. Bu tezde koşut zamanlı sistemlerin zamansal ve uzaysal davranışlarını inceleyebilen yeni bir model denetleme metodolojisi önerilmektedir. Önerilen bu model denetleme metodolojisi ile sistemlerin uzaysal davranışları var olan model denetleme uygulamalarından daha kapsamlı bir şekilde incelenebilmektedir.

Çevrel cebir ve çevrel mantık bu tez kapsamında modellerin ve sistem özelliklerinin ifade edilmesinde kullanılacak biçimsel dillerdir. Çevrel cebir $\pi$-cebrinden türetilmiş bir işlev cebridir. Çevrel cebir ile koşut zamanlı sistemler, konumlar ve konumsal değişiklikler üzerinden modellenebilir. Çevrel mantık koşut zamanlı sistem modellerinin zamansal ve uzaysal özeliklerinin belirtilebildiği bir kipler mantığıdır. Çevrel mantık çevrel cebir üzerine bina edilmiştir. Önerilen model denetleme metodolojisi girdi olarak bir koşut zamanlı sistemin çevrel cebrin bir alt kümesi ile ifade edilmiş bir modelini ve çevrel mantığın bir alt kümesi ile belirtilmiş sistem özelliklerini alır.

Model denetleyici çıktı olarak ya başarı mesajı döner ya da verilen sistem özelliklerinin sağlanmadığı durumları döner. Önerilen model denetleme metodolojisi çevrel cebir ve çevrel mantık kullanan ve hali hazırda gerçeklenmiş olan tek araçtır.

Bu tezin kapsamında, gerçeklenmiş olan araç için performans sonuçlarının durum çalışmaları üzerinden gösterilmesi de bulunmaktadır. Çok etki alanlı ağlar için tanımlanmış güvenlik politikalarının güvenlik açıklıkları bulundurmadıkları ve belirli bir ağ yapılandırmasında tutarlı oldukları biçimsel yöntemlerle doğrulanmalıdır. Bu tezdeki durum çalışmalarında çevrel cebir ile modellenmiş çok etki alanlı ağlar, çevrel mantık ile belirtilmiş özelliklere karşı, önerilen model denetleyici ile doğrulanmaktadır.

# TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . . iii
ABSTRACT . . . . . iv
ÖZET . . . . . vi
LIST OF FIGURES . . . . . x
LIST OF TABLES . . . . . xii
LIST OF SYMBOLS/ABBREVIATIONS . . . . . xiv
1. INTRODUCTION . . . . . 1
1.1. Motivation . . . . . 3
2. PRELIMINARIES . . . . . 5
2.1. State Transition Systems . . . . . 5
2.2. Computation Tree Logic . . . . . 6
2.2.1. Operators . . . . . 6
2.3. Ambient Calculus . . . . . 7
2.3.1. Ambient . . . . . 10
2.3.2. Name . . . . . 10
2.3.3. Name Restriction . . . . . 12
2.3.4. Inactivity . . . . . 12
2.3.5. Parallel Composition . . . . . 12
2.3.6. Replication . . . . . 13
2.3.7. Capabilities . . . . . 13
2.3.8. Communication . . . . . 13
2.3.9. Variables . . . . . 13
2.4. Ambient Logic . . . . . 14
2.4.1. Excluded Connectives . . . . . 20
2.4.2. Derived Connectives . . . . . 20
3. PROPOSED MODEL CHECKING METHODOLOGY . . . . . 23
3.1. Approach to the Problem . . . . . 25
3.2. Internal Representation of Specifications and Formulas . . . . . 27
3.3. Reduction Of Ambient Logic Formulas . . . . . 29

3.3.1. Nesting Problem of Temporal Operators . . . . . 31
3.4. State Transition System Generation . . . . . 33
3.5. Checking Spatial Modalities of Ambient Logic . . . . . 37
3.5.1. Auxiliary Functions . . . . . 38
3.5.1.1. Wildcard Search . . . . . 38
3.5.1.2. Guessing Expected Ambients . . . . . 39
3.5.1.3. Searching Sublocation . . . . . 40
3.5.2. Matching Processes of Ambient Logic Connectives . . . . . 43
3.5.2.1. Locations . . . . . 43
3.5.2.2. Negation . . . . . 43
3.5.2.3. Disjunction . . . . . 44
3.5.2.4. Somewhere . . . . . 44
3.5.2.5. Parallel Composition . . . . . 44
3.6. Kripke Structure Generation . . . . . 47
3.7. NuSMV Code Generation . . . . . 48
4. COMPLEXITY ANALYSIS . . . . . 52
4.1. Time Complexity . . . . . 52
4.1.1. Time Cost of The Match Process at Parallel Composition Connective . . . . . 53
4.1.2. Time Cost of The Match Process at Somewhere Connective . . . 54
4.2. Space Complexity . . . . . 55
5. CASE STUDIES . . . . . 56
6. RELATED WORK . . . . . 60
7. CONCLUSION . . . . . 63
APPENDIX A: Ambient calculus specifications used at case studies . . . . . 65
APPENDIX B: Ambient logic formulas used at case studies . . . . . 67
REFERENCES . . . . . 68

# LIST OF FIGURES

Figure 2.1. (a) is a Kripke Structure. (b) is computation tree of the Kripke Structure at (a) . . . . . 6

Figure 2.2. Semantics of temporal operators of CTL. Light gray area shows the states where p is satisfied. Dark gray area shows the states where q is satisfied. . . . . 7

Figure 2.3. CTL Syntax. $\phi$ ranges over a set of atomic propositions . . . . . 8

Figure 2.4. Mobility and communication primitives . . . . . 11

Figure 2.5. Syntax of ambient logic . . . . . 17

Figure 3.1. Model checking tasks. . . . . 24

Figure 3.2. Block diagram of the proposed model checking process . . . . . 26

Figure 3.3. Internal representation of state information. Graph at (a) is ambient topology of state where graph at (b) is capability tree. . . . . 28

Figure 3.4. Internal representation of F = ◊ { ⋄ { m[l[j[]] | T] | T }} ∨ □ { ⋄ { k[{r[] | T } ∨ 0 ] | T}} as a graph. . . . . 30

Figure 3.5. The CTL formula after decomposition of graph at Figure 3.4 into temporal and spatial subgraphs. . . . . 31

Figure 3.6. The spatial formulas after decomposition of graph at Figure 3.4 into temporal and spatial subgraphs. . . . . 32

Figure 3.7. State transition system after all states are generated for P = { m[in k. open l.n[] | k[out m.in j.r[]]] | l[in m. in k. j[]] } . Labels on the edges shows the capability causes the transition. . . . . 35

Figure 3.8. Details of states at Figure 3.7. . . . . 36

Figure 3.9. The evaluation of *wildcard* and *guessExpectedAmbients* functions for $F ::= n1[n2[]] \mid n3[] \vee n5[T] \vee \diamond n[10] \mid \diamond\{n16[] \mid \{n11[] \mid n12[]\} \Rightarrow n14[]\}$. Nodes having wildcard property are filled with grey. The set of the expected ambient compositions is shown in the boxes with dashed border line at the top of the related node. . . . . 41

Figure 3.10. A match example for process P = n1[] | n3[] | n4[] | n7[ n5[] | n6[]] | n8[] and formula F = $n1[] \mid \{n2[] \vee \{n3[] \mid n4[]\}\} \mid \diamond \{n5[] \mid n6[]\} \mid \neg n8[]$. Graphs consisting of rectangle nodes are ambient topologies assigned to spatial formula graph nodes. Graphs consisting of circle nodes is spatial formula graph. . . . . 46

Figure 3.11. Kripke Structure obtained by checking spatial modalities of states of state transition system at Figure 3.7 with respect to spatial formulas at Figure 3.6. . . . . 47

# LIST OF TABLES

Table 2.1. Structural congruence . . . . . 15

Table 2.2. Reductions . . . . . 16

Table 2.3. The fragment of ambient calculus used in this thesis . . . . . 16

Table 2.4. Derived connectives . . . . . 21

Table 2.5. The fragment of ambient logic used in this thesis . . . . . 22

Table 3.1. State transition system generation algorithm . . . . . 37

Table 3.2. Algorithm of *wildcard* . . . . . 39

Table 3.3. Algorithm of *guessExpectedAmbients* . . . . . 40

Table 3.4. Algorithm of *findSublocation* . . . . . 42

Table 3.5. Variable declaration example for NuSMV . . . . . 48

Table 3.6. An example of assignments of state variables at NuSMV . . . . . 49

Table 3.7. A generated NuSMV code example . . . . . 51

Table 5.1. Properties of specifications . . . . . 56

Table 5.2. State transition system generation cost. . . . . 57

Table 5.3. Properties of formulas . . . . . 57

Table 5.4. Performance results with 1 GB heapsize . . . . . 57

Table 5.5. Performance results with 8 MB heap size . . . . . 58

Table 5.6. Performance results of NuSMV with generated code . . . . . 58

# LIST OF SYMBOLS/ABBREVIATIONS

| | |
|---|---|
| $G_{AT}$ | Ambient Topology Graph |
| $N_{AT}$ | Set of Ambient Topology Nodes |
| $A_{AT}$ | Set of Ambient Topology Arcs |
| $G_{CT}$ | Capability Tree |
| $N_{CT}$ | Set of Capability Tree Nodes |
| $A_{CT}$ | Set of Capability Tree Arcs |
| $G_F$ | Formula Graph |
| $N_F$ | Set of Formula Graph Nodes |
| $A_F$ | Set of Formula Graph Arcs |
| $N_L$ | Set of Location Nodes |
| $N_{Binary}$ | Set of Binary Connective Nodes |
| $N_{Unary}$ | Set of Unary Connective Nodes |
| $N_{PC}$ | Set of Parallel Composition Nodes |
| $A_{PC}$ | Set of Parallel Composition Arcs |
| $A_{Binary}$ | Set of Binary Connective Arcs |
| $A_{Unary}$ | Set of Unary Connective Arcs |
| $l$ | Number of Location Nodes |
| $d_w$ | Number of Disjunctions with Wildcard Property |
| $d$ | Number of Disjunctions with Wildcard Property |
| $not$ | Number of Negations |
| $sw_w$ | Number of Somewhere Connectives with Wildcard Property |
| $sw_w$ | Number of Somewhere Connectives with Wildcard Property |
| $a_e$ | Number of Expected Ambients |
| $a_{ne}$ | Number of Unexpected Ambients |
| | |
| $\Pi$ | The set of Processes |
| $\Phi$ | The set of Formulas |
| $\vartheta$ | The set of Variables |
| $\Lambda$ | The set of Names |

| | |
|---|---|
| $\phi$ | The set of Atomic Propositions |
| LTL | Linear-time Temporal Logic |
| CTL | Computation Tree Logic |

# 1. INTRODUCTION

In computer science, formal methods are mathematical languages, techniques and tools which can be used at specification and verification of various systems. They are applied in specification and verification of broad range of systems when high integrity systems like hardware, software, security protocols and biological systems are subjects. The use of formal methods at describing the system and its properties is called formal specification. When they are used as specification instruments they are expected to provide designs which is unambiguous, more reliable and more robust. Formal methods also can be applied to the work of verification of systems, which is called formal verification. Formal verification is supposed to provide building proofs or disproofs of desired properties that system must hold. It can be used also for finding anomalies at behaviors of the system. Some principal techniques of formal verification are simulation, testing, automated theorem proving and model checking.

One of the main stream approaches at formal verification is model checking which can be defined as building a finite model of system and performing an exhaustive search. The main idea in model checking is defining an automatic mechanism which explores all of the possible states that the system can be found in and test these states if a set of desired properties is hold in or not. Model checking must be fully automatic that it should not need user interaction when the formal specifications of system and properties are provided. It is clear that the set of the possible states of system must be finite to perform such an exploration and check over the system. Because of the set of states are finite, the search over these states can be reduced to a graph search. Various types of systems can be subjects of model checking. An important application area of the model checking is the analysis of security protocols.

The concept of the analysis that model checkers do, can be diverse. Most of current model checkers are based on temporal analysis of the behaviors of the systems. There exits well developed model checking tools for formal languages where analyzed modalities are generally temporal. But recently another notion of analysis comes into

prominence beside the temporal behaviors, the mobility. There are model checkers which include a limited representation of mobility. They are limited because of lacking a key notion for mobility, locations. The location notion is important because the separation of the environments, that mobility is performed through, must be distinguished by some construct. For example when modeling the movements of user between domains in a multi-domain network, a construct is needed to represent the domains of network in the model as the environment that mobility happens in. While there are proposals for modeling concurrent interactive systems with respect to mobility and locations, there exists a few model checker for analyzing models build by these proposals. In this thesis a new model checking methodology is proposed, which is able to check a formal language specification including mobility aspects against a logic that is capable of representing both temporal and spatial (with notion of location) modalities.

Process algebras are mathematical tools used for modeling real world systems for decades. All process algebras have special modeling aspects like modeling communication between interactive systems. Ambient calculus is a process algebra which is focused on modeling mobile computation and mobile processes. Ambient calculus is a useful tool to model complex systems because of its facilities in expressing hierarchies of locations and their mobility [14, 15]. It provides various constructs for representing spatiality and mobility. This thesis uses ambient calculus as formal language for specifying system models.

Like other examples of process calculi, ambient calculus it not capable of specifying properties of system models. In computer science such properties can be specified formally by a modal logic. A modal logic is a formalism that attempts to deal with modalities which are represented by modal operators. The most known modal logics are temporal logics. But for the system and calculus in which main aspect is mobility and spatiality, some extra modalities is needed for dealing with mobility and spatial properties. Ambient logic is a modal logic to express properties of mobile computations for temporal and spatial modalities [2]. Ambient logic is strictly based on ambient calculus. It can describe the structural and behavioral properties of constructs of ambient calculus. This thesis uses ambient logic for specifying properties of the system models.

This thesis proposes a new methodology for model checking where model checking problem is formalized as: given a desired property, expressed as an Ambient Logic Formula $\mathscr{A}$, and a model specification, expressed as a Ambient Calculus specification $P$, decide if $P \vDash \mathscr{A}$ i.e. $\mathscr{A}$ is hold in model P. This thesis offers a complete definition of a model checking tasks with algorithms and implementation details. To minimize the size of model checking problem, reduced fragments of ambient logic and ambient calculus are used. In the proposed methodology an existing model checker is benefited for temporal modalities. Despite of reduced fragments of the logic and the calculus, the proposed methodology is sufficient to verify concurrent systems with respect temporal and spatial modalities. A set of case studies at networking is presented at this thesis to show the sufficiency of implementation of proposed methodology for verifying concurrent systems.

The thesis is organized as follows: In Chapter 2, transition systems, computational tree logic, ambient calculus and ambient logic will be introduced. In Chapter 3, the proposed methodology is presented with its abilities and limitations. In Chapter 4, complexity of the the proposed solution is discussed. In Chapter 5, case studies are shown. In Chapter 6 other existing methods and comparisons with these methods are discussed. Chapter 7 concludes the work.

## 1.1. Motivation

As the network grows, use of dividing the network into multiple domains simplifies administration of network. Multi domain networks can be used in several scenarios including military and enterprize networks. In such networks users are allowed to use network connectivity of multiple domains. But each domain might be under different administrations which regulates the activities of the users with respect to their domain configurations. Multiple domain networks are especially fitting to organizations that span multiple locations, information classifications, or business functions.

When dealing with multi-domain a new concept comes into consideration, roaming. Roaming can be identified as traversing among domains. Roaming users present

challenges for authorization and access control mechanisms in environments with multiple domains. Authorization and access control mechanisms determine the access rights for roaming users are more complicated than single domain.

A security policy determines security mechanisms at administrative domains as actions that users capable of over the resources of these domains. As being defined for a single administrative domain, security policies also can be defined for a collection of administrative domains, multiple administrative domains. In their ongoing work Ünal and Çağlayan propose a framework for specification and verification of security policies of multiple domain mobile networks [1]. In their framework security policies also model the mobility of roaming users and the resources. This framework makes use of ambient calculus to formally define the mobile processes and ambient logic to formally define the restriction and the conditions about these processes. In their framework security policies need to be formally checked against security breaches. Also security policies must be ensured by using a formal method that they are consistent in a given network configuration.

Although there are implementations of various model checkers and algorithms for model checking of ambient calculus against ambient logic, there is not any implementation of an ambient calculus model checker which using ambient logic. One of the main outputs of the this thesis is an implementation of a model checking tool can be used at formal verification of the security policies in the framework of [1].

# 2. PRELIMINARIES

## 2.1. State Transition Systems

A state transition system is an abstract machine used in the study of computation. A transition system model a system by means of states and transitions between states. In a state transition system the set of states and the set of transitions are not necessarily finite, or even countable. The main difference of state transition system from automate is that states are labeled instead of transitions in state transition systems. State transition systems can be represented as digraphs. Formally, an unlabeled state transition system is a tuple (S, R) where

- S is a set of states
- R $\subset$ S $\times$ S is a binary relation over S.

A Kripke Structure is a kind of state transition system where states are labeled. Labels of a state is a set of atomic propositions which hold in that state. Atomic propositions can be considered as the marking of systems properties.

**Definiton 2.1.1 (Kripke Structure)** *Let AP be a non-empty set of atomic propositions. A Kripke Structure is a four-tuple; M = (S, S0, R, L) where*

- *S is a finite set of states.*
- *S0 $\subseteq$ S, is the set of initial states.*
- *R $\subseteq$ S$\times$ S, is a transition relation*
- *L: S $\rightarrow$ $2^{AP}$ is a function that labels each state with the set of atomic propositions true in this state.*

A computation tree of a state transition system is a tree visualizing all possible computation paths.

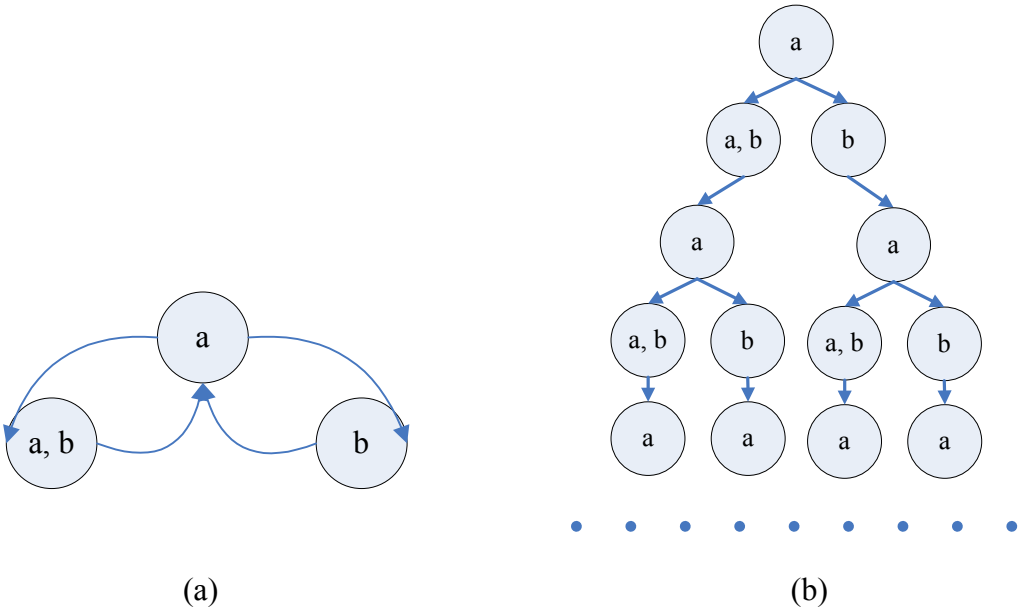


Figure 2.1. (a) is a Kripke Structure. (b) is computation tree of the Kripke Structure at (a)

## 2.2. Computation Tree Logic

Temporal logic is a formalism for representing, and reasoning about, propositions qualified in terms of time. Computation tree logic (CTL) is a type of temporal modal logic. Temporal logics may differ according to how they handle branching in the underlying computation tree. CTL's model of time is a tree-like structure in which formulas reason about many executions at once. CTL formulas are interpreted over transitions systems like Kripke Structures, linearized as trees. CTL is used in formal verification of software or hardware systems by model checkers like SPIN and NuSMV.

### 2.2.1. Operators

CTL is given by the standard boolean logic enhanced with temporal operators. In CTL, temporal operators are separated in two group, path quantifiers and linear-time operators. All linear-time operators must always be quantified by path quantifier operator.

- Logical operators

  The logical operators are : $\neg$,$\vee$,$\wedge$,$\Rightarrow$ and $\Leftrightarrow$. Along with these operators CTL formulas can also make use of the boolean constants $true$ and $false$.

- Temporal operators
  - Quantifiers over paths
    - A - for every path
    - E - for some path
  - Path-specific quantifiers
    - X - holds at next state
    - G - holds at the entire subsequent path.
    - F - holds eventually somewhere on the path
    - U - holds until.

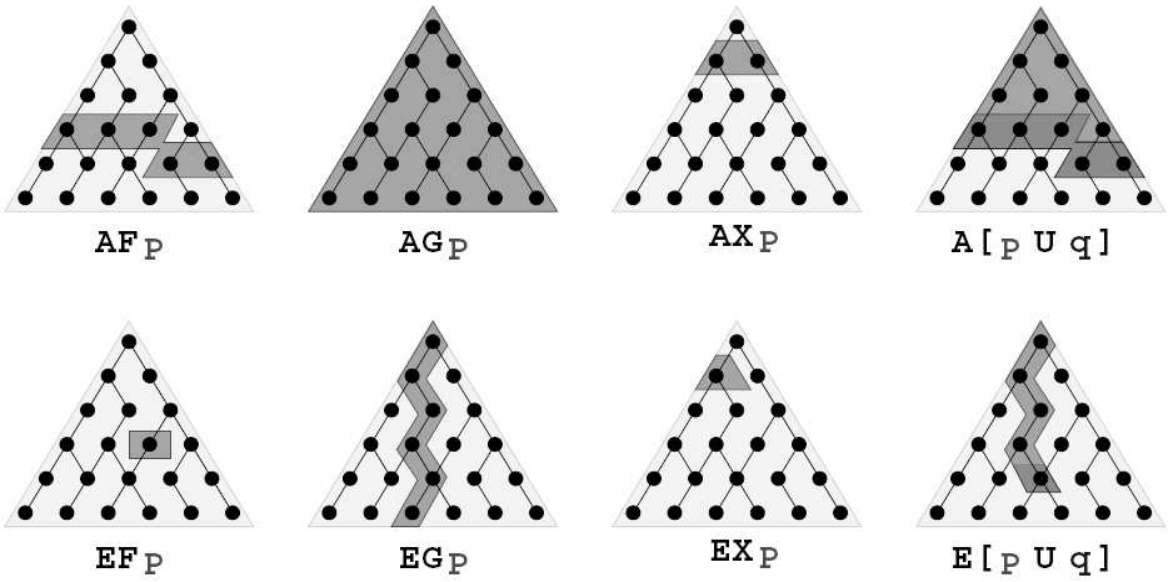


Figure 2.2. Semantics of temporal operators of CTL. Light gray area shows the states where p is satisfied. Dark gray area shows the states where q is satisfied.

## 2.3. Ambient Calculus

The process calculi (or process algebras) are diverse family of related approaches for modeling interactive concurrent systems formally. The process calculi focus on

| | | |
|---|---|---|
| $\mathscr{F}$, $\mathscr{G}$ | ::= | |
| | | $T$ |
| | | $F$ |
| | | $\phi$ |
| | | $\neg\mathscr{F}$ |
| | | $\mathscr{F} \vee \mathscr{G}$ |
| | | $\mathscr{F} \wedge \mathscr{G}$ |
| | | $\mathscr{F} \Rightarrow \mathscr{G}$ |
| | | $\mathscr{F} \Leftrightarrow \mathscr{G}$ |
| | | $AX\mathscr{F}$ |
| | | $EX\mathscr{F}$ |
| | | $AF\mathscr{F}$ |
| | | $EF\mathscr{F}$ |
| | | $AG\mathscr{F}$ |
| | | $EG\mathscr{F}$ |
| | | $A[\mathscr{F}U\mathscr{G}]$ |
| | | $E[\mathscr{F}U\mathscr{G}]$ |

Figure 2.3. CTL Syntax. $\phi$ ranges over a set of atomic propositions

expressing concurrency, synchronization, and communication. They also provide algebraic laws that allow process descriptions to be manipulated and analyzed. By these algebraic laws, details of internal computations of system is abstracted in the model. There exits various types of process calculi where each of them focusing different aspect of computation. Their common properties can be listed as parallel composition of processes, sequencing of interactions, hiding of information and recursion or process replication. Ambient calculus, proposed by Cardelli and Gordon, is a process calculus which is able to theorize about concurrent systems that include mobility and locations [4]. Ambient calculus based on notion ambient which represents a bounded place (location) in which computation can occur or a resource can be nested in. It is asserted that ambient calculus is a very expressive process calculus to specify spatial and temporal behaviors of concurrent systems [14, 15]. This thesis uses ambient calculus for specifying system models. Although there are typed versions of ambient calculus in the proposed thesis untyped monadic version of ambient calculus is used.

Like all process algebras, ambient calculus relies on the notion of process. Processes are the exact counterparts of real world system elements at model. Properties of processes at Ambient Calculus can be analyzed in two separate views, spatial properties and temporal properties.

The notion of ambient is the basic element of the spatiality at processes. Ambients are bounded places identified by a name where processes reside in or out. Ambients can be nested in other ambients. This provides hierarchical organization of locations.

All process algebras have constructs which resembles the changes of the real world systems over time, called temporal constructs. Most of the calculi provide temporal construct as communication primitives. This approach is adequate to represent the interaction between parallel processes. Because the main aspect of ambient calculus is modeling mobility of the systems more inherently then existing calculi, ambient calculus provide temporal constructs other than communication primitives. Temporal constructs, named actions and capabilities interchangeably, are introduced in ambient calculus for rearrangement of ambient hierarchy as movement of processes or dissolving

locations. There are three main capabilities in ambient calculus as entrance, exit and dissolution which are expressed as *in n*, *out n*, *open n* respectively. Capabilities can be ordered as a sequence to represent sequential execution as well as they can exist as parallel. Their effect on ambients is regulated by the name references they contain. Capabilities ordered as sequences are called path. They must be attached to an ambient or inactivity ($0$). While the ability of change of ambient hierarchy represents the mobility, they can be used to represent every kind computation [4]. There is also communication primitives at ambient calculus which enable processes within the same ambient may exchange messages.

Syntax of ambient calculus is showed at the Figure 2.4. In this thesis a fragment of the ambient calculus is used. This fragment is described while introducing ambient calculus constructs. Main trade off at defining the fragment is between expressivity of the fragment and the implementation complexity of the model checker. The fragment described here is capable of expressing systems defined at [1].

### 2.3.1. Ambient

As mentioned before ambients are core construct of the ambient calculus. They represent a bounded place for computation. Ambients strictly define what is inside and what is outside. All interactions among processes are regulated by the ambients and their topology. Each ambient must define its boundaries with square brackets and must be identified by a name.

### 2.3.2. Name

Names are not used for identifying ambients. They are used as access keys for ambients to be used by capabilities i.e. capabilities can effect or use ambients with name known by them. Two distinct ambients can have the same name. For example in statement $n[P] \mid n[Q]$ there are two distinct ambients which are surrounding two distinct processes.

| | | | |
|---|---|---|---|
| $P, Q$ | ::= | processes | |
| | $(\nu n)P$ | | restriction |
| | $0$ | | inactivity |
| | $P \mid Q$ | | composition |
| | $!P$ | | replication |
| | $M[P]$ | | ambient |
| | $M.P$ | | capability |
| | $(x).P$ | | input |
| | $\langle M \rangle$ | | async output |
| $M$ | ::= | capabilities | |
| | $x$ | | variable |
| | $n$ | | name |
| | $in \quad M$ | | can enter into M |
| | $out \quad M$ | | can exit out of M |
| | $open \quad M$ | | can open M |
| | $\epsilon$ | | null |
| | $M.M$ | | path |

Figure 2.4. Mobility and communication primitives

But in the fragment of ambient calculus used in this thesis names are restricted to be unique. So they become an identifier in scope of this thesis. This restriction reduces the implementation complexity by eliminating renaming and bookkeeping tasks for ambients, which are not core model checking tasks.

### 2.3.3. Name Restriction

As mentioned before names are not unique identifiers for ambients in full fragment of ambient calculus. So the ambients which have same name can be accessed or used by the same set of capabilities. In ambient calculus name restriction is used for distinguishing the ambients with same names when necessary. For example in the statement $open\ n.0 \mid n[P] \mid n[Q]$ there is an capability, $open\ n$, which can dissolve ambients with the name $n$. So by consuming this capability, statement can evolve into two different future statements, $P \mid n[Q]$ and $n[P] \mid Q$. But with name restriction it is possible to hide an ambient from this capability. In the statement $openn.0 \mid n[P] \mid (vn)n[Q]$, the name $n$ is identify as a private name in the scope of $n[Q]$ and distinguished other uses at the same statement. So the capability $open\ n$ is not applicable to second ambient at the statement anymore. This operator is used for name hiding to provide hidden locations and resources. Because names are unique identifiers in the fragment of ambient calculus of this thesis it is not sound to support name restriction operator. Name restriction is excluded in fragment of ambient calculus of this thesis.

### 2.3.4. Inactivity

The inactive process, 0, is a common operator that we encounter most of process calculi. It specifies the empty process which does nothing. It is not reducible.

### 2.3.5. Parallel Composition

Parallel execution of the processes is represented by this operator. It is a commutative and associative operator. It is also another common operator of the process calculi.

### 2.3.6. Replication

Replication operator denotes infinite and parallel executed copies of its operand. It is used for representing iteration and recursion. Fragments of ambient calculus with this construct make model checking problem undecidable [5]. Replication is excluded in fragment of ambient calculus of this thesis.

### 2.3.7. Capabilities

A capability path is a sequence of capabilities which contains one or more capabilities. A path must be attached to an ambient or inactivity process. A path is executed sequentially from starting the first capability of the sequence. There are three kinds of capabilities: one for entering an ambient, one for exiting from an ambient and one for opening up an ambient. Capabilities contain references to their objects as names. Given a name $n$, the capability $in\ n$ reduces as an entrance into ambient named n, the capability $out\ n$ reduces as exit from ambient named n and the capability open n reduces as dissolution of ambient named n. Operational semantics of the capabilities are shown at Table 2.3.

### 2.3.8. Communication

In ambient calculus communication constructs are asynchronous and local to an ambient. Ambient calculus does not support channel names for communication where some other process calculi do. While communication is used to exchange both names and capabilities in the full fragment, communication of the capabilities is excluded in fragment of ambient calculus of this thesis. Operational semantics of the communication primitives are shown at Table 2.3.

### 2.3.9. Variables

Variables are place holders for names at the ambient calculus processes where an input operation is included at capability path. When an output operation provides a

name to an input operation every instance of variable in the scope of input capability is replaced with incoming name.

The semantics of ambient calculus is based on structural congruence relation. structural congruence identifies processes up to elementary spatial rearrangements [4].

The dynamic properties of ambient calculus originate from capabilities and communication primitives. This set of construct of ambient calculus is called temporal constructs. The semantics of these construct are identified by reduction relations. Table 2.3 shows the reduction rules for temporal constructs.

$$
\begin{aligned}
\text{EC} \quad ::= \quad & World[Domain[Server[key[\langle message\rangle \mid out\ Server.in\ Client.0]] \mid \\
& Client[open\ key.(x).x[in\ folder.0] \mid folder[]]]] \\
\rightarrow \quad & World[Domain[Server[]] \mid key[\langle message\rangle \mid in\ Client.0] \mid \\
& Client[open\ key.(x).x[in\ folder.0] \mid folder[]]] \\
\rightarrow \quad & World[Domain[Server[]] \mid Client[key[\langle message\rangle] \mid \\
& open\ key.(x).x[in\ folder.0] \mid folder[]]] \\
\rightarrow \quad & World[Domain[Server[]] \mid Client[\langle message\rangle \mid (x).x[infolder.0] \mid \\
& folder[]]] \\
\rightarrow \quad & World[Domain[Server[]] \mid Client[message[infolder.0] \mid folder[]]] \\
\rightarrow \quad & World[Domain[Server[]] \mid Client[folder[message[]]]]
\end{aligned}
$$

At above an ambient calculus process example is shown with its evolution resulted from reduction of capabilities. Process EC models an encrypted messaging from Server to Client. Encryption of message is encoded by ambient named *key*.

## 2.4. Ambient Logic

As mentioned before modal logics are used for expressing properties of models which cannot be expressed by the constructs of calculi. Ambient logic of Cardelli and

Table 2.1. Structural congruence

| | |
|---|---|
| $P \equiv P$ | (Struct Refl) |
| $P \equiv Q \Rightarrow Q \equiv P$ | (Struct Symm) |
| $P \equiv Q, Q \equiv R \Rightarrow P \equiv R$ | (Struct Trans) |
| | |
| $P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$ | (Struct Res) |
| $P \equiv Q \Rightarrow P\|R \equiv Q\|R$ | (Struct Par) |
| $P \equiv Q \Rightarrow !P \equiv !Q$ | (Struct Repl) |
| $P \equiv Q \Rightarrow n[P] \equiv n[somewhere Q]$ | (Struct Amb) |
| $P \equiv Q \Rightarrow M.P \equiv M.Q$ | (Struct Capability) |
| | |
| $P\|Q \equiv Q\|P$ | (Struct Par Comm) |
| $(P\|Q)\|R \equiv P\|(Q\|R)$ | (Struct Par Assoc) |
| $!P \equiv P\|!P$ | (Struct Repl Par) |
| | |
| $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$ | (Struct Res Res) |
| $(\nu n)(P\|Q) \equiv P\|(\nu n)Q \text{if} n \notin fn(P)$ | (Struct Res Par) |
| $(\nu n)(m[P]) \equiv m[(\nu n)P] \text{if} n \neq m$ | (Struct Res Amb) |
| | |
| $P\|0 \equiv P$ | (Struct Zero Par) |
| $(\nu n)0 \equiv 0$ | (Struct Zero Res) |
| $!0 \equiv 0$ | (Struct Zero Repl) |
| | |
| $P \equiv Q \Rightarrow (x).P \equiv (x).Q$ | (Struct Input) |
| $\epsilon.P \equiv P$ | (Struct $\epsilon$) |
| $(M.M').P \equiv M.M'.P$ | (Struct .) |

Table 2.2. Reductions

| | |
|---|---|
| $n[inm.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$ | (Red In) |
| $m[n[outm.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$ | (Red Out) |
| $openn.P\|n[Q] \rightarrow P\|Q$ | (Red Open) |
| | |
| $P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$ | (Red Res) |
| $P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$ | (Red Amb) |
| $P \rightarrow Q \Rightarrow P\|R \rightarrow Q\|R$ | (Red Par) |
| | |
| $P \equiv P, P \rightarrow Q, Q \equiv Q \Rightarrow P \rightarrow Q$ | (Red $\equiv$) |
| $(x).P\|\langle n\rangle \rightarrow P\{x \leftarrow n\}$ | (Red Comm) |
| $\rightarrow^*$ | reflexive and transitive closure of $\rightarrow$ |

Table 2.3. The fragment of ambient calculus used in this thesis

| Construct | Expression | Status |
|---|---|---|
| Name | $n, m$ | restricted to be unique |
| Name restriction | $\nu$ | excluded |
| Variable | $x$ | included |
| Parallel composition | \| | included |
| Capabilities | *in* | included |
| | *out* | |
| | *open* | |
| Inactivity | 0 | included |
| Replication | ! | excluded |
| Prefixing | . | included |
| Ambient | [] | included |
| Input | () | included |
| Output | $\langle\rangle$ | restricted to only names can be communicated |

Gordon is very expressive modal logic for expressing spatial and temporal properties of ambient calculus [2]. Ambient logic is strictly based on ambient calculus i.e. all spatial and temporal constructs are reflected in the logic. The main differences of ambient logic from latter logics are more expressive space modalities and simpler temporal connectives [14, 15]. With full fragment of ambient logic it is possible to derive formulas expressing capabilities of processes for movement and for communication, as well as the persistence of processes, and free occurrences of names in processes. In this thesis a fragment of ambient logic is used.

| | | |
|---|---|---|
| $\eta, \mu$ | | a name n or a variable x |
| $\mathscr{A}, \mathscr{B}, \mathscr{C}$ | $::=$ | |
| | $T$ | true |
| | $\neg\mathscr{A}$ | negation |
| | $\mathscr{A} \vee \mathscr{B}$ | disjunction |
| | $0$ | void |
| | $\mathscr{A} \mid \mathscr{B}$ | composition |
| | $\mathscr{A} \rhd \mathscr{B}$ | guarantee |
| | $n[\mathscr{A}]$ | location |
| | $\mathscr{A}@\eta$ | placement |
| | $\eta \textcircled{R} \mathscr{A}$ | revelation |
| | $\mathscr{A} \oslash \eta$ | hiding |
| | $\Diamond\mathscr{A}$ | sometime modality |
| | $\diamond\mathscr{A}$ | somewhere modality |
| | $\forall x.\mathscr{A}$ | universal quantification |

Figure 2.5. Syntax of ambient logic

Ambient logic has temporal and spatial modalities in addition to propositional logic elements. At the Figure 2.5 primitive connectives of full fragment of the ambient logic are shown. Other connectives can be derived from these primitive connectives.

Semantics of the connectives of the ambient logic will be given through satisfaction

relations. The definition of satisfaction is based heavily on the structural congruence relation [2]. The semantics of the connectives included in the fragment of ambient logic used in this thesis are introduced at definitions below. These definitions are taken from [2]. The satisfaction relation is denoted by $\vDash$ symbol. To express the process P satisfies the formula $\mathscr{A}$, $P \vDash \mathscr{A}$ is used. The symbol $\Pi$ denotes the set of processes, $\Phi$ denotes the set of formulas, $\vartheta$ denotes the set of variables, and $\Lambda$ denotes the set of names.

Names are used in logic in the same manner they used in ambient calculus. Although names are restricted to be unique at ambient calculus, they do not have to be unique at ambient logic formulas. Variables are used only at name quantification operator. The fragment of ambient logic does not include variables and name quantification.

**Definiton 2.4.1** *The atomic formula T of ambient logic is satisfied by all processes of ambient calculus.*

$$\forall P \in \Pi . P \vDash T$$

**Definiton 2.4.2** *Negation of a formula is satisfied by any process which does not satisfy original formula.*

$$\forall P \in \Pi, \mathscr{A} \in \Phi . P \vDash \neg \mathscr{A} \triangleq \neg P \vDash \mathscr{A}$$

**Definiton 2.4.3** *A process satisfies the formula $\mathscr{A} \vee \mathscr{B}$ if it satisfies either $\mathscr{A}$ or $\mathscr{B}$.*

$$\forall P \in \Pi, \mathscr{A}, \mathscr{B} \in \Phi . P \vDash \mathscr{A} \vee \mathscr{B} \triangleq P \vDash \mathscr{A} \vee P \vDash \mathscr{B}$$

**Definiton 2.4.4** *The formula 0 is satisfied by only processes structurally congruent to inactivity process.*

$$\forall P \in \Pi . P \vDash 0 \triangleq P \equiv 0$$

**Definiton 2.4.5** *The formula $n[\mathscr{A}]$ is satisfied by processes which are structurally congruent to $n[P']$ for any P' where $\mathscr{A}$ is satisfied by $P'$.*

$$\forall P \in \Pi, n \in \Lambda, \mathscr{A} \in \Phi. P \vDash n[\mathscr{A}] \stackrel{\triangle}{=} \exists P' \in \Pi.\ P' \vDash \mathscr{A} \wedge P \equiv n[P']$$

**Definiton 2.4.6** *The formula $\mathscr{A}|\mathscr{B}$ is satisfied by any process that can be decomposable into two processes as $P'|P''$ where $P'$ satisfies $\mathscr{A}$ and $P''$ satisfies $\mathscr{B}$.*

$$\forall P \in \Pi, \mathscr{A}, \mathscr{B} \in \Phi. P \vDash \mathscr{A}|\mathscr{B} \stackrel{\triangle}{=} \exists P', P'' \in \Pi.\ P \equiv P'|P'' \wedge P' \vDash \mathscr{A} \wedge P'' \vDash \mathscr{B}$$

**Definiton 2.4.7** *The nesting relation, denoted by $\downarrow$, is defined over two processes as $P \downarrow Q$ and indicates that Q is nested one level down in any ambient which exists at the top of the topology of P.*

$$P \downarrow P' \quad \text{iff } \exists\, n, P''. P \equiv n[P']|P''$$

**Definiton 2.4.8** *Relation $\downarrow^*$ is reflexive transitive closure of $\downarrow$. $P \downarrow^* Q$ indicates that P contains Q in somewhere of its topology.*

$$\downarrow^* \text{ is the reflexive and transitive closure of } \downarrow$$

**Definiton 2.4.9** *Somewhere connective, $\diamond$, is used for specifying nesting properties of processes on the basis provided by Definition 2.4.8. The formula $\diamond\mathscr{A}$ is satisfied by processes which satisfies A in some inner location.*

$$\forall P \in \Pi, \mathscr{A} \in \Phi. P \vDash \diamond\mathscr{A} \stackrel{\triangle}{=} \exists P' \in \Pi.\ P' \vDash \mathscr{A} \wedge P \downarrow^* P'$$

**Definiton 2.4.10** *Sometime connective, $\Diamond$, is used for specifying temporal behavior of the processes on the basis provided by reduction relations ($\rightarrow$) defined at section 2.3.*

*The relation $\rightarrow^*$ is reflexive transitive closure of reduction relation. $\Diamond\mathscr{A}$ is satisfied by processes which can evolve into a future process holding $\mathscr{A}$.*

$$\forall P \in \Pi, \mathscr{A} \in \Phi. P \vDash \Diamond\mathscr{A} \triangleq \exists P' \in \Pi.\ P' \vDash \mathscr{A} \wedge P \rightarrow^* P'$$

### 2.4.1. Excluded Connectives

The formula $\mathscr{A}@n$, placement, is satisfied by processes which can satisfy $\mathscr{A}$ when they placed into an ambient named n. Location adjunct connective is not included in the fragment of ambient logic used in thesis.

A reading of $P \vDash \mathscr{A} \triangleright \mathscr{B}$ is that P (together with its context) manages to satisfy $\mathscr{B}$ under any possible attack by an opponent that is bound to satisfy $\mathscr{A}$ [2]. This connective is not included in the fragment of ambient logic used in thesis because it makes model checking process undecidable [5].

The connectives revelation, $\oslash$, and hiding, Ⓡ, operate on restricted names and their scopes. Because name restriction is not included at calculus these connectives are not included in the fragment of ambient logic used in thesis.

Name quantification ranges over names. It maps all possible names to a formula with respect to a variable. This connective is not included in the fragment of ambient logic used in thesis.

### 2.4.2. Derived Connectives

Some of the connectives of the ambient logic are derivable from the primitive connectives. Because of this, these connectives are not included in the fragment of logic directly. Formulas including these connectives must be rewritten with connectives of the primitive set. In table 2.4 these derived connectives are shown. In table 2.5 the fragment of ambient logic used at this thesis is summarized.

Table 2.4. Derived connectives

| Expression | Derivation | Connective |
|---|---|---|
| F | $\neg T$ | False |
| $\mathscr{A} \wedge \mathscr{B}$ | $\neg(\neg\mathscr{A} \ \vee \ \neg\mathscr{B})$ | Conjunction |
| $\mathscr{A} \Rightarrow \mathscr{B}$ | $\neg\mathscr{A} \ \vee \ \mathscr{B}$ | Implication |
| $\mathscr{A} \Leftrightarrow \mathscr{B}$ | $(\mathscr{A} \Rightarrow \mathscr{B}) \wedge (\mathscr{B} \Rightarrow \mathscr{A})$ | Logical Equivalence |
| $\mathscr{A} \parallel \mathscr{B}$ | $\neg(\neg\mathscr{A} \mid \neg\mathscr{B})$ | Decomposition |
| $\mathscr{A}^{\forall}$ | $\mathscr{A} \ \parallel \ T$ | Every component satisfies $\mathscr{A}$ |
| $\mathscr{A}^{\exists}$ | $\mathscr{A} \mid F$ | Some component satisfies $\mathscr{A}$ |
| $\Box\mathscr{A}$ | $\neg\Diamond\neg\mathscr{A}$ | Everytime modality |
| $\square\mathscr{A}$ | $\neg \diamond \neg\mathscr{A}$ | Everywhere modality |
| $\mathscr{A} \Rightarrow \mathscr{B}$ | $\neg(\mathscr{A} \mid \neg\mathscr{B})$ | Fusion adjunct |
| $n[\Rightarrow \mathscr{A}]$ | $\neg n[(\neg\mathscr{A}]$ | If there is an $n$, its contents satisfies $\mathscr{A}$ |

Table 2.5. The fragment of ambient logic used in this thesis

| Construct | Expression | Status |
|---|---|---|
| name | $n, m$ | Included |
| variable | $x$ | Not included |
| void | 0 | Included |
| Parallel composition | $\mid$ | Included |
| Negation | $\neg$ | Included |
| Disjunction | $\vee$ | Included |
| Guarantee | $\triangleright$ | Not included because of decidability issues |
| Location | $[]$ | Included |
| Location Adjunct | @ | Not Included |
| Revelation | Ⓡ | Not included because name restriction is not supported at the fragment |
| Hiding | $\oslash$ | Not included because name restriction is not supported at the fragment |
| Sometime | $\Diamond$ | Included |
| Somewhere | $\diamond$ | Included |
| Name quantification | $\forall$ | Not included |

# 3. PROPOSED MODEL CHECKING METHODOLOGY

One of the main stream approaches at formal verification is model checking. Model checking is a model-oriented, automatic process of checking the validity of P in M, where M is a finite-state model of a system and P is a property.

Model checking can be divided into three processes as modeling, specification and verification. In model checking the core element is model of system. Models are abstract system descriptions of the original system [12]. Modeling can be seen as process of converting the system into a formalism i.e. a high level representation of system. Generally full specifications of systems hold more information than the interested properties. This brings a lot of unnecessary states to explore. An abstract representation can be obtained by removing unwanted details from the system description in such a way that properties of interest are preserved. The abstracted system description can be used to generate a smaller state space which can effectively be used in a verification process. The specification part of model checking consists of describing the model and the properties by a formal language. The specification of model and properties are the inputs of verification process. Verification process of model checking can be defined as exploring all states in a model and checking whether a certain set of properties is valid for these states. A basic algorithm can be given as converting specification of model into a data structure, like Kripke Structure, by starting from an initial state, generating successive system states by evaluating the possible inputs, checking if the given properties are hold in these states. The check process at here is an exhaustive search, where all reachable states must be visited.

Although the model checking process is guaranteed to terminate due to finite state space, the process of generating all state space will have an exponential complexity and visiting all states will be quite hard, even impossible for current hardware technology. The size of the state space for such a model depends on the number of concurrent processes, the number and range of the internal variables, the type of the exchanged messages, and the nature of the communication [13]. Several methods have

been developed for dealing with state explosion problem. In our thesis, coping with state explosion problem is not one of the main concerns.

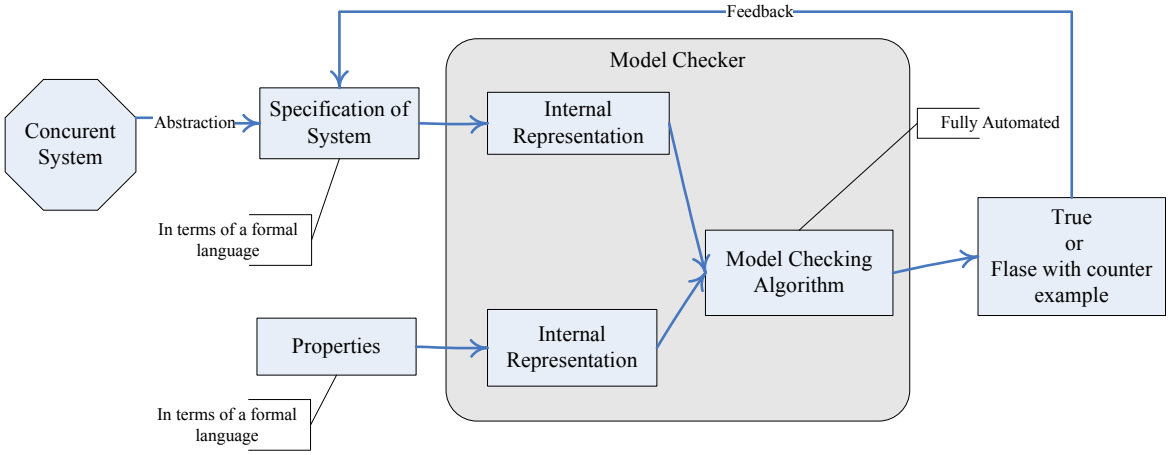


Figure 3.1. Model checking tasks.

In this thesis a model checking methodology is proposed for model checking of systems modeled as ambient calculus specifications against properties specified as ambient logic formulas. Model Checking is very large problem, especially when complex calculi and logics are the subjects. To minimize to effort for design and implementation, fragments of the calculus and the logic is taken.

The ambient logic model checking can be decomposed into two major search problems. First is searching future evolutions of a given model. Evolution of the model means the reorganizations of the spatial configuration of the model in course of time. Capabilities of ambient calculus cause changes at spatial configuration of the state which yields another (future) state. A logic formula can include constructs quantify rest of formula over future states. So when evaluating truth of a formula against an ambient calculus specification, analysis of reachable future states is needed.

Second major search problem of the model checking problem is the search of spatial congruence of model and the logic formula. Both of ambient calculus specification and ambient logic formulas have spatial patterns. This spatial pattern can be seen as a forest of the ambients. Logic formulas also contain extra constructs which does not denote spatiality directly but regulates semantics of spatial constructs. So the model

checker must match these patterns in the calculus specifications in order to evaluate the satisfaction.

## 3.1. Approach to the Problem

Modal checkers based on temporal logic is exits for more than 15 years. In rest of this paper term "temporal model checker" refers this kind of model checkers. Reusing one of these well developed and researched methodologies reduces the scale of our problem. To benefit from one of these existing methodologies we divide our problem into two sub problems as temporal modal checking and spatial model checking. The proposed algorithm based on this approach derived from the work of Mardare et al [6].

Benefiting from a temporal model checker inside the proposed model checking methodology will reduce the effort needed for implementation. Temporal model checker is used for carrying out satisfaction process for the sometime and everytime connectives of ambient logic. Nevertheless the proposed model checking methodology still has to generate all possible future states and build a state transition system from these states. This state transition system will be processed into a Kripke Structure which will be given to temporal model checker as input.

This thesis proposes model checking algorithms for spatial modalities while temporal modalities are handled by an existing temporal modal checker as black box. In this thesis NuSMV is used as temporal model checker. Outline of the proposed algorithm for the model checking problem is shown as below.

- Define atomic propositions with respect to spatial properties of ambient logic formula and register the (atomic proposition-spatial modality) couples.
- Reduce ambient logic formula to temporal logic formula (CTL) by replacing spatial modalities with atomic propositions.
- Generate state transition system of the ambient calculus specification with respect to reduction relations.
  - Create initial state from given ambient calculus specification

  - Generate new states by applying available capabilities with respect to reduction relations defined at [4].
  - Add new states to state transition system with transition relation.
- Generate Kripke Structure from state transition system
  - Assign the values of the atomic propositions for each state of state transition system(labeling) by applying model checking for spatial modalities on ambient topology of the related state.
  - Add a new state with its label (values of atomic propositions) to Kripke Structure
- Generate NuSMV code from Kripke Structure and CTL Formula.

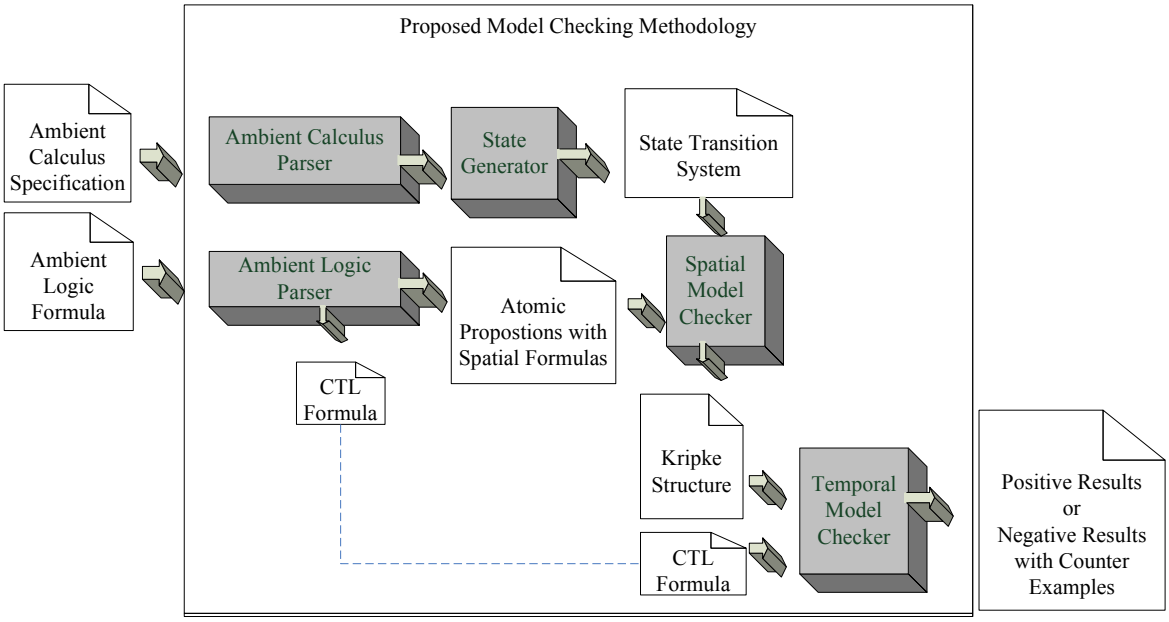


Figure 3.2. Block diagram of the proposed model checking process

This thesis takes the algorithm for generation state transition system of Mardare as base and offers a new data structure and algorithm which enhances calculation time of state transition system.

In the work [5, 10] exhaustive search is offered for searching possible decompositions of processes and searching sub locations when checking spatial modalities. This thesis takes the brute force search as naïve algorithm for searching possible decomposi-

tions of processes and searching sub locations and offers heuristics to reduce the search space.

### 3.2. Internal Representation of Specifications and Formulas

Internal representation of the information of the calculus and logic is an important decision which influences the search operations. In [6] state information is represented in terms of sets. In [5] calculus information and the logic information is represented as strings and algorithms are based on string operations. In this thesis all kind of information like ambient topology of calculus or the structure of the logic formulas are represented as graphs.

State information of an ambient calculus process consists of static properties and dynamic properties. Static properties of state are the ambients and their hierarchical organization, i.e. ambient topology. The dynamic properties of the state, the potential of process to evolve, are the capabilities and their dependencies to each other. Static and dynamic properties of an ambient calculus specification are kept in separate data structures.

**Definiton 3.2.1** *Ambient Topology,* $G_{AT} = (N_{AT}, A_{AT})$*, is acyclic digraph where nodes denoting ambients of specification and arcs denoting parent-child relation among ambients.*

- $N_{AT}$ *is a set of the nodes*
- *nodes are labeled with elements of* $\Lambda$
- $A_{AT}$ *is a set of arcs*
- *an arc* $a = \{x, y \mid x, y \in N_{AT}\}$
- *indegree of nodes are 1*
- *outdegree of nodes can be any positive integer*

**Definiton 3.2.2** *Capability Tree,* $G_{CT} = (N_{CT}, A_{CT})$*, is a acyclic digraph where nodes*

*denoting capabilities and arcs denoting priority relation among capabilities. Nodes contain the information about which ambient the capability is attached and which ambient the capability effects.*

- $N_{CT}$ *is the set of the nodes*
- $A_{CT}$ *is a set of arcs*
- *an arc* $a = \{x, y \mid x,\ y\ \in\ N_{CT}\}$
- *indegree of nodes are 1*
- *outdegree of nodes can be any positive integer*

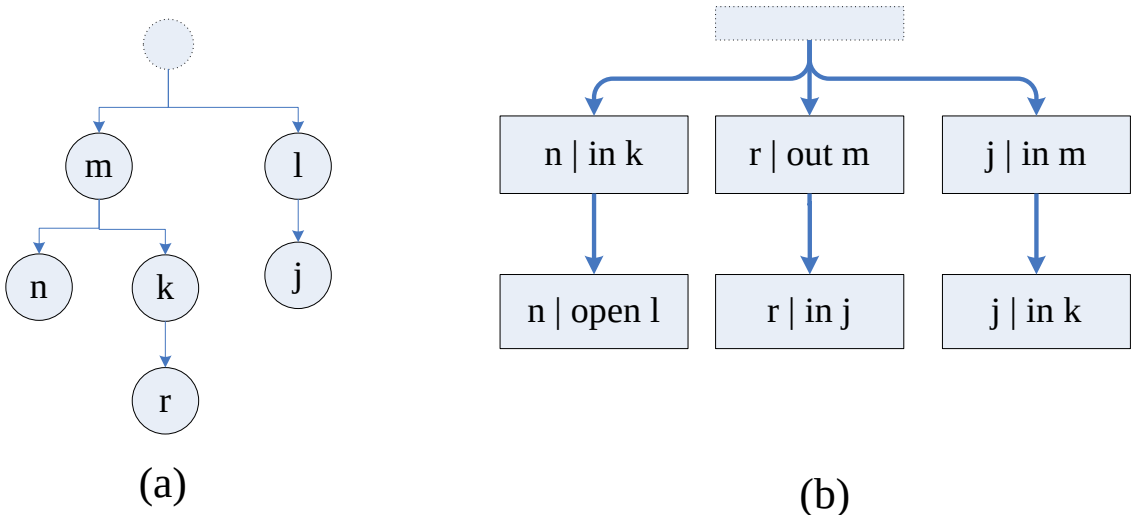


Figure 3.3. Internal representation of state information. Graph at (a) is ambient topology of state where graph at (b) is capability tree.

Graphs representing formulas are more complex than the others. They are acyclic digraph where node denotes connectives and locations and arcs denote the operator operand relation. There are multiple types of nodes and arcs at formula graphs because of the different structure of the ambient logic connectives.

**Definiton 3.2.3** *Formula,* $G_F = (N_F, A_F)$*, is acyclic digraph where*

- $N_F\ =\ (N_L\ \cup\ N_{Binary}\ \cup\ N_{Unary}\ \cup\ N_{PC})$.

- $N_L$ *is the set of nodes representing ambients*
- *elements of* $N_L$ *are labeled with elements of* $\Lambda$.
- $N_{Unary}$ *is the set of nodes representing unary connectives* ($\neg$, $\diamond$, $\Diamond$) *at formulas.*
- $N_{Binary}$ *is the set of nodes representing binary connectives,* ($\vee$) *at formulas.*
- $N_{PC}$ *is the set of nodes representing parallel compositions at formulas.*
- $A_F = (A_{PC} \cup A_{Binary} \cup A_{Unary})$
- *an arc* $a \in A_{PC} = x, y | x \in N_{PC},\ y \in (N_L \cup N_{Binary} \cup N_{Unary})$. *Elements of* $A_{PC}$ *represents parallel compositions of connectives at ambient logic formulas.*
- *an arc* $a \in A_{Unary} = (x, y \mid x \in (N_L \cup N_{Unary}), y \in N_{PC})$
- *an arc* $a \in A_{Binary} = (x, y | x \in N_{Binary},\ y \in N_{PC})$
- *indegree of nodes of* $N_F$ *is one.*
- *outdegree of nodes of* $N_{PC}$ *can be any positive integer*
- *outdegree of nodes of* $N_{Unary}$ *and* $N_L$ *are is one*
- *outdegree of nodes of* $N_{Binary}$ *is two.*
- *Elements of* $N_{PC}$ *can have a special attribute to represent the T construct of the logic.*
- *If T attribute of a* $N_{PC}$ *node is set to true this means the parallel composition of process that the* $N_{PC}$ *node stands for, includes T constant.*

Graphs representing ambient calculus processes denote parallel composition relation implicitly. The ambients placed under same node are considered as parallel constructs. So the parallel composition operator is not needed represented at graphs explicitly. But in graphs representing ambient logic formulas parallel composition must be represented explicitly because of binary connectives. To distinguish two distinct sub formulas connected by a binary operator, parallel composition connective of ambient logic must be denoted explicitly at graphs of ambient logic formulas.

### 3.3. Reduction Of Ambient Logic Formulas

To be able to use an existing temporal model checker, the ambient logic formulas have to be reduced to temporal logic formulas. Temporal operators of ambient logic

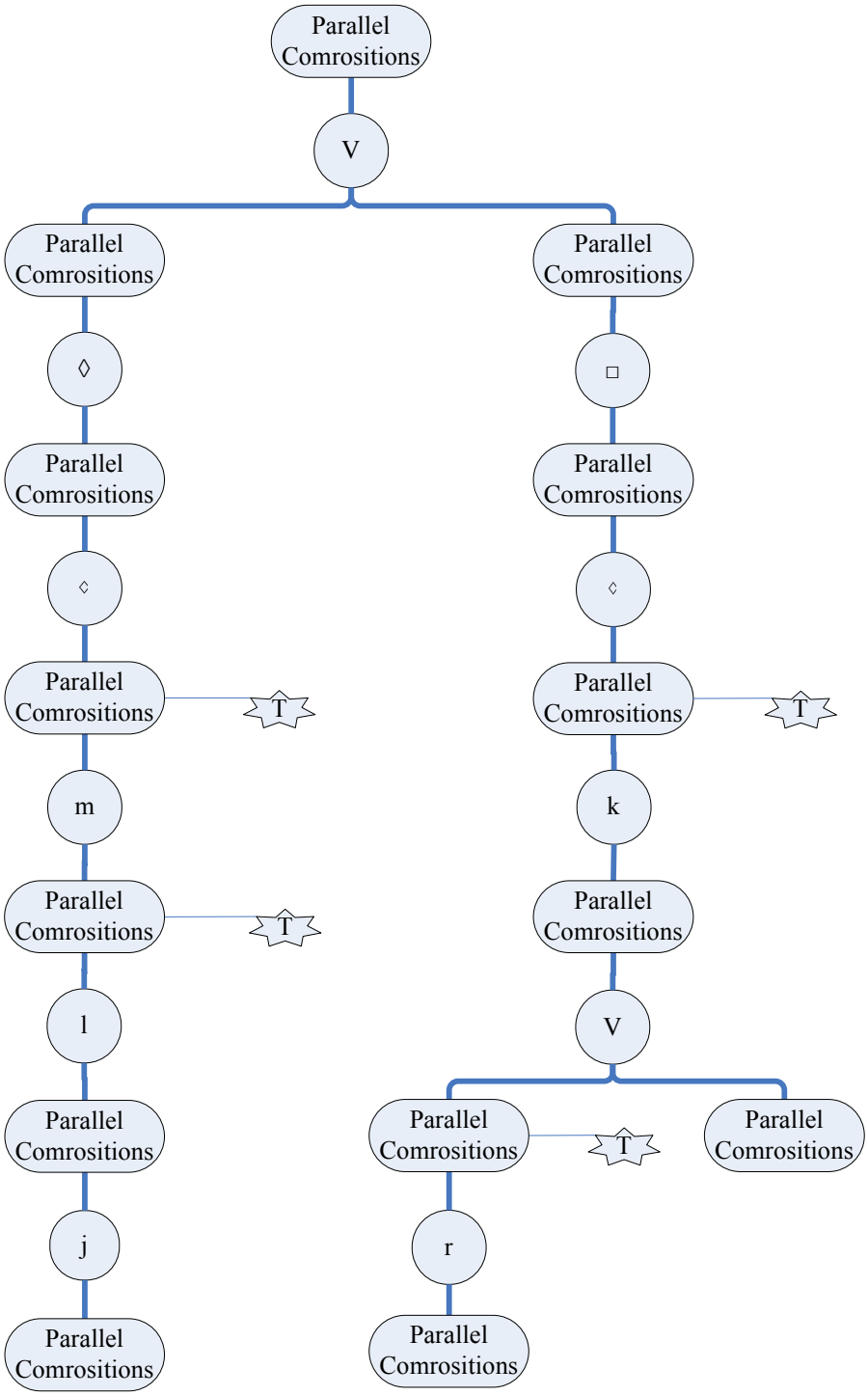


Figure 3.4. Internal representation of F = ◊ { ⋄ { m[l[j[]] | T] | T }} ∨ □ { ⋄ { k[{r[] | T } ∨ 0 ] | T}} as a graph.

are sometime (◊) and everytime (□) connectives. These operators are equivalent to EF and AG operators of CTL respectively. Only thing to reduce ambient logic formulas into a CTL equivalent form is replacing spatial modalities of the ambient logic formulas by atomic propositions.

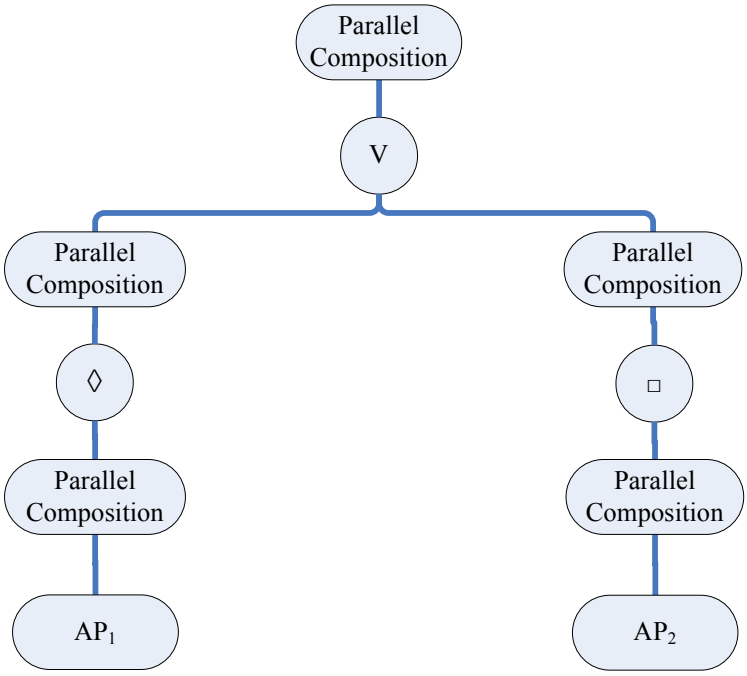


Figure 3.5. The CTL formula after decomposition of graph at Figure 3.4 into temporal and spatial subgraphs.

### 3.3.1. Nesting Problem of Temporal Operators

Using an existing tool for temporal model checking will reduce our work but it also has a drawback. While using a temporal model checker, some of the ambient logic formulas should be restricted.

- $n[\Diamond\mathscr{A}]$
- $n[\Box\mathscr{A}]$
- $\Diamond\mathscr{A}|\Diamond\mathscr{B}$
- $\Box\mathscr{A}|\Box\mathscr{B}$
- $\Box\Box\mathscr{A}$

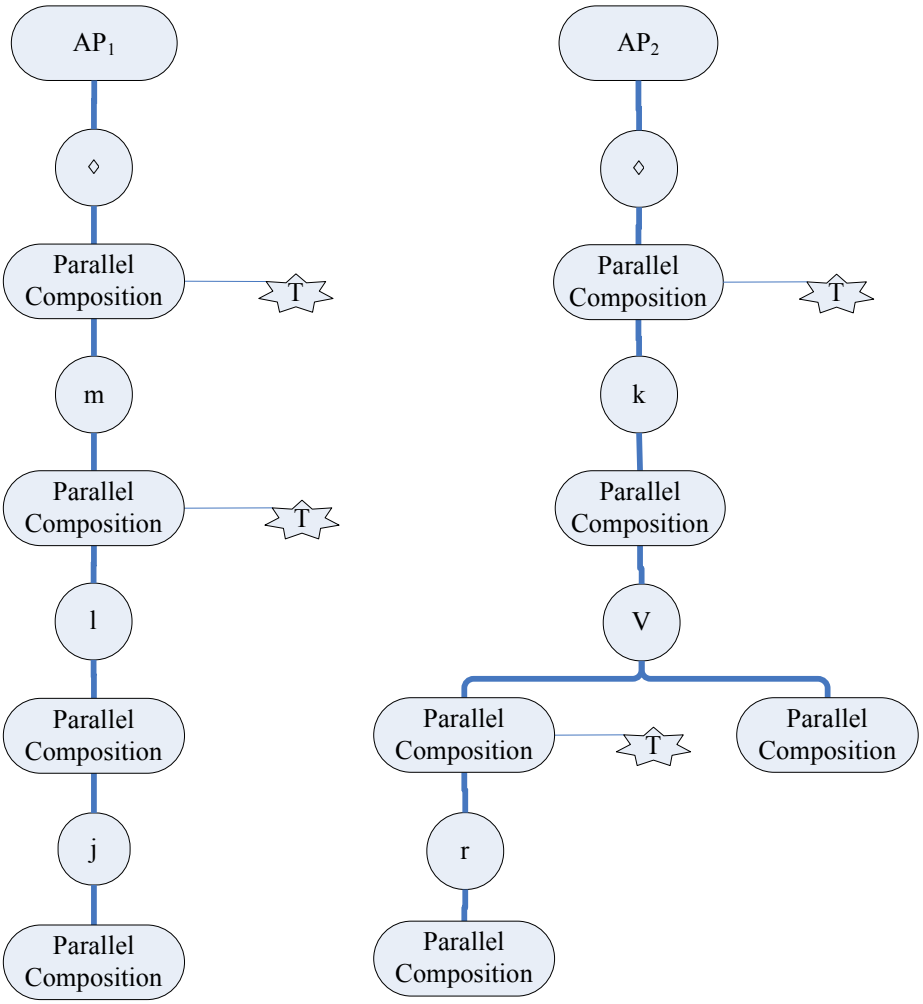


Figure 3.6. The spatial formulas after decomposition of graph at Figure 3.4 into temporal and spatial subgraphs.

- $\Box\Diamond\mathscr{A}$
- $\Diamond\,\Box\,\mathscr{A}$
- $\Diamond\Diamond\mathscr{A}$

At list above, some valid ambient logic formulas are listed. These formulas are not reducible to CTL formulas. They are not reducible because they have temporal operators at the sub-formulas which are operands of spatial operators. For example a formula like $\Diamond\ \mathscr{A}\ |\ \Diamond\ \mathscr{B}$ can't be converted to a CTL formula because temporal modalities are one nesting down from spatial modality "|". These formulas also don't have semantically equivalent derivatives which can be converted into CTL formula. In proposed methodology ambient logic formulas must be restricted as having temporal operators at only higher levels of spatial operators i.e. temporal operators can't exist in sub formulas connected by spatial operators. The restriction also brings an advantage. This restriction of ambient logic formulas provides the use of other CTL operators like $AF$ or $EG$ in a more straightforward way.

### 3.4. State Transition System Generation

In proposed model checking methodology the state transition system is generated from the initial model specification. State transition system is used at generation of Kripke Structure. The succeeding states are generated by executing the dynamic properties, capabilities. Because replication is not included in the proposed fragment, states are incapable of generation states which are equivalent to themselves or their former states as future states. As a result of this, state transition system can be represented by an acyclic digraph where nodes denote states and the edges denote attainability by one capability execution. Because there may be parallel capabilities that either of them can be executed first, the transitions at the state transition system are branching, not linear. The alternative execution orders of capabilities produce branching state transitions. Generating reachable states are researched in [5, 7, 17, 18]. Proposed thesis takes approach of [7] as starting point and offers a data structure which reduces calculation time.

When deciding the next capability to execute, there are some condition checks must be carried out. These conditions are the presence of the object ambient at right place and the availability of subject ambient. If the object ambient of the capability is not present at right place or it prefixed by a capability path, the capability cannot be executed. If the parent ambient of the subject ambient is prefixed by a capability path, the capability cannot be executed. In proposed thesis the presence of the object ambient at right place and the existence of prefixes are checked at every time the capability to be executed.

In proposed thesis a novel method (capability trees) to represent temporal behaviors of stated is offered which eliminates the time to check if the subject ambient is available. Capability paths are organized as an acyclic digraphs with respect to their interdependencies resulted by prefixing of parents (see Figure 3.3). This graph is built at parsing stage so no extra preprocessing is needed. Choosing the next capability to execute is started from the root of this graph. This method guarantees that the capabilities of the parent processes are executed before the capabilities of child processes. As a result this, no check for availability of the subject ambient is needed.

The proposed state transition system generation process is a recursive procedure that takes a capability graph and a state as inputs. In fact the capability graph is a part of the state information but in procedure it is explicitly mentioned to provide more evident sight of algorithm. Initial state information from the parser is passed the first call of the procedure. The proposed state transition system generator procedure calculates the child states of the given state and invokes itself with appropriate parameters for each child state to let them calculate their own child states. A state transition system will be resulted the end of this process. All states at this state transition system will be checked against spatial formulas when Kripke Structure generated.

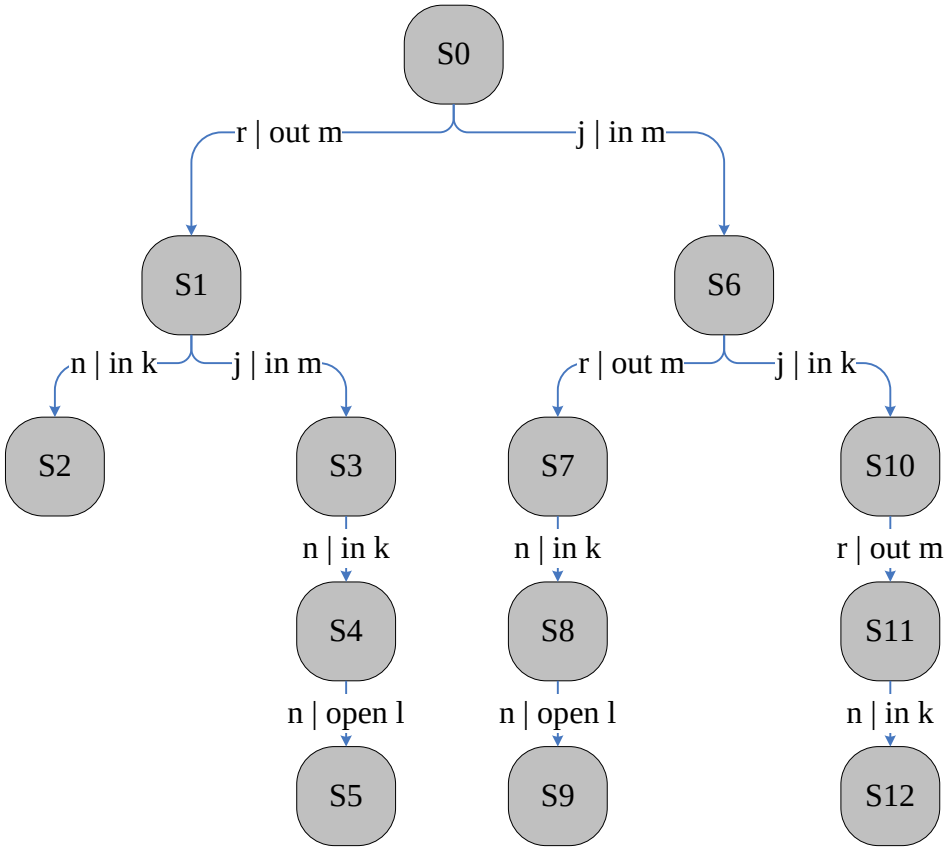


Figure 3.7. State transition system after all states are generated for P = { m[in k. open l.n[] | k[out m.in j.r[]]] | l[in m. in k. j[]] } . Labels on the edges shows the capability causes the transition.

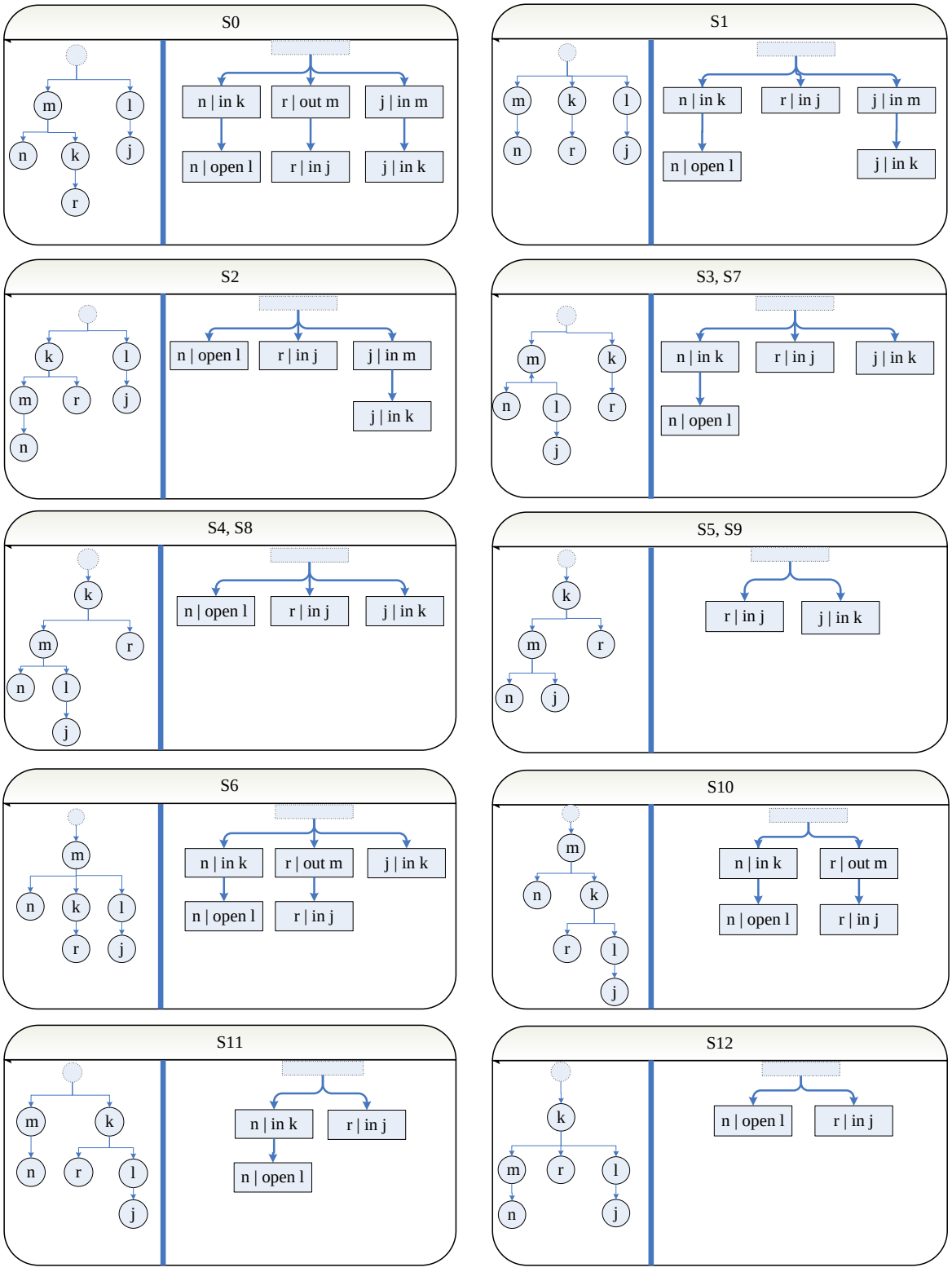


Figure 3.8. Details of states at Figure 3.7.

Table 3.1. State transition system generation algorithm

```
PROCEDURE stateSpaceGenerator
BEGIN
  input : GRAPH is capability tree, PARENT is state
  output : none

  CURRENT as capability node
  CHILD as state
  REPEAT
    CURRENT ← next top most node of GRAPH
    CHILD ← execute CURRENT over PARENT
    add CHILD to PARENT as successor state
    stateSpaceGenerator(PARENT - CURRENT, CHILD)
  UNTIL GRAPH is not empty OR all topmost nodes of GRAPH is explored
END
```

## 3.5. Checking Spatial Modalities of Ambient Logic

The basic needs for building a model checker for ambient calculus and ambient logic, are defined at [2] and section 2.4 as satisfaction rules. In the proposed methodology all the states generated must be checked against the spatial formulas. Ambient logic formulas are decomposed into a CTL formula and a set of spatial formulas by the formula reduction. Spatial formulas consist of spatial connectives of ambient logic and classical propositional connectives listed below.

- true
- void
- Parallel composition
- Negation
- Disjunction
- Location
- Somewhere

The data structures that spatial model checking takes as inputs are the ambient topology and the spatial formula graphs. The spatial model checking is used while Kripke Structures are being generated.

Matching of an ambient topology and a spatial formula is a recursive procedure in which ambient topology nodes are assigned to formula nodes. Matching process starts with assigning root of the ambient topology to the root of the spatial formula. Spatial formula nodes can forward the assigned ambient topology node to its children partially or completely in a recursive manner. Match process is successful when all nodes at ambient topology is matched a spatial formula node. Match processes at different type of spatial formula nodes are different. These different match process are introduced after auxiliary functions.

### 3.5.1. Auxiliary Functions

Auxiliary functions defined this section are the heuristics used at matching parallel composition and somewhere connectives. Former works try to match every alternative while searching a match for these connectives. In proposed algorithm the number of these trials are reduced by the help of auxiliary functions.

3.5.1.1. Wildcard Search. Some connectives at ambient logic match every kind of ambient topology. These connectives are used for matching ambients of ambient topology which are not expressed at formulas. The constant T of the logic matches any ambient topology assigned to it. Negation connective of the logic can be seen another kind of wildcard connective. Negations matches any ambient topology unless the sub formula of the negation matches this ambient topology. Another source of wildcard property is somewhere connectives. At definitions 2.4.8, 2.4.7 and 2.4.9 the parallel process of the parent ambient are neglected when searching sublocations. So if the sublocation search is obtained by applying $\downarrow$ one or more times somewhere connective gets wildcard property. Function *wildcard* is a recursive function used for determining if a node of formula graphs has wildcard property or not.

Table 3.2. Algorithm of *wildcard*

```
PROCEDURE wildcard
BEGIN
  input: NODE is a formula graph node
  output: boolean constant

  IF NODE is a location
    return false
  IF NODE is a disjunction
    return wildcard(node.first_child) ∨ wildcard(node.second_child)
  IF NODE is a somewhere
    return true
  IF NODE is a negation
    return true
  IF NODE is a parallel_composition
  BEGIN
    IF NODE has a T property
      return true
    FOR EACH child of NODE
    BEGIN
      IF wildcard(child) = true
        return true
    END
  END
    return false
END
```

3.5.1.2. Guessing Expected Ambients. It is not obvious to see which ambients are expected at sub formulas of disjunction and somewhere connectives. *guessExpectedAm-*

*bients* function is a recursive function, returns a set of expected ambient combinations for a formula graph node. The returned set includes all possible ambient combinations expected by children of that node. The returned value is a set instead of a single ambient combination.

Table 3.3. Algorithm of *guessExpectedAmbients*

```
PROCEDURE guessExpectedAmbients
BEGIN
  input : NODE is spatial formula node
  output: set of set of string

  IF NODE is a location
    return  NODE.name
  IF NODE is a disjunction
    return guessExpectedAmbients(node.first_child)
          ∪ guessExpectedAmbients(node.second_child)
  IF NODE is a somewhere
    return guessExpectedAmbients(node.child)
  IF NODE is a negation
    return
  IF NODE is a parallel_composition
    return cartesian products of the elements of
          returned values of guessExpectedAmbients for each child
END
```

3.5.1.3. Searching Sublocation. Function *findSublocation* is a recursive function used to find parent of an ambient at an ambient topology.

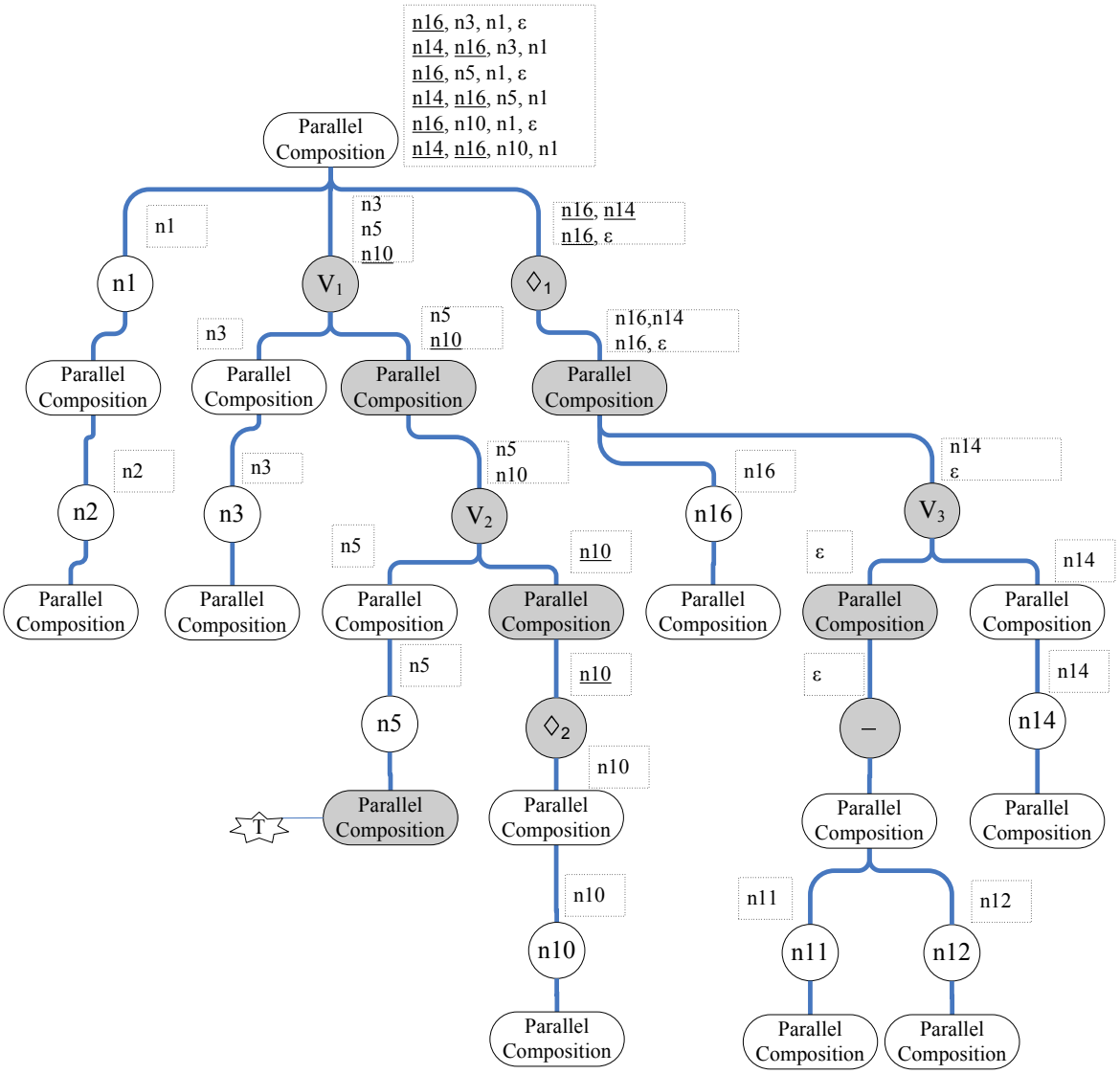


Figure 3.9. The evaluation of *wildcard* and *guessExpectedAmbients* functions for $F ::= n1[n2[]] \mid n3[] \ \vee \ n5[T] \ \vee \ \diamond n[10] \mid \diamond \{n16[] \mid \{n11[] \mid n12[]\} \ \Rightarrow \ n14[]\}$. Nodes having wildcard property are filled with grey. The set of the expected ambient compositions is shown in the boxes with dashed border line at the top of the related node.

Table 3.4. Algorithm of *findSublocation*

```
PROCEDURE findSublocation
BEGIN
  input : WANTED is string, ROOT is ambient topology node
  output: RESPONSE is ambient topology node

  FOREACH child of ROOT
  BEGIN
    IF child.name = WANTED
      return ROOT
  END
  FOREACH child of ROOT
  BEGIN
    RESPONSE ← findSublocation(WANTED,child)
    IF RESPONSE ≠ null
      return RESPONSE
  END
  return null
END
```

### 3.5.2. Matching Processes of Ambient Logic Connectives

In a match between an ambient topology and spatial formula graph, all nodes of ambient topology must be matched with a node of spatial formula graph. Some nodes of spatial formula graphs can forward the ambient topology nodes assigned to them to their children while others match assigned ambient topology nodes directly. The proposed spatial model check algorithm tries alternative assignments of given ambient topology nodes over given spatial formula graph with respect to definitions in section 2.4. The proposed spatial model checking algorithm is recursive where matching process starts from roots of graphs and continues level by level. If a suitable matching found at upper level then matching process continues to find matches for lower levels. The match process is regulated by the semantics of spatial formula graph node. The match process for each kind of spatial formula graph node is introduced below.

3.5.2.1. Locations. Matching process at nodes representing locations is defined according to definition 2.4.5. These nodes can be assign only one ambient topology node. If the ambient topology node has not the same name with the location node, match process for the location node fails. If the ambient topology node has the same name with the location node, location node assigns the children of ambient topology node to its parallel composition child. The result of the match is successful if the parallel composition child of location node succeeds to find a match between its children and the children of the assigned ambient topology node.

3.5.2.2. Negation. Matching process at the nodes representing negation connective is defined according to definition 2.4.2. These nodes can be assigned a collection of ambient topology nodes. Negation assigns the whole set of the assigned ambient topology nodes directly to its child parallel composition node. If the parallel composition child of the negation node succeeds to find a match, match process for negation node fails. If no match between ambient topology nodes and children of the parallel composition is found, match process for negation node is successful.

3.5.2.3. Disjunction. Matching process at the nodes representing disjunction connective is defined according to definition 2.4.3. These nodes can be assigned a collection of ambient topology nodes. As stated in section 3.2, nodes of type disjunction have two parallel composition children. Disjunction assigns the whole set of the ambient topology nodes directly to its child parallel composition nodes. If at least one of the parallel composition nodes succeeds to find to a match, match process for disjunction node is successful.

3.5.2.4. Somewhere. Matching process at the nodes representing somewhere connective is defined according to definition 2.4.9. These nodes can be assigned a collection of ambient topology nodes. Somewhere node can start matching its parallel composition node from any level of assigned ambient topology node collection. Match process at somewhere nodes has to try all possible levels of the ambient topology nodes until it find a successful match. This thesis proposes a heuristic to fasten the matching of somewhere node. Searching a single node in the ambient topology is cheaper than trying a full match in every level. By giving name of a location node child of parallel composition child of somewhere node and the collection of ambient topology nodes assigned to somewhere node, *findSublocation* finds level of ambient topology which somewhere must start its match process. This technique eliminates the searches of the levels which have not any possibility to match.

3.5.2.5. Parallel Composition. Matching process at the nodes representing parallel composition connective is defined according to definition 2.4.6. These nodes can be assigned a collection of ambient nodes. Match process at parallel composition decomposes assigned ambient topology node collection into subsets which will be forwarded to children of the parallel composition node. While there are exponentially many alternative decompositions, in this thesis the number of these alternatives is tried to be reduced by the help of *guessExpectedAmbients* and *wildcard* functions.

These decompositions are made in two phase. In first phase the expected ambient topology nodes are assigned to child nodes of parallel composition node. By the

help of *guessExpectedAmbients* function the set of expected ambient topology nodes, called guess set, are found for each child of parallel composition node. Then the every expected ambient topology node, in the collection assigned to parallel composition node, is forwarded to a child of parallel composition with a guess set including name of the expected ambient topology node. At the end of first phase all expected ambient topology nodes of the the assigned collection is assigned the related child of the parallel composition node. But there will be still unassigned ambient topology nodes at the collection.

Because there are ambient topology nodes which are not expected from any child node, these nodes must be assigned to one of children of parallel composition with wildcard property or neglected if the parallel composition node has T property. In second phase children of the parallel composition is checked if they have wildcard property or not by the help of *wildcard* function. After determining the set of children with wildcard property, unexpected nodes of the ambient topology collection are assigned to suitable elements of this set. If the parallel composition node does not have T property or a child with wildcard property to assign unexpected nodes, match process terminates unsuccessfully for this parallel composition node.

After assigning all ambient topology nodes at the assigned collection to children of the parallel composition new matching processes are started for every child of the parallel composition node. If one of the children fails, the match process for the parallel composition node tries to find another decomposition of the assigned collection of ambient topology nodes. Match process succeeds if match processes of all children succeeds for a decomposition of the assigned collection of ambient topology nodes. In Figure 3.10 matching of an ambient topology and a spatial graph for a successful match is shown.

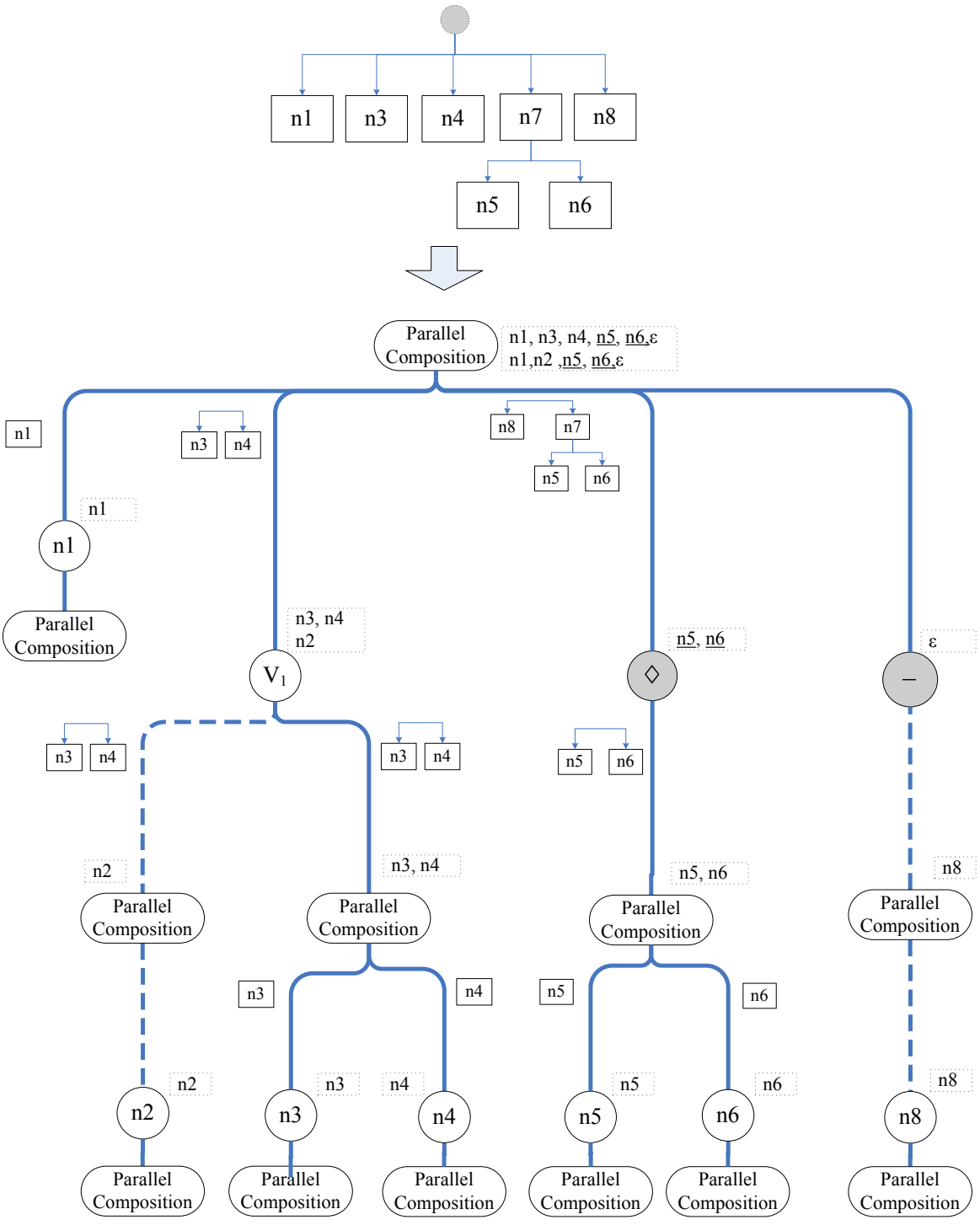


Figure 3.10. A match example for process P = n1[] | n3[] | n4[] | n7[ n5[] | n6[]] | n8[] and formula F = $n1[] \mid \{n2[] \vee \{n3[] \mid n4[]\}\} \mid \diamond\{n5[] \mid n6[]\} \mid \neg n8[]$. Graphs consisting of rectangle nodes are ambient topologies assigned to spatial formula graph nodes. Graphs consisting of circle nodes is spatial formula graph.

### 3.6. Kripke Structure Generation

The data structure built at state transition system generation is used to obtain Kripke Structure. This state transition data structure provides sets S, S0 and relation R of a Kripke Structure. The elements of set of atmoic propositions come from formula reduction. In formula reduction spatial formulas are replaced with atomic propositions. The function L is generated by applying spatial model checking for each state in state transition data structure against each spatial formula. Kripke Structure is obtained by attaching the values, coming from spatial model checking, into the state transition system graph.

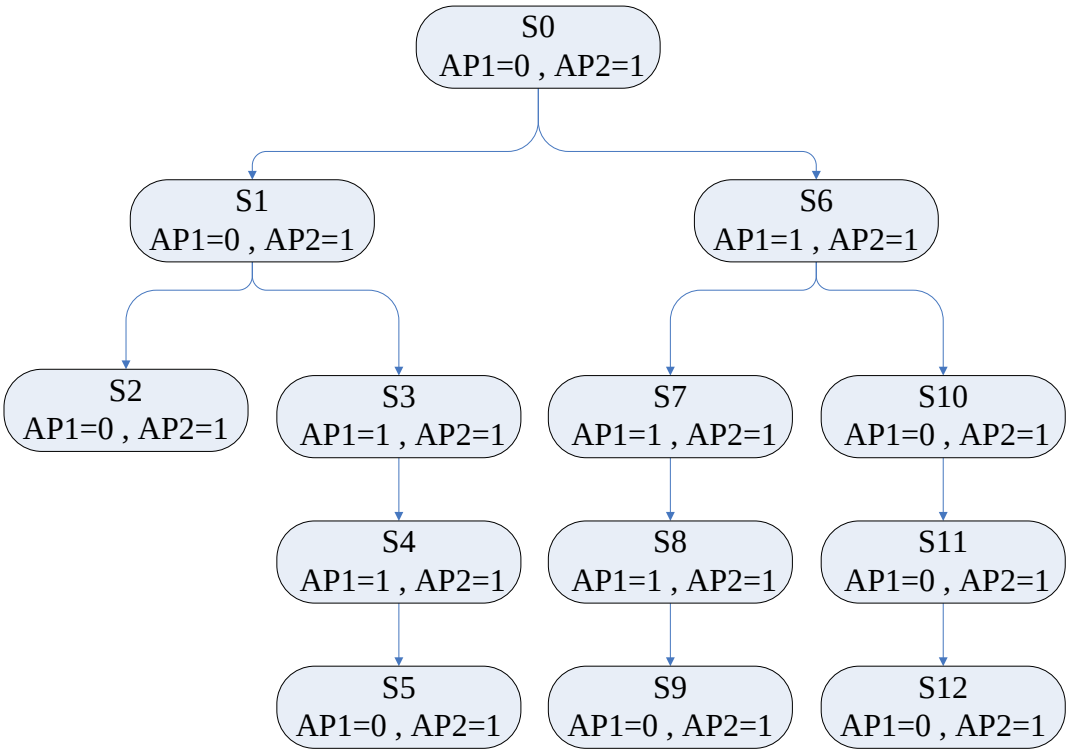


Figure 3.11. Kripke Structure obtained by checking spatial modalities of states of state transition system at Figure 3.7 with respect to spatial formulas at Figure 3.6.

## 3.7. NuSMV Code Generation

NuSMV is a symbolic model checker originated from the CMU SMV, the original BDD-based model checker developed at CMU. The NuSMV project aims at the development of a state-of-the-art symbolic model checker. It has a well structured and flexible architecture which reduces the effort needed to modify and extend NuSMV. It is open to use and modify due to it's an OpenSource license1. NuSMV is very robust, portable, efficient, and easy to understand by people other than the developers [16]. NuSMV allows for the representation of synchronous and asynchronous finite state systems, and for the analysis of specifications expressed in Computation Tree Logic (CTL) and Linear Temporal Logic (LTL), using BDD-based and SAT-based model checking techniques [16]. Heuristics are available for achieving efficiency and partially controlling the state explosion. Proposed model checking mechanism uses NuSMV as temporal model checker. Former processes of model checking mechanism provide CTL formula and a Kripke Structure. Rest of the work is generating a NuSMV code which semantically equivalent to the Kripke Structure and Formula.

In NuSMV states are defined over states variables. In proposed thesis two kind of NuSMV variables are used. A variable which is type of symbolic constant enumeration, is used for specifying states. Its name is *state*. The possible values of this variable are the identifiers of the states in Kripke Structure. The other kind of the state variables used in NuSMV code generation is boolean variables for representing atomic propositions. For each atomic proposition produced at formula reduction, a variable exists in NuSMV code which is type of boolean.

Table 3.5. Variable declaration example for NuSMV

```
VAR
   AP1 : boolean;
   AP2 : boolean;
   state : {S0,S1,S2};
```

NuSMV codes have an assignment section for determining the values of the vari-

ables. There are three types of assignments, assigning initial values, assigning relative values and assigning next values of variables. The label of the states of Kripke Structure is encoded at this section as relative assignment. Values of the atomic proposition variables are assigned according to value of variable *state*. The only initial assignment is done for variable *state* which determines the start of the state search. Transitions at the Kripke Structure are defined over assignments of the next values of variable *state*. At Table 3.6 assignments for NuSMV is illustrated.

Table 3.6. An example of assignments of state variables at NuSMV

```
ASSIGN
  init(state) := S0;
  AP1 :=
    case
      state = S0 : 0;
      state = S1 : 0;
      state = S2 : 0;
    esac;
  AP2 :=
    case
      state = S0 : 1;
      state = S1 : 1;
      state = S2 : 1;
    esac;
  next(state) :=
    case
      state = S0 : S1, S2;
      1: state;
    esac;
```

CTL formulas provided by formula reduction must be converted into input Language of NuSMV. This conversion is straightforward. A string is generated according to CTL formula graph provided by formula reduction where sometime connective is

represented as EF and everytime connective is represented as AG. The atomic propositions are reflected into the string with their names. At Table 3.7 the generated NuSMV code for Kripke Structure at Figure 3.11 and formula at Figure 3.4 is represented.

Table 3.7. A generated NuSMV code example

```
MODULE main
VAR
  AP1 : boolean;
  AP2 : boolean;
  state : S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12;
ASSIGN
  init(state) := S0;
  AP1 :=
    case
      state = S0 : FALSE;
      state = S1 : FALSE;
      state = S2 : FALSE;
      state = S3 : TRUE;
      state = S4 : TRUE;
      state = S5 : FALSE;
      state = S6 : TRUE;
      state = S7 : TRUE;
      state = S8 : TRUE;
      state = S9 : FALSE;
      state = S10 : FALSE;
      state = S11 : FALSE;
      state = S12 : FALSE;
    esac;
  AP2 :=
    case
      state = S0 : TRUE;
      state = S1 : TRUE;
      state = S2 : TRUE;
      state = S3 : TRUE;
      state = S4 : TRUE;
      state = S5 : TRUE;
      state = S6 : TRUE;
      state = S7 : TRUE;
      state = S8 : TRUE;
      state = S9 : TRUE;
      state = S10 : TRUE;
      state = S11 : TRUE;
      state = S12 : TRUE;
    esac;
  next(state) :=
    case
      state = S0 : S1, S6;
      state = S1 : S2, S3;
      state = S3 : S4;
      state = S4 : S5;
      state = S6 : S7, S10;
      state = S7 : S8;
      state = S8 : S9;
      state = S10 : S11;
      state = S11 : S12;
      1: state;
    esac;

  SPEC EF(AP1) — AG (AP2)
```

# 4. COMPLEXITY ANALYSIS

In proposed methodology, model checking process is decomposed into two independent processes as generating state transition system and checking spatial modalities for states. Time and space complexities for these processes are examined separately.

## 4.1. Time Complexity

Time complexity of generation states transition system is dependent to number of capabilities. Each capability causes a future state. The functionality of the capabilities tightly dependent to each other. They can operate on same ambients so execution of a capability can disable other capabilities operating on the same ambient. For each different execution order of capabilities operating on same ambient, there are different number of applicable capabilities and future states. Another dependency between capabilities are priority relations between them. The organization of the ambients which capabilities attached on and sequencing of capabilities bring priority relation between capabilities. A capability can not be executed before the other capabilities with higher priority.

Because of the dependency facts mentioned above it is very difficult to guess the number of states in state transition system from the number of capabilities. In the worst case all capabilities are independent. Independent means all capabilities are at same priority and all off them operates different ambients. In worst case execution order of the capabilities does not alter the set of the applicable capabilities. If the capability number is $n$, there are $n$ capabilities to execute first, $n-1$ to execute second, and so on until the last capability will be executed at $n_{th}$ order. The time cost of generating state transition system in worst case is

$$\sum_{k=0}^{n} \frac{n!}{k!} \tag{4.1}$$

But in general dependent capabilities are more common at ambient calculus specifi-

cations. The alternative model checking algorithms have the same complexity with proposed algorithm for generating future states. There is not any work on algorithms reducing the number of states at state transition system for ambient calculus.

Time complexity of checking spatial modalities are dependent to the number of the connectives of the spatial formulas. But it can be seen from definitions at section 2.4 that the match processes for satisfying different types of spatial connectives are not similar. Satisfaction searches for the connectives location, disjunction, negation and inactivity are done with two comparison at most while satisfaction search for parallel compositions and somewhere connection tries arbitrary number of alternatives to find a match. It can be said that the search for parallel compositions and somewhere connectives are dominant at time consumption of the spatial model check process.

### 4.1.1. Time Cost of The Match Process at Parallel Composition Connective

The match process for parallel composition connective consist of two phase. First one for assigning expected nodes and second one for assigning unexpected nodes. Let

- $l$ is the number of location in the parallel composition,
- $d_w$ is the number of disjunctions which have wildcard property in the parallel composition,
- $d$ is the number of disjunctions which have not wildcard property in the parallel composition,
- $not$ is the number of negations in the parallel composition,
- $sw_w$ is the number of somewhere connectives which have wildcard property in the parallel composition
- $sw$ is the number of somewhere connectives which have not wildcard property in a parallel composition,
- $G$ is the cost of calculating *guessExpectedAmbients* function
- $W$ is the cost of calculating *wildcard* function
- $a_e$ is the number of topmost ambients of the ambient topology, are expected in guess functions,

- $a_{ne}$ is the number of topmost ambients of the ambient topology, are not expected in guess functions,

The cost of first phase is dependent to disjunction connectives because each disjunction causes two different expected node combinations. The number of different assignments of expected nodes is

$$2^{d_w+d} \tag{4.2}$$

Unexpected nodes of ambient topology can be assigned only the formula nodes which has wildcard property. The number of different assignments of unexpected nodes are

$$a_{ne}^{(sw_w+not+d_w)} \tag{4.3}$$

The overall time cost of the match process is product of 4.2 and 4.3 plus costs of the auxiliary functions

$$2^{d_w+d} \times a_{ne}^{(sw_w+not+d_w)} + G + W \tag{4.4}$$

Time cost of brute force search for trying all decomposition alternatives of ambient topology nodes consisting $a_e$ and $a_{ne}$ for the parallel composition is

$$(a_{ne} + a_e)^{(sw_w+sw+l+not+d+d_h)} \tag{4.5}$$

Time cost of *guessExpectedAmbients* and *wildcard* function are linear with the number of connectives at spatial formula because they are called for each connective at formula in a recursive manner.

### 4.1.2. Time Cost of The Match Process at Somewhere Connective

Time cost of finding a matching for somewhere connective is the sum of cost of *findSublocation* and cost of the match process for the parallel composition connective

child of the somewhere connective. Let

- $PC$ is the cost of match process of parallel composition child of somewhere connective,
- $F$ the cost of *findSublocation* function,
- $a$ is the number of the ambients of the ambient topology,

The overall time cost of the match process for somewhere connective is

$$F + PC \tag{4.6}$$

Time complexity of brute force search for finding a matching for somewhere connective is

$$a \times PC \tag{4.7}$$

Time cost of *findSublocation* function is linear with $a$ because it only looks for an ambient with a specific name in an ambient topology. Time complexity of match process of parallel composition is showed at 4.4.

## 4.2. Space Complexity

In state transition systems, states are generated due to capability executions. Proposed algorithm builds state transition system with depth first manner. After calculation of its successor states, a state can be discarded from memory. The depth of the state transition system can be the number of capabilities at most. So the space complexity of the space generation is $O(n)$ where the n is number of capabilities.

Only one copy of the formulas is used at whole process of checking spatial modalities. The space needed for this process is the size of the formula. Size of the formula is dependent of the connectives at formula. So the space complexity of the formula is $O(n)$ where n is the number of the connectives at formula.

# 5. CASE STUDIES

In the scope of this thesis the proposed methodology for model checking of ambient calculus against ambient logic is implemented with Java. In this section the implementation is tested with case studies from the [1]. Test are applied on the cluster at cluster.tam.boun.edu.tr. The node of cluster which test is run on, has 8 x 2.93 GHz CPUs, 9.76 GB memory.

These case studies include three ambient calculus specifications and two formulas. Each ambient calculus specification models a multi domain network where domains, host, user, and files are modeled as ambients. Formulas represent different properties of these models. In tables at this section unit of the time measurements is second and unit of the memory measurements is kilobyte.

Table 5.1. Properties of specifications

| Specification | Ambient Number | Capability Number | Number of States of State Transition System |
|---|---|---|---|
| Spec1 | 16 | 32 | 560 |
| Spec2 | 16 | 39 | 33123 |
| Spec3 | 16 | 37 | 628527 |

Definitions of the specifications are given in Appendix A. All of the specifications consist of same set of ambients where their starting ambient topology is same. The main difference between them is the type and the number of the capabilities. Because of this the future evolvements of the models vary dramatically. These three specifications are checked against two different formulas. Definitions of the formulas are given in Appendix B.

To measure the efficiency of the proposed algorithm, a variant of the algorithm is implemented as naïve algorithm. The naïve algorithm does not make use of auxiliary functions instead it checks spatial modalities exhaustively. All specifications are

Table 5.2. State transition system generation cost.

| Specifications | Heap Size | Time (second) | Memory (KB) |
|---|---|---|---|
| Spec1 | 1 GB | 1.039 | 246478 |
| | 8 MB | 1.234 | 6228 |
| Spec2 | 1 GB | 12.453 | 350504 |
| | 8 MB | 18.120 | 7734 |
| Spec3 | 1 GB | 154.897 | 362384 |
| | 8 MB | 241.592 | 7911 |

Table 5.3. Properties of formulas

| Formula | Branching factor | Depth |
|---|---|---|
| Formula1 | 1 | 4 |
| Formula2 | 2.6 | 3 |

Table 5.4. Performance results with 1 GB heapsize

| Specifications | | Formulas | | | |
|---|---|---|---|---|---|
| | | Formula1 | | Formula2 | |
| | | Time (second) | Memory (KB) | Time (second) | Memory (KB) |
| Spec1 | Proposed | 1.265 | 262208 | 1.520 | 262208 |
| | Naïve | 1.634 | 262208 | 1.829 | 262208 |
| Spec2 | Proposed | 15.474 | 350872 | 17.134 | 351080 |
| | Naïve | 15.334 | 350672 | 18.405 | 353392 |
| Spec3 | Proposed | 172.289 | 362304 | 199.8151 | 375352 |
| | Naïve | 171.812 | 364696 | 201.765 | 375160 |

Table 5.5. Performance results with 8 MB heap size

| Specifications | | Formulas | | | |
|---|---|---|---|---|---|
| | | Formula1 | | Formula2 | |
| | | Time (second) | Memory (KB) | Time (second) | Memory (KB) |
| Spec1 | Proposed | 1.440 | 6494 | 1.584 | 6427 |
| | Naïve | 1.786 | 6565 | 2.000 | 7643 |
| Spec2 | Proposed | 21.446 | 7808 | 21.503 | 7776 |
| | Naïve | 21.012 | 7764 | 21.965 | 7905 |
| Spec3 | Proposed | 293.428 | 7817 | 297.025 | 7833 |
| | Naïve | 285.322 | 7826 | 297.518 | 7897 |

Table 5.6. Performance results of NuSMV with generated code

| Specifications | Formulas | |
|---|---|---|
| | Formula1 | Formula2 |
| Spec1 | 0.126 second | 0.135 second |
| Spec2 | 463.918 seconds | 615.361 seconds |
| Spec3 | Segmentation fault | Segmentation fault |

checked against two formulas with both proposed algorithm and the naïve algorithm. Because memory management of java is complicated due to garbage collector, the model check process is run with two different heap size. In Table 5.4 and Table 5.5 the performance results for model check processes are shown. The Time consumption of NuSMV for processing generated code is shown at Table 5.6.

It can be said that state transition system generation outweighs the spatial model check for both time and space consumption. Another result from these case studies is that brute force check is better when formulas with lower branching factors is dealt. Proposed heuristics are beneficial when formulas with high branching factors is dealt. Proposed model checker is capable of handling specifications with 628527 state with memory under 8 MB. Time consumption of NuSMV gets as more significant as the state number increases. Size of the generated NuSMV code increases linear with state number. NuSMV cannot process the generated code for the specification with 628527 states; it terminates with segmentation fault. Memory consumption showed at Table 5.4 can not tell the exact relation between capability number and memory consumption but it can be said memory consumption does not grow exponentially while capability number grows linearly.

# 6. RELATED WORK

There are model checkers for logics including spatiality aspects based on calculi other than ambient calculus. One of them is The Spatial Logic Model Checker [22] of Vieira et al. The Spatial Logic Model Checker is a model checker providing the automatic verification systems expressed in the finite control fragment of the $\pi$-calculus which supports recursion. $\pi$-calculus is a high-level mathematical modeling language for concurrent systems. Mobility is a basic primitive in $\pi$ calculus too but it lacks locations. A spatial logic [23] is used in [22]. Spatial logic of [23] includes composition, local name restriction, and a primitive fresh name quantifier as spatial operations in addition to propositional and temporal operators.

Another model checker proposal for spatial modalities without notion of locations is [21]. Lafuente presents an approach for the verification of spatial properties with Spin in [21]. SPIN uses Promela to model the concurrent systems formally and linear-time temporal logic (LTL) to specify system properties. In [21] SPIN is extended in order to make it able to check spatial properties. They propose a model checking algorithm for the logic and propose how SPIN can be minimally extended to include the algorithm.

A model checker algorithm based on ambient calculus and ambient logic is proposed in [3] first by Cardelli and Gordon. Their algorithm is devised for replication-free ambient calculus and guarantee free ambient logic. Name restriction is also removed from the fragment of ambient calculus that they used. So the operators operate on restricted names are not available in their fragment of ambient logic. An ambient calculus model checker using ambient logic, depends on searches for reachable states and sub-locations to check the sometime and somewhere modalities. In [3] the straightforward definitions of the routines to accomplish these searches are omitted, and brute force search for calculating process decompositions for matching parallel formulas is offered.

In [10], Charatonik et al introduce a finite-control fragment of the ambient calculus. The replication-free fragment of ambient calculus can express processes able to make only a finite number of computation steps. Previous work in [3] ambient calculus is restricted to processes lacking infinite executions and name restriction. [10] proposes a model checking algorithm for ambient calculus which provides recursively-defined, possibly nonterminating processes first. The only short coming of recursive defined process is the restriction of the output capability. In the fragment of ambient calculus of [10] only names can be operands of output operation. In [10], a model checker algorithm is proposed for the finite control fragment of the ambient calculus against guarantee free ambient logic. They proved that the model checking problem based on finite control ambient calculus is decidable and PSPACE-complete. Like [3], [10] omits the straightforward definitions of the routines to accomplish the reachable state search and sub-locations search. In [10] brute force search for calculating process decompositions for matching parallel formulas is offered while stating that this task is PSPACE-complete.

Charatonik et. al. study the model checking problem for the ambient calculus with public names against the ambient logic in [5]. Their work settle the complexity bounds of the model checking problem for the different fragments of the ambient calculus against the ambient logic. They show that ambient calculus with replication makes model checking process undecidable due to infinite array of replicas of the processes. They also show that an ambient logic formula including guarantee operator needs infinite quantification over processes which makes model checking problem undecidable too. They proved that the problem is PSPACE-complete in the decidable case of the replication-free ambient calculus with public names and the guarantee-free ambient logic. They also assert that there are no interesting fragments with polynomial-time model checking algorithms for ambient logic. They present a new model checking algorithm to avoid the processes grow exponentially during their execution due to communication by devising a new representation of processes. Like [3] and [10], [5] omits the straightforward definitions of the routines to accomplish the reachable state search and sub-locations search.

An alternative way of performing model checking for ambient calculus is offered by Mardare et al [6]. They propose a methodology for model checking for biological systems which are described by using ambient calculus. They introduce their own logic to express the properties of processes described by ambient calculus. Their logic is derived from CTL but it supports a spatial operator for spatial modalities. Expressivity of their logic for spatial modalities is lower than ambient logic. Their model checking approach can be defined as taking an ambient process definition and a logic formula as input and build a NuSMV code from these inputs. The methodology offered in this thesis is strongly influenced by their work. The main difference with the proposed model checking methodology and [6] is that in the proposed model checking methodology ambient logic is used, which is more expressive for mobility and spatiality aspects than the logic used in [6]. Another difference is that capabilities are stored in a simple list in [6], while this thesis proposes storing capabilities in a priority tree. Storing capabilities in a priority tree eliminates extra checks for availability of the next capability to execute.

# 7. CONCLUSION

In this thesis a methodology for model checking of ambient calculus against ambient logic is proposed. The proposed methodology is based on separating analysis of temporal and spatial properties from each other. While analysis of temporal modalities are handle by the help of an existing tool, new algorithms for analyzing spatial modalities proposed in this thesis. The approach of separating analysis of temporal and spatial properties, is used at model checking of biological systems expressed with ambient calculus at [6]. The proposed thesis differs from [6] on the logic it used and the way of handling capabilities. While a CTL derived logic with a spatial operator is used in [6] ambient logic is used at proposed thesis. Because ambient logic is strictly based on ambient calculus it is more expressive for spatial properties of models.

The proposed separation of temporal and spatial analysis provides use of well developed temporal model checkers. But to be able to use an existing temporal model checker, ambient logic formulas have to nest spatial operators under temporal operators. This makes proposed algorithm less expressive than algorithms at [3, 5, 10].

The algorithms offered at [3, 5, 10] does not include implementation detail and straightforward definitions of some procedures. A main contribution of this thesis is providing an implementation of proposed algorithms which can be used at model checking of security policies defined for multi domain networks with ambient calculus and ambient logic.

In the scope of this thesis heuristics proposed for reducing time cost of analysis of spatial properties. The results of performance test showed that proposed heuristics for checking spatial modalities, fasten the model checking process for only formulas which's graphs has bigger branching factors. This improvement on time consumption of the model checking process is not very significant in compare to total time consumption. This is caused due to the way auxiliary functions are used. They are invoked at every separate spatial model check process for each state while only one invoke is enough for

all spatial model check processes.

Case studies and complexity analysis show that size of the state transition system is the most significant element at time and spatial cost of model checking. Number of states grows exponentially as capability number increase linearly. A partial order reduction might decrease the number of the states of the state transition system and reduce time consumption and size of generated NuSMV code. In [20] partial order reduction techniques are offered for *pi*-calculus but there is not a similar work for ambient calculus. Investigating partial order reduction techniques for ambient calculus is possible direction for future work.

There is no Turing-complete fragment of ambient calculus without recursion and replication [19]. There are works showing model checking for a fragment of ambient calculus with recursion is decidable [10]. Adding recursion to the fragment of ambient calculus, another direction for future work, will extend the class of the accepted models of the proposed model checking methodology.

# APPENDIX A: Ambient calculus specifications used at case studies

Spec1::= World[DomainA[Host1[User1[] |File1[data1[out File1.0|out Host1.0|out DomainA.0| in DomainB.0|in DomainC.0|in Host4.0 | in Host2.0|out Host2.0|in File3.0|in User2.0|out User2.0|in User4.0|out User4.0|in Host3.0]]]]|DomainB[Host3[File3[]]|Host2[User2[in File1.0|in File2.0|out File1.0|out File2.0|in File3.0|in File4.0|out File3.0|out File4.0]|File2[]]]|DomainC[Host4 [User4[out DomainC.0|in DomainB.0| in Host3.0| in File3.0|out File3.0|out Host3.0|out DomainB.0|in DomainC.in Host4. in File4.0]|File4[]]]]

Spec2::= World[DomainA[Host1[User1[out DomainA.0|in DomainA.0|in DomainB.0|out DomainB.0|in File1.0|out File1.0]|File1[data1[out File1.0|out Host1.0|out DomainA.0| in DomainB.0|in DomainC.0|in Host4.0 | in Host2.0|out Host2.0|in File3.0|in User1.0|in User2.0|out User2.0|in User4.0|out User4.0|in Host3.0]]]]|DomainB [Host3[File3[]]|Host2[User2[in File1.0|in File2.0|out File1.0|out File2.0|in File3.0|in File4.0|out File3.0|out File4.0]|File2[]]]|DomainC[Host4 [User4[out DomainC.0|in DomainB.0| in Host3.0| in File3.0|out File3.0|out Host3.0|out DomainB.0|in DomainC.in Host4. in File4.0]|File4[]]]]

Spec3::= World[DomainA[Host1[User1[out Host1.0|out DomainA.0|in DomainB.0]|File1[data1[out File1.0|out Host1.0|out DomainA.0| in DomainB.0|in DomainC.0|in Host4.0 | in Host2.0|out Host2.0|in File3.0|in User1.0|out User1.0|in User2.0|out User2.0|in User4.0|out User4.0|in Host3.0]]]]|DomainB[Host3[File3[]]|Host2[User2[in File1.0|in File2.0|out File1.0|out File2.0|in File3.0|in File4.0|out File3.0|out File4.0]|File2[]]]|DomainC[Host4 [User4[out DomainC.0|in DomainB.0| in Host3.0| in File3.0|out File3.0|out Host3.0|out DomainB.0|in DomainC.in Host4. in File4.0]|File4[]]]]

# APPENDIX B: Ambient logic formulas used at case studies

Formula1::= □ { ¬ ◇ { ⋄ { Host4[ ⋄ { data1[T] | T}] } } }

Formula2::= □ World [ DomainA [Host1 [T] | T] | DomainB[Host2 [T] | Host3 [T] | T] | DomainC[ Host4 [T] | T]] ∨ ◇{⋄{Host4[ ⋄{data1[T]}] | T}}

# REFERENCES


1. Unal, D. and M.U. Caglayan, "Security Policy Specification and Verification Framework for Multi-Domain Mobile Networks", to appear

2. Cardelli, L. and A.D. Gordon, "Ambient Logic", `http://lucacardelli.name/Papers/AmbientLogic.A4.pdf`, 2005

3. L. and A.D. Gordon, "Anytime, anywhere: Modal logics for mobile ambients", *In Proceedings POPL'00* pp. 365-377, 2000.

4. Cardelli, L. and A.D. Gordon, "Mobile Ambients", *Theoretical Computer Science*, vol. 240 pp. 177-213 2000.

5. Charatonik, W., Dal Zilio, S., Gordon, A.D., Mukhopadhyay, S. and J. Talbot, "Model checking mobile ambients", *Theoretical Computer Science*, vol.308, pp. 277-331, November 2000.

6. Mardare, R., Priami, C., Quaglia, P. and O. Vagin, "Model checking biological systems described using Ambient Calculus" *Computational Methods in Systems Biology*, pp. 85-103, 2005

7. Mardare, R. and C. Priami, "Computing the accessibility relation for ambient calculus. Technical report", `http://www.dit.unitn.it`, 2003.

8. Mardare, R. and C. Priami, "A propositional branching temporal logic for the ambient calculus, Technical Report", `http://www.dit.unitn.it`, 2003.

9. Charatonik, W., Dal-Zilio, S., Gordon, A.D., Mukhopadhyay, S. and J. Talbot, "The Complexity of Model Checking Mobile Ambients", *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures*, pp. 152-167, April 2001

10. Charatonik, W., Gordon, A.D. and J. Talbot, "Finite-Control Mobile Ambients", *Proceedings of the 11th European Symposium on Programming Languages and Systems*, pp. 295-313, April 2002

11. Charatonik, W. and J. Talbot, "The Decidability of Model Checking Mobile Ambients", *Proceedings of the 15th International Workshop on Computer Science Logic*, pp. 339-354, September 2001

12. G.J. Holzmann, "Software Model Checking", *NATO Summer School*, vol. 180, pp. 309-355, August 2000.

13. Onem, E., "Formal Security Analysis of a Secure On-Demand Routing Protocol for Ad Hoc Networks Using Model Checking", *M.S. Thesis, Computer Engineering, Bogazici University*, 2007.

14. Hirschkoff, D., Lozes, E. and D. Sangiorgi. "Separability, Expressiveness, and Decidability in the Ambient Logic", *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pp .423-432, July 2002

15. Hirschkoff, D., Lozes, E. and D. Sangiorgi. "On the expressiveness of the Ambient Logic" *Logical Methods in Computer Science* vol. 2, 2006.

16. "NuSMV: A new symbolic model checker", `http://nusmv.irst.itc.it`, 2006

17. Hansen, R.R., Jensen, J.G., Nielson, F. and H.R. Nielson, "Abstract Interpretation of Mobile Ambients", *In Proc. Static Analysis Symposium SAS'99*, vol. 1694 pp. 134-148, 1999.

18. C. Braghin, A. Cortesi, S. Filippone, R. Focardi, F. L. Luccio and C. Piazza. "A Tool for Boundary Ambients Nesting ANAlysis", *Electronic Notes in Theoretical Computer Science*, vol. 99, pp. 319-337, August 2004.

19. Maffeis,S., Phillips,I., "On the computational strength of pure ambient calculi", *Electronic Notes in Theoretical Computer Science*, vol. 96, pp. 29-49, , 2004.

20. Affeldt, R. and N. Kobayashi, "Partial Order Reduction for Verification of Spatial Properties of Pi-Calculus Processes", *Electronic Notes in Theoretical Computer Science*, vol. 128, pp. 151-168. April 2005

21. A. L. Lafuente, "Towards Model Checking Spatial Properties with SPIN", *Lecture Notes in Computer Science*, vol. 4595, pp. 223-242. 2007

22. Vieira H. and L. Caires, "The Spatial Logic Model Checker User's Manual", `http://www-ctp.di.fct.unl.pt/SLMC/manual.ps`, 2005.

23. Caires L. and L. Cardelli. "A Spatial Logic for Concurrency(part I)", *Information and Computation* vol. 186(2), pp. 194-235,2003.