工學博士學位論文

자율주행 차량의 교차로 트래픽 제어를 위한 그룹 상호배제 알고리즘

Group Mutual Exclusion Algorithm for Intersection Traffic Control of Autonomous Vehicle

忠 北 大 學 校 大 學 院

컴퓨터工學科 컴퓨터공학專攻

金 榮 穆

2017 年 02 月

工學博士學位論文

자율주행 차량의 교차로 트래픽 제어를 위한 그룹 상호배제 알고리즘

Group Mutual Exclusion Algorithm for Intersection Traffic control of Autonomous Vehicle

指導敎授 朴 聖 勳

컴퓨터工學科 컴퓨터공학專攻

金 榮 穆

이 論文을 工學博士學位 論文으로 提出함

2017 年 02 月

本 論文을 金榮穆의 工學博士學位論文으로 認定함

審 査 委 員 長 _____ 印

審 査 副 委 員 長 _____ 印

審 査 委 員 _____ 印

審 査 委 員 _____ 印

審 査 委 員 _____ 印

忠 北 大 學 校 大 學 院

2016 年 11 月

차 례

Group Mutual Exclusion Algorithm for Intersection Traffic Control of Autonomous Vehicle*

Kim, Yeong Mok

Department of Computer Engineering
Graduate School, Chungbuk National University
Cheongju, Korea
Supervised by Professor Park, Sung Hoon

Abstract

Intersection Traffic Control(ITC) attracted extensive attention with the increase of traffic accidents and congestion that causes a huge social and economic losses. Also the topic on ITC has been widely studied by research communities. These studies fall into two categories: traffic signal scheduling and trajectory maneuver based on Intersection Central Unit(ICU). The optimization of signal scheduling needs a lot of computations and the ICU based system is not flexible and not cost efficient. To solve this problem the distributed ITC system for autonomous vehicle should be the necessary requirement. However to our knowledge there is almost no work that has been devoted to

---

* A thesis for the degree of Doctor in February 2017

distributed ITC for autonomous vehicle.

This paper proposes a token-based group mutual algorithm for ITC of autonomous vehicle. We use a token as a privilege to pass CZ(Cross Zone) and use two kinds of tokens: *primay token* as a privilege and *sub token* as a auxiliary means to increase the performance of vehicles flow. How to transfer the token among vehicles in a dynamic system that a vehicle can not stay in CZ after it finish passing is the core of our algorithm. We design the circulation of the token efficiently among vehicles in a same group. Also we apply IL-chain concept to improve the flow of vehicles. IL-chain is a cluster that the captain of IL makes it by sending messages to the followers. The captain of a chain enters and exits CZ with its chain members and all vehicles linked in a chain move CZ in a group.

The proposed algorithm can handle quite a big traffic volume well with decreased message complexity. It outperforms the reference algorithms in message complexity and presents better result in system throughput than traffic signal system. This paper also will vitalize the researches about the distributed ITC of autonomous vehicle.

그 림 차 례

표 차 례

약 어

AV	: Autonomous Vehicle
AIM	: Autonomous Intersection Management
CG	: Crossing Zone Grid
CL	: Crossing Zone Logical Lane
CSG	: Compatible Stream Group
CZ	: Crossing Zone
FSA	: Finite State Automata
FIFO	: First-In-First-Out
GME	: Group Mutual Exclusion
GPS	: Global Positioning System
ICU	: Intersection Control Unit
Ig	: Input Lane Grid
IL	: Input Lane
LS	: Lane Set
QZ	: Queuing Zone
VTokenIC	: Group Mutual Exclusion Algorithm for Intersection Trafic Control of Autonomous Vehicle

Ⅰ. 서 론

최근 컴퓨터 테크놀로지의 발전과 유비쿼터스 시대가 도래함에 따라 자율주행 차량의 운행이 가능하게 되었다. 이러한 자율주행 차량의 출현과 더불어 지속적으로 증가하는 교통사고와 도로혼잡의 해결을 위하여 자율적 교차로 통과 시스템에 대한 관심이 커지고 있다. 이에 따라 자율 주행 차량의 교차로에서의 교통 제어 시스템에 대한 연구도 활발히 진행되고 있다. 이러한 자율주행 차량의 교차로에서의 교통 제어와 관련된 연구 동향을 살펴보면 다음과 같다.

하나는, 교통신호 통제 연구로서 컴퓨터의 인공 지능을 활용한 교통신호 시스템 개선이며, 다른 하나는 교통 신호등 없이 교차로를 자동으로 통제하는 차량 궤적관리 연구이다. 그러나 기하급수적으로 증가하는 교통의 혼잡과 차량의 역동성 아래에서 최적의 신호체계 및 차량궤적을 산출하여 통제하는 것은 어려운 일이다. 또한, 이들은 중앙 통제장치에 의하여 운영이 됨에 따라, 다음과 같은 문제점을 지니고 있다. 중앙통제 장치는 추가적인 장비와 설치 작업이 필요하므로 비용이 수반되고, 또한 한번 설치되면 설계가 유동적이지 않아 이의 변경 및 개선이 쉽지 않다. 가장 큰 문제점은 중앙통제장치 한 곳이 고장이 나면 교통통제 전체 시스템이 작동되지 않아 엄청난 혼란을 야기할 수 있다는 것이다.

이러한 문제점을 해소하고, 자율주행 차량의 교통통제 시스템의 유연성을 증대하기 위해서는 자율주행 차량의 교차로 교통제어 시스템은 분산형 시스

템으로 개발이 되어야 한다. 즉, 자율주행 차량이 신호등 및 중앙통제 장치 없이 교차로를 안전하게 통과하기 위해서는 자율주행 차량들 간의 실시간 정보 교환을 통한 철저한 협업이 전제되어야 한다.

그러나 지금까지 교차로 교통제어에 대한 분산형 시스템의 연구는 거의 없었다. 2015년에 처음으로 교차로 교통 통제에 분산 시스템의 상호배제 이론을 적용한 논문 “A Distributed mutual exclusion algorithm for intersection traffic control” (Weigang Jiebin, Aoxue Lou, Jiannong Cao, 2015)이 발표되었다. 이는 허가기반의 상호배제 알고리즘으로서, 자율주행 차량들 간의 메시지 교환을 이용하여 교차로의 통과 우선순위를 결정하는 알고리즘을 설계하였다. 이 논문은 교차로 교통 제어 시스템에 상호배제 이론을 적용한 처음의 연구로서 큰 의의가 있다. 그러나 이는 허가기준의 그룹 상호배제 알고리즘으로서 차량의 교차로 교차구역(임계구역)에 진입에 대한 권한을 타임스탬프에 의한 우선순위로 결정해야 함에 따라 3회 이상의 메시지 브로드캐스트를 하여야 하는 등 많은 메시지 트래픽을 야기한다. 또한 차량 흐름의 측면에서 교차로에 진입한 순서대로 교차로 통과가 결정되므로 효율성이 저하될 가능성도 있다. 따라서 자율주행 차량의 교차로 교통제어에 대한 연구의 발전을 위해서는 메시지 트래픽을 줄이고 차량 흐름의 효율성을 제고시킬 수 있는 토큰 기반의 그룹 상호배제에 대한 추가적인 연구가 필요하다.

토큰 기반의 그룹 상호배제 알고리즘에 대한 주요 연구를 요약하면 다음과 같다. Mamun-Nagazato 알고리즘은 하나의 토큰으로 운영하여 메시지 트래픽은 효율적이나, 이는 임계구역 내에서 체류시간을 아는 경우에 최고의 성능을 내는 점에서 동적 상황을 전제로 하는 본 논문 적용에 있어 한계

가 있다. Mittal-Mohan 알고리즘에서는 효율성을 증대하기 위하여 주 토큰과 부 토큰 두 가지 종류의 토큰을 활용한다. 또한 이 논문의 핵심은 주 토큰을 이전하는 방법으로 그룹 간 가중치를 적용한 우선순위 기반 체계의 운영이다. Kakugawa-Kamei 알고리즘은 코테리에 근거한 토큰 기반의 그룹 상호배제 알고리즘을 제안하였다. 여기에서도 주 토큰과 부 토큰을 활용한다. 또한 메시지 트래픽을 줄이기 위하여 쿼럼을 사용한다.

본 논문이 일반적 상호배제 문제와 달리 갖는 어려움은 자율주행 차량의 주행 특성상 교차로 통과 후 교차로 내에 체류할 수 없는 것이다. 따라서 차량 진입 진출 및 차량의 수가 지속적으로 변경되는 동적인 환경에서 교차로 교통 제어를 안전하게 하면서 메시지 트래픽을 줄여야 하는 어려움이 있다. 이를 해결하기 위하여 본 논문은 주 토큰 이전을 효율적으로 순환하도록 토큰 기반의 알고리즘을 설계하였으며, 주 토큰과 부 토큰 두 가지 종류의 토큰을 사용한다. 또한 차량의 교차로 통과 효율성을 높이기 위하여 진입차선의 체인 개념을 도입하였다.

본 논문의 구성은 제 1장 서론에서 본 논문을 제안하게 된 연구 배경, 연구 목적 및 연구의 구성에 대해 기술하였다. 제 2장 관련 연구에서는 현재 진행되고 있는 교차로 교통통제 시스템에 대한 연구들에 대해 요약 정리하였으며, 토큰기반 그룹 상호배제 알고리즘 연구 중 본 논문과 관련성이 큰 4가지 논문 내용을 요약 기술한다. 제 3장에서는 본 알고리즘의 시스템 모델과 자율주행 차량의 유한상태 오토메타에 대해 설명하고, 본 알고리즘의 설계 내용인 정형적 프로토콜을 제시한다. 제 4장에서는 알고리즘을 실험하여 성능을 분석하는 실험 및 결과분석을 기술한다. 마지막으로 제 5장에서 본 논문의 결론을 도출하여 요약하고, 향후 방향을 기술한다.

II. 관련 연구

2.1 자율주행 차량의 교차로 통제

자율주행 차량의 기술 발전과 더불어 이에 관련한 교통통제에 대한 연구도 활발히 진행되고 있다. 이러한 자율주행 자동차의 교통통제 연구는 현재 도로교통의 혼잡과 사고 발생의 원천이 되고 있는 교차로에서의 교통통제가 핵심적인 분야로서 진행되고 있다. 이와 관련된 연구 동향을 살펴보면 교통신호등 통제 연구와 자율주행 차량 궤적관리 연구 두 가지로 구분된다.

2.1.1 교통 신호등 통제 연구

기존의 교차로 교통통제 연구는 교통신호 통제 연구가 주를 이루고 있으며, 이는 교통신호 편성으로 자동차가 초록 신호의 점멸에 따라 정지와 진행 스타일로 진행되는 방법이다. 최근 교통신호 연구는 컴퓨터의 인공 지능을 활용한 스마트 교통신호 스케쥴링에 초점을 맞추고 있다. 이렇게 교통공학에 활용되는 인공지능에는 알고리즘, 퍼지 로직, 뉴럴 네트워크 등이 포함된다.

교통신호에 의한 교차로 통제는 과거 1세기 동안 많은 연구와 개선이 이루어졌음에도 불구하고, 교통량의 증가와 변동성으로 인하여 교통신호를 통하여 최적화된 교통통제 시스템을 개발하는 데에는 아직 해결해야 할 문제

가 많은 실정이다. 더욱이 컴퓨터 인공지능 알고리즘의 복잡성은 실시간 교통신호 통제에 대한 최적화된 컴퓨터 알고리즘을 개발하는 것을 어렵게 한다. 또한 이 때문에 많은 컴퓨터 관련 비용이 수반된다. 이렇듯 교통신호 통제 연구는 교통 공학적인 연구 자체만으로는 아직 만족한 수준에는 도달하기 어려운 실정이다.

이와 관련된 교통신호 교차로 통제 시스템에 대한 연구를 서술하면 다음과 같다.

Malik *et al.*은 각 차선에 센서 노드들과 하나의 센서 네트워크 환경 이내에 각 차선에 컨트롤러를 설치함으로써 교통정보를 습득하는 방법과 최적 통과 순서를 찾는 알고리즘을 발표하였으며, Khalil *et al.*은 교차로에 하나의 교통 신호에 도착과 출발 노드의 세트를 설치함으로써 자율주행 차량의 정보를 수집하는 방법을 제안하였다.

Jeong *et al.*은 개별 차량의 지연 시간을 측정함으로써 디큐잉 시간을 측정하는 방법을 제시하였다. 또한 Lee and Oh는 상향, 하향 차선에 이미지 감지기 세트를 설치하여 대기길이를 측정하는 알고리즘을 제시하였다.

서울시는 COSMOS(Cycle, Offset, Split Model for Seoul)라는 이름의 교통신호 통제 시스템을 2001년부터 운영하고 있다. COSMOS는 직진 감지기, 좌회전 감지기, 교차로 입구의 교통 혼잡 감지기 등 루프 감지기를 설치함으로써 포화도와 혼잡도의 정도를 분석한다. 대기길이를 사용하여 교통을 통제하는 것은 실시간 통제 방법 중 가장 우수한 방법이다. 그러나 이의 단점은 이는 각 차선의 감지기로부터 받는 정보에 의지하기 때문에 각 차선에 감지기들이 설치되어야 하는 것이다.

이 이외에도 교통신호 통제와 관련되는 연구는 많으나 대표적인 연구로는

D. Ginat, A.U.(1989)의 Reinforcement of Learning in Neurofuzzy traffic signal control, Jan de Gier(2011)의 Traffic flow on realistic road network with adaptive traffic light, Behal Ali Alshami(2014)의 Adaptive dynamic multiple traffic light control system 등이 있다.

2.1.2 자율주행 차량의 궤적관리 연구

이에 따라 최근에는 교차로에 교통 신호등이 없이 교차로를 자동으로 통제(Autonomous Intersection Management; AIM)하는 차량 궤적관리 연구가 활발히 진행되고 있다. 차량의 궤적관리 연구는 인근 차량의 움직임을 사전에 파악하여 차량의 궤적을 예상하고, 교차로에서 차량의 중첩을 미연에 방지하는 것이다. 따라서 여기에서는 교차로에서의 차량의 상태를 파악하고, 차량 이동 궤적 포착을 통해 교차로 통과의 우선순위를 관리하는 것이 중요하다. 또한, 이를 위해서는 교차로에 중앙 통제장치(Intersection Control Unit; ICU)를 설치하여 운영하는 것이 필요하다. ICU는 계속적으로 근처 차량들과 정보를 나누고, 이를 통해 수집된 정보로 차량이 교차로를 통과할 수 있는 최적화된 궤적 및 교차로 통과 일련순서를 계산한다. 또한 이러한 수집 및 계산된 정보를 인근 차량들에게 실시간으로 전달하여 교차로 교통통제를 한다.

Li와 Wang(2006)은 모든 가능한 차량통과 일련순서를 나열하였으며, 이 중 가장 효과적인 일련순서를 찾는 궤적 계획 알고리즘을 제안하였다.

Wu et al. (2009)는 AIM에 근거하여 최적 해법을 얻으려는 다이내믹 프로그래밍 알고리즘을 발표하였다. 다이내믹 프로그래밍은 적은 교통 부하 아래에서 최적화 해결책을 찾기 위하여 사용되었고, 과중한 교통 부하 아래

에서 최적화에 근접하는 해결책을 이끌어 내기 위하여 개미집단 시스템을 사용하였다.

Makarem *et al.*은 실시간 스케쥴링을 가능하게 하고, 컴퓨테이션의 부하를 줄이기 위하여 소프트 제약조건과 컨트롤러를 소개하였다. 그러나 이 방법은 최적화의 문제를 해결하지 못하였다.

Kamel *et al.*은 교차로에서의 교통 흐름을 원활하게 하기 위하여 협조 체계를 설계하였다. 그러나 이 연구의 목적은 개별 차량의 운행 이력을 고려하지 않고 전반적인 교통 흐름을 개선하기 위함이었다.

차량궤적의 최적화와 함께 안전성도 AIM 연구의 중요한 부분이었으며, 차량궤적의 안전성은 적절한 궤적의 결합, 차량궤적에서의 차량의 속도 조절 등에 의하여 얻어진다. 이와 관련된 연구는 다음과 같다.

Drsner and Sone은 셀 기반의 교차로 예약 시스템을 활용한 교차로 관리 알고리즘을 개발하였다. 이는 교차로 관리자 프로그램이 일시적인 셀의 점유에 대한 요청을 협업하게 허락함으로써 각 자율주행 차량에 대한 안전한 통과를 보장하는 통행권이 보장된다.

Hafner *et al.*은 불완전한 상태의 정보와 장해요인 아래에서도 연속적으로 운영되는 시스템을 운영하기 위한 안전성 통제 시스템을 제안하였다. 이와 더불어 충돌방지 시스템을 설계하기 위한 이론적인 통제 방법을 제안하였으며, 이를 두 개의 차량에 실험을 하였다.

Kowshik *et al.*은 차량의 움직임을 분석하여 최악의 경우에도 안전성을 확보할 수 있는 하이브리드 시스템 구조를 제안하였다.

Milanes *et al.*은 차량이 교차로를 통과할 때 차량 간의 통신을 통하여 차량들의 상태를 파악하여 안전성을 확보하는 자동 퍼지 통제시스템을 설계하

였다. 차량들의 경로에 따른 다이나믹스를 가지고 이를 스케쥴링 문제로 변환하였다. 이들은 또한 스페인, 프랑스, 네델란드에서 각각 개발한 3대의 실제 자율주행 자동차의 교차로 통과를 프랑스에서 차량의 무선 통신 네트워크를 활용한 테스트 사이트에서 시현하였다.

자율주행 차량의 차량궤적에 관한 연구는 이외에도 Arem van et al., Wuthisuwong and Traecgter, Hausknecht 등이 AIM에 대한 알고리즘을 발표하였다.

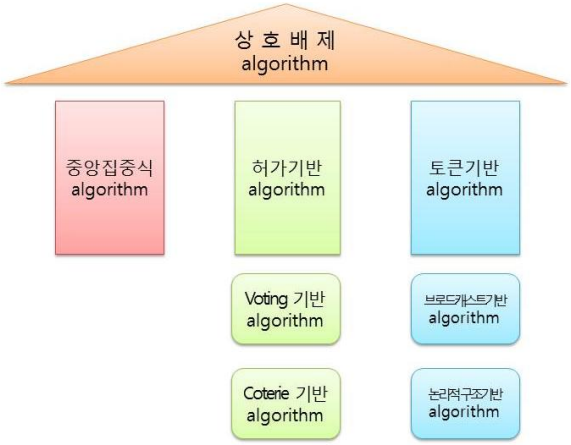
그러나 이러한 자율주행 차량의 궤적관리 연구도 기하급수적으로 증가하는 교통의 혼잡과 차량의 역동성 아래에서 최적 궤적을 산출하여 통제하는 것은 매우 어려운 일이며, 수많은 각 교차로 마다 기반시설을 구축하고, 유지관리 하는데 비용이 증가한다는 문제를 안고 있다.

따라서 교통의 복잡성과 차량의 변동성 아래에서 유연한 교차로 통제를 실현하기 위해서는 자율주행 차량 사이의 통신(V2V)을 기반으로 한 분산 시스템으로 연구 및 개발이 되어야 한다. 이에 반해 현재 진행되고 있는 연구의 주축인 자율주행 차량의 교차로 통제 연구는 분산형으로 이루어지는 것이 아니며, 컴퓨터의 클라이언트 서버와 유사한 중앙 통제장치에 의하는 것이다.

2.2 전통적 상호배제 알고리즘

본 장에서는 이러한 분산 시스템의 상호배제 문제를 다루는 알고리즘들을 분류하고, 이에 대한 연구를 정리한다. 상호배제 알고리즘은 <그림 2.1>에서 보는 것처럼 크게 다음 세 가지로 구분된다. 첫째는 중앙집중식 알고리

즘, 두 번째, 비 토큰기반 알고리즘(또는, 허가기반 알고리즘이라고도 한다), 셋째, 토큰기반 알고리즘이며, 이들 중 주요부분인 허가기반과 토큰기반 접근방식에 대한 특성을 간단히 살펴보면 다음과 같다.



<그림 2.1> 상호배제 알고리즘 구조도

(1) 허가기반 알고리즘

중앙 집중식 알고리즘의 문제점을 해결하기 위하여 의사결정을 전담하는 진행자 대신 그의 역할을 전체 시스템의 프로세스들에게 분산시키기 위하여 허가기반 알고리즘이 제안되었다. 허가기반 알고리즘은 임계영역에 진입하고자 하는 프로세스는 시스템 내의 다른 모든 프로세스 또는 대다수의 프로세스들로부터 허가를 받아야 임계영역 진입이 가능하도록 하는 알고리즘이다. 이러한 허가기반 알고리즘의 가장 대표적인 알고리즘은 램포트 알고리즘(Lamport L., 1978)과 이를 더 최적화한 리칼트 알고리즘(Recart G. ans A.K. Agrawala, 1981) 이다.

(2)토큰기반 알고리즘

토큰기반 알고리즘에서는 임계영역 사용에 대한 충돌 문제를 해결하기 위하여 토큰이라는 보조 수단을 활용하며, 토큰을 보유한 프로세스만이 임계영역에 진입할 수 있다. 또한 시스템에는 토큰이 하나만 존재하기 때문에, 임계영역을 사용하기를 원하는 프로세스는 프로세스 간 메시지 교환으로 현재 토큰 홀더로부터 토큰을 전달 받아야 한다. 이러한 토큰기반 알고리즘은 토큰 요청 및 전달 방식에 따라 다시 다음과 같이 세분된다.

- 브로드캐스트기반 알고리즘

  브로드캐스트기반 알고리즘에서는 임계구역에 진입하기를 원하는 프로세스는 다른 프로세스들에게 요청 메시지를 보내고, 토큰이 전달되기를 기다린다. 이의 대표적인 알고리즘으로 Suzuki & Kasami, Nishio et al. M. Singhal, Chang et al. 등이 있다.

- 논리적 구조 기반 알고리즘

  논리적 구조 기반 알고리즘은 트리, 링과 같은 논리적 구조를 사용한다. 임계영역에 대한 요청은 요청 프로세스와 토큰 보유 프로세스 사이의 논리적 구조의 경로를 따라 간다. 또한, 토큰도 토큰 보유 프로세스와 요청 프로세스 사이의 경로를 따라 간다.

  이의 대표적인 알고리즘은 Kerry Raymond 알고리즘이 있으며, 이와 관련된 알고리즘으로는 Argawala & Abbadi, Jun Kiniwa 등이 있다.

2.3 그룹 상호배제 알고리즘

상호배제 이론은 분산 시스템의 가정 근본적인 문제로서 40여 년 동안 이 문제에 대한 보다 효율적인 방법을 찾아내기 위하여 많은 연구들이 이루어져 왔다. 전술한 전통적인 상호배제 이론에서는 임계구역 내에 하나의 프로세스 이외의 다른 프로세스들은 임계구역 동시 진입 및 사용이 허락되지 않는다. 이 때문에 프로세스들은 임계구역을 사용하기 위하여 많은 시간을 기다려야 하는 문제가 있다. 이러한 전통적 상호배제 이론의 확장으로서 2002년에 그룹 상호배제 이론에 근거한 Congenial Talking Philosopher 문제(Joung, Y.j., 2002)가 발표되었다. 그룹 상호배제 이론에서는 전통적인 상호배제 문제와는 달리, 동일한 타입 또는 그룹에 속하는 두 개 이상의 프로세스들은 하나의 임계영역을 동시에 사용할 수 있다. 그러나 이때에 현재 임계영역을 사용 중인 프로세스들과 상이한 그룹에 속하는 프로세스들은 임계영역을 동시에 사용할 수 없으며, 이를 상호배제 방식으로 사용하여야 한다.

지금까지 이러한 그룹 상호배제 알고리즘에 대하여 많은 연구들이 이루어지고 있으며, 다음에는 최근에 연구된 그룹 상호배제 알고리즘 중 본 논문의 내용과 관련성이 있는 알고리즘의 내용을 기술한다.

2.3.1 Mamun-Nagazato 알고리즘

Mamun이 제안한 알고리즘은 토큰 기반의 그룹 상호배제 알고리즘이다(Quazi Ehsanul Kabir Mamun, Hidenori Nakazato, 2006). 이 알고리즘에서는 시스템 내에 하나의 프로세스에 의하여 단 하나의 토큰만이 존재한다. 토큰은 임계구역 진입을 기다리는 요청들로 구성된 대기열(아이디, 세션, 패

시지타임)을 보유한다. 여기서 프로세스가 임계구역 내에 체류하는 시간을 패시지 타임이라 한다.

어떤 프로세스가 세션에 접속하기를 원하면 토큰을 받은 후에 세션을 시작한다. 그리고 다른 모든 프로세스들에게 패시지타임을 포함한 세션 메시지를 브로드캐스트하여 세션을 선언한다. 세션을 선언하는 프로세스를 중심 프로세스라 한다. 또한 이러한 선언을 전달받은 모든 프로세스는 자신과 동일한 세션에는 중심 프로세스와 동시에 임계영역에 진입할 수 있다. 이를 달리 이야기 하면, 프로세스가 어느 특정한 세션의 임계영역에 접속하기 원하면, 그 프로세스는 진입구역에서 원하는 세션이 사용 가능한지를 확인한다. 사용이 가능하면 그 프로세스는 해당 세션을 동시에 사용할 수 있다. 만약 사용이 불가능하여 다른 세션을 접속하기를 원하면, 그 프로세스는 토큰을 받기 위해서 중심 프로세스에게 요청을 보낸다. 현재 세션이 종료되면 중심 프로세스는 토큰을 요청했던 프로세스에게 보낸다. 새로운 토큰 홀더인 P_k는 새로운 세션 Y를 선언하고, 이전 세션인 세션 X의 임계영역에 있던 프로세스들이 모두 나올 수 있도록 가드타임 동안 기다려야 한다.

이 알고리즘은 그룹 상호배제 문제의 새로운 토큰 기반의 프로토콜이 제안되고 분석되어 있다. 이 알고리즘은 임계구역 내에서 어느 정도의 시간을 사용할지를 알고 있는 프로세스 그룹에 대하여 최고의 성능을 낸다.

2.3.2 Mittal-Mohan 알고리즘

Mittal은 Suzuki-Kasami 알고리즘에 근거하여 토큰 기반 그룹 상호배제 알고리즘(Neeraj Mittal, Prajwal K. Mohan, 2007)을 제안하였다. 여기에는 알고리즘의 효율을 높이기 위하여 프라이머리 토큰과 세컨더리 토큰 두 가

지 종류의 토큰을 사용한다. 언제든지 시스템 내에는 오직 하나의 프라이머리 토큰만이 존재한다. 그러나 세컨더리 토큰의 수는 수시로 변할 수도 있다. 시스템을 시작할 때 프로세스 P_1은 프라이머리 토큰을 갖고, 세컨더리 토큰의 수는 제로이다. 하나의 토큰은 그와 관련된 하나의 타입(또는 그룹)을 갖고 또한 그 타입(또는 그룹)의 임계영역을 진입하는 데에만 사용될 수 있다. 프로세스는 일정 타입의 프라이머리 또는 세컨더리 토큰을 보유한 때에만 그 타입의 임계영역에 진입할 수 있다. 프라이머리 토큰과 세컨더리 토큰의 차이점은 다음과 같다: 프라이머리 토큰을 보유한 프로세스는 다른 프로세스에게 세컨더리 토큰을 발행하는 것이 허용된다. 그러나 세컨더리 토큰을 보유한 프로세스는 다른 프로세스에게 토큰을 발행하는 것이 허용되지 않는다.

프로세스는 임계영역을 요청하는 경우 그 시스템 내의 모든 프로세스들에게 요청 메시지를 보낸다. 프라이머리 토큰을 보유한 프로세스는 토큰의 타입과 동일한 타입의 임계영역에 대한 미 이행된 요청을 받은 때에, 그 요청하는 프로세스에게 세컨더리 토큰을 발행한다. 프라이머리 토큰을 보유한 프로세스는 토큰 사용이 완료되면, 프라이머리 토큰을 다른 프로세스에게 넘겨준다. 프로세스 P_i는 이전의 프라이머리 토큰 홀더 P_j로부터 토큰을 받은 때 그 토큰을 즉시 사용하지 못할 수도 있다. 이는 이전 세션에서 여러 개의 세컨더리 토큰이 발행되었고, 이들 토큰 중 약간은 아직 임계영역에 남아 있을 수 있기 때문이다. 따라서 새로운 토큰 홀더 P_i는 임계영역에 진입하기 전에 안전성 체크를 위하여 하나의 세션에 발행되는 세컨더리 토큰의 발행된 수와 이들이 임계영역을 나가면서 보낸 Release 메시지의 수를 비교한다.

또한, P_i는 자신의 세션을 종료한 후 다음 순서의 프라이머리 토큰 홀더를 결정하기 위하여 우선순위 기반 체계를 사용한다. 다음 세션의 타입을 결정하기 위하여 먼저 프라이머리 토큰 대기열 내의 모든 미결 요청들은 타입에 따라 그룹핑하여 부분집합으로 구분한다. 이처럼 나누어진 부분집합 중 다음 세션의 토큰 홀더로는 다음에 기술하는 두 파트의 합계의 값이 최대치인 부분집합을 선정한다. 첫 번째 파트는 그 부분집합 내의 요청들의 수에 의해 주어지는 부분집합의 사이즈이고, 두 번째 파트는 그 부분집합의 모든 요청들의 에이지의 합계에 의해 주어진다.

이 알고리즘은 어떠한 그룹이 다른 그룹들 보다 더 자주 요청되는 경우에 효과가 나며, 여기서는 다음 세션의 타입을 선택하는 체계가 중요한 역할을 한다.

2.3.3 Kakugawa-Kamei-Masuzawa 알고리즘

이들은 완전히 연결된 시스템들에서의 코테리에 근거한 토큰기반의 그룹 상호배제 문제를 해결하기 위한 알고리즘을 제안하였다(Hirotsugu Kakugawa, Sayaka Kamei, Toshimitsu Masuzawa, 2008). 이의 주요 내용은 다음과 같다. 이 알고리즘은 주 토큰과 부 토큰 두 가지 형태의 토큰을 운영한다. 주 토큰은 시스템 내에서 하나만 존재한다. 부 토큰은 주 토큰에 의하여 생성되고, 부 토큰의 개수는 변화한다. 임계영역에 프로세스가 없는 경우에 프로세스는 임계영역에 진입하기 위하여 주 토큰을 획득한다. 임계영역 내에 같은 그룹의 프로세스들이 있는 경우에는 그 프로세스는 부 토큰을 받음으로써 임계영역에 진입한다. 또한, 통신 복잡도를 줄이기 위하여 통신 구조로서 쿼럼을 사용한다. 이 알고리즘의 윤곽은 다음과 같다. ①프로세

스 P_i가 임계영역 진입을 원할 때, P_i는 쿼럼 내에 있는 각 프로세스 P_j에게 요청 메시지를 보내고, 토큰이 전달되기를 기다린다. ②주 토큰 홀더 P_k가 P_i의 요청 메시지를 전달 받으면, 요청 메시지를 주 토큰의 대기열에 보관한다. 대기열에 있는 요청들은 특정한 방법(Mittal의 알고리즘과 동일한 방법)에 의하여 계산된 우선순위에 의하여 토큰이 전달된다. ③프로세스 P_i는 주 토큰 또는 부 토큰을 받음에 의하여 임계영역에 진입하다. ④P_i가 임계영역을 나올 때는 주 토큰은 다음 세션의 주 토큰 홀더에게 이전되어 순환하고, 부 토큰 홀더는 릴리스 메시지를 보냄으로서 부 토큰을 반환한다.

이 알고리즘은 높은 동시진행도를 나타내는 성과를 얻었다. 그러나 이는 높은 동시진행도를 유지하면서 대기시간과 동기화 지연을 감소시키는 방안이 필요하다.

2.3.4 Weigang의 분산 교차로 교통통제 알고리즘

2015년에 처음으로 교차로 교통통제 시스템에 상호배제 이론이 적용된 알고리즘이 제안되었다(Weigang Wu, Jiebin Zhang, Aoxue Luo, 2015). 이는 허가기반의 그룹상호배제 알고리즘으로 기본적인 아이디어는 다음과 같다. 차량들은 교차로 통과를 위한 각각 다른 우선순위를 갖고 있으며, 이는 일반적으로 도착시간에 의하여 결정된다. 교차로에 진입한 차량은 통과 요청 메시지를 보내고, 그 요청 메시지 송신자 보다 빠른 우선순위를 갖고 있는 메시지 수신자는 리젝트 메시지를 보내어 요청을 보낸 차량을 진입하지 못하게 한다. 만약에 전체 메시지 수신자 중 어떠한 차량도 요청을 보낸 차량의 진입을 방해하지 않으면 송신자는 교차로를 통과할 수 있다. 또한 높은 효율성을 위하여 동시진행 차선에 있는 차량의 경우 동시에 통과할 수 있는

구조를 설계하였다. 이 알고리즘의 운영에 대해 다음과 같이 요약한다.

• *Request*의 진행

차량i가 대기구역에 진입할 때에 차량i는 진입 요청 메시지를 교차로 내의 전체 차량에게 통보한다. 그리고 차량i는 타임아웃을 설정해 놓고 다른 차량들로부터 리젝트 메시지를 기다린다. 이 때 차량j가 요청 메시지를 받으면, 차량i와 차량j가 동시진행 차선에 있으면 차량j는 차량i의 요청 메시지를 무시하고 리젝트 메시지를 보내지 않는다. 그러나 차량i와 차량j가 충돌 차선에 있으면, 차량j는 리젝트 메시지를 보낸다. 이러한 운영 체계에 따르면 동시진행 차선에 있는 두 차량은 만약 우선순위 상 두 차량 사이에 있는 차량이 충돌 차선에 있는 경우에 동시진행 차선의 두 차량은 동시진행을 할 수 없을 수도 있다. 이러한 문제를 해결하기 위해서는 추가적인 운영 체계인 선점방식을 적용한다.

• *Passing*의 진행

타임아웃이 되기까지 리젝트 메시지가 수신되지 않으면, 차량i는 교차로의 핵심구역을 통과하기 시작한다. 만약 이때 동일한 차선에 있는 다른 대기 차량들이 있을 때에는 이 차량들을 동시진행 리스트에 포함 시키고, 팔로우 메시지를 모든 차량에게 브로드캐스트 한다. 동시진행 리스트의 크기는 임계치(과거의 데이터 및 실시간 교통량에 의해 계산)에 의하여 제한되어야 한다. 차량j는 팔로우 메시지를 받았을 때 자신이 동시진행 리스트에 있으면 교차로의 핵심구역을 통과하기 시작한다. 여기서 동시진행 차선의 두 차량은 만약 차량 i와 j 가 충돌차선에 있지만, 다른 차량k가 i와 강한 동시진행 관계에 있는 경우에는 j는 i에게 진행을 양보한다. 이 때 차량j는 리젝트 메시지를 브로드캐스트 한다.

• *Release*의 진행

차량i는 교차로 핵심구역을 퇴장할 때 자신의 핵심구역 통과가 팔로우 메시지에 의한 것이 아니거나 또는 자신이 동시진행 리스트의 마지막 차량이면, 차량i는 퍼미트 메시지를 브로드캐스트 한다.

이 알고리즘은 자율주행 자동차의 교차로 교통통제에 분산시스템의 상호배제 이론을 처음 적용한 점에서 의미가 크다. 그러나 이는 허가기준의 상호배제 이론으로서 교차로 내의 모든 차량들이 서로 간에 우선순위를 알아야 하므로 요청 메시지, 팔로우 메시지, 퍼미트 메시지를 모든 차량에게 브로드캐스트 해야만 한다. 이 알고리즘은 이러한 메시지 브로드캐스트에 의하여 메시지 트래픽이 큰 수준이다.

III. 자율주행 차량의 교차로 트래픽 제어를 위한 상호배제 알고리즘

이 장에서는 자율주행 차량의 교차로에서 통행 제어를 위한 토큰 기반의 그룹 상호배제 알고리즘(Group Mutual Exclusion Algorithm for Intersection traffic control of Autonomous Vehicle; *VTokenIC*)에 대하여 기술한다. 본 논문에서는 자율주행 차량이 차량 사이의 메시지 교환으로 협업하여 교차로를 현재의 교통 신호체계 없이 통과하는 자율적 교차로 통과 시스템을 설계한다.

3.1 시스템 모델과 상호배제 속성

*VTokenIC*의 시스템 모델에서 사용되는 표기법은 <표 3.1>과 같다.

<표 3.1> 시스템 모델 표기법

N	the number of processes
$P_1,P_2,..,P_n$	processes
e, f, g	events
s, t, u	local states
prev(s)	the state immediately before s
next(s)	the state immediately next to s
$\rightarrow$	happened before relation
$\rightsquigarrow$	remotely precedes relation
$<_{im}$	immediately precedes relation
$<$	locally precedes relation
$\parallel$	concurrent relation

3.1.1 시스템 모델과 전제조건

*VTokenIC*의 시스템 내에 진입하는 차량의 수는 유동적이다. 그러나 특정 시점에서의 교차로 내에 들어와 있는 차량의 수는 셀 수 있는 가산집합이다. 따라서 시스템에 해당하는 교차로는 일습의 채널을 통하여 메시지를 송수신함으로써 차량 간에 통신하는 AV={AV_1,AV_2,...,AV_n}의 n개의 자율주행 차량으로 구성된다. 자율주행 차량은 위치한 진입차선 별로 AV의 그룹 G={grp_1, grp_2, ..., grp_8}을 구성한다. 따라서 진입차선에 진입한 $AV_i \in AV$는 교차구역에 진입하는 기준인 하나의 $grp_i \in G$에 해당한다.

모바일 애드 혹 네트워크 모델은 무향그래프(undirected graph) G=(V,E)로 정의한다. 정점들(vertices)의 집합 V={v_1,v_2,...,v_n}(n≥1)의 각 정점은 모바일 노드를 나타낸다. 각각의 통신 링크는 양 방향성이라 가정한다. 또한 집합 E는 모든 노드 i,j가 정점집합 V의 원소일 때, 에지 (i,j)가 에지(edge) 집합 E의 원소이다.

본 논문은 이벤트(event)들의 순서로 구성되어 있고, 이러한 이벤트들의 진행을 캡쳐링하는데 *happened before* 모델(Vijay k. Garg, Element of Distributed Computing)을 적용한다. *happened before* 모델이란 각 프로세스에게 발생하는 사건과 상태에 대해 논리적인 시간의 개념을 이용하여 순서를 나타내는 것이다. 하나의 프로세스의 진행 내에서 이벤트들 간의 *immediately precedes* 관계는 $e <_{im} f$로 표시하며, 이는 이벤트 *e*는 이벤트 *f* 보다 앞선다는 것을 의미한다. 또한, 만약 *e*는 하나의 메시지의 전송 이벤트이고, *f*는 동일한 메시지의 수신 이벤트라면, P_i 내의 이벤트 *e*는 P_j 내의 *f* 이벤트는 *remotely precedes*라고 한다. 이러한 관계들을 바탕으로 *happened before* 모델은 다음 두 가지 조건을 만족하여야 한다.

• $(e <_{im} f) \vee (e \rightsquigarrow f) \Rightarrow (e \rightarrow f)$, and

• $\exists g : (e \rightarrow g) \wedge (g \rightarrow f) \Rightarrow (e \rightarrow f)$

만약, 위 두 조건을 만족하지 않으면, 이벤트 e와 f는 *concurrent* 관계라 하며, $e \| f$로 표시한다.

또한, 네트워크 시스템에 대한 제약조건 및 가정을 다음과 같이 한다.

(1) 각 차량은 다른 차량과 식별 가능한 고유한 아이디를 갖고 있다.

(2) 차량 간의 네트워크의 형태는 차량 간 1:1 통신이 가능한 완전 네트워크(full network)를 가정한다.

(3) 자율주행 자동차들은 차량 내에 탑재된 무선 통신기기를 사용하여 차량 간의 메시지 송신, 수신을 하여 차량 간 통신을 할 수 있다. 이 때 차량의 아이디는 차량 간 통신 주소로 사용되며, 통신기기의 전송 범위는 교차로의 길이 보다 크다고 가정한다.

(4) 메시지 통신 시 네트워크 장애는 발생하지 않는 것으로 가정한다. 이는 교차로 내의 차량들은 원 홉 애드 혹 네트워크를 구성해서 각 차량 사이에 직접 통신을 할 수 있음을 의미한다.

(5) 무선 네트워크는 메시지 손실을 보기 쉬우나, 본 논문이 가정한 원홉 통신은 멀티 홉 네트워크에 비해 신뢰 수준이 훨씬 높으므로, 메시지 손실은 없는 것으로 가정한다.

3.1.2 일반적 상호배제 속성

*VTokenIC*에서는 자율주행 차량이 교차로에 중앙 통제장치 없이 차량들 사이에 실시간 정보 교환을 통하여 철저한 협업이 이루어져야 한다. 자율주행 차량이 서로간의 메시지 교환만으로 협업 및 경쟁을 통해 교차로를 충돌

없이 통과하기 위해서는 프로세스가 임계영역을 경쟁적으로 사용하는 상호배제 속성이 지켜져야 한다. 분산 시스템에서 상호배제 문제는 고정된 수량의 프로세스들과 임계영역(critical section)이라 불리는 하나의 공유자원으로 구성된 시스템을 가정한다. 이처럼 여러 개의 프로세스들이 하나의 임계영역에 협력적으로 접속하는 알고리즘은 다음의 세가지 속성을 만족하여야 한다.

(1) 안전성(Safety)

안전성 속성은 두 개의 프로세스가 동시에 임계영역을 사용하는 허락을 받아서는 안 되는 것이다. 이를 *happened before model*에서 정형적으로 기술하면 다음과 같이 표시된다.

$$(s \parallel t) \wedge (s \neq t) \Rightarrow \neg(cs(s) \wedge cs(t))$$

(2) 활동성(Liveness)

활동성 속성은 임계영역을 요청한 모든 요청은 필연적으로 허락되어야 하는 것으로서, 모든 요청은 필히 이행이 되어야 하는 것을 말한다. 이를 정형적으로 기술하면 다음과 같이 표시된다.

$$req(s) \Rightarrow (\exists t: s \lesssim t \wedge cs(t))$$

(3) 공정성(Fairness):

공정성은 서로 다른 요청들은 그 요청이 생성된 순서대로 허락이 되어야 하는 것이다. 이를 정형적으로 기술하면 다음과 같이 표시된다.

$$(req\_start(s) \wedge req\_start(t) \wedge s \rightarrow t) \Rightarrow next\_cs(s) \rightarrow next\_cs(t)$$

3.1.3 *VTokenIC*의 그룹 상호배제 속성

*VTokenIC*에서는 교차로에 진입한 각 차량은 자신이 원하는 방향으로 교차구역(임계영역)을 통과하는 요청을 하고, 해당 진입차선에서 도착 순서대로 기다린다. 교차로 내에서의 안전 운행을 위하여 통과 권한을 받은 차량만 교차구역에 진입을 할 수 있으며, 교차로 운영의 효율성을 위하여 교차구역의 동시통과를 허용한다. 따라서 교차구역 통과 권한을 받은 차량과 동시진행 관계의 차선에 위치한 차량들은 동일 그룹으로서 교차구역을 동시에 진입하여 통과할 수 있다. 즉, *VTokenIC*에서는 일반적 상호배제 문제와 달리 "동시진행"과 "상호배제"의 문제를 함께 다루는 그룹 상호배제 이론을 적용한다. 이러한 점들을 고려하여 본 알고리즘의 그룹 상호배제 속성을 정의하면 다음과 같다.

(1) 안전성(safety)

안전성 속성은 현재 교차구역(임계영역) 내에 있는 차량과 다른 그룹에 속하는 차량은 교차구역에 동시에 있을 수 없다는 것이다. 즉, 만약 두 대 이상의 차량이 교차구역 내에 에 있으면, 이 차량들은 동일한 그룹에 속한다. 이를 정형적으로 표시하면 다음과 같다.

$$(\forall i,j \in AV:\ i \neq j:\ g_i,g_j \in G:\ s \in g_i,\ t \in g_j:\ (s \parallel t) \wedge (s \neq t) \Rightarrow \neg(s.inCZ \wedge t.inCZ))$$

(2) 활동성(liveness)

대기구역에 진입하여 교차구역 통과를 요청한 차량은 필히 교차구역에 진입한다. 이를 정형적으로 표시하면 다음과 같다.

$$req(s) \Rightarrow (\exists t: ((s \parallel t) \wedge (s \neq t)) \wedge cz(t))$$

(3) 동시진행성(concurrent entering)

만약 모든 차량의 요청들이 동일한 그룹에 속하는 경우에는, 차량은 다른 차량이 그 교차구역을 떠날 때까지 기다리게 요청되어서는 안 된다.

3.2 자율주행 차량의 유한상태 오토메타(FSA)

오토메타 이론은 컴퓨터 시스템을 사용한 기계들의 기본 사항으로서, 소프트웨어의 디지털 회로와 전자기계에서 기본적인 규칙을 제공하는 중요한 부문이다. 또한 이는 유한상태를 포함한 모든 타입의 시스템들을 확인하는데 사용된다. 유한상태 오토메타(Finite State Automata; FSA)는 컴퓨터를 사용한 기계들의 설계에 사용되는 수학적 모델이다(Ashwag Alrehily. *et al*, 2015,). 본 알고리즘의 FSA 모델링을 위한 표기법은 <표 3.2>와 같다.

<표 3.2> FSA 표기법

AV_i	QZ에 진입한 알고리즘의 대표 AV
QZ	교차로 내의 대기구역
CZ	교차로 내의 교차구역
EZ	교차로 내의 진출구역
IL	QZ내의 진입차선 (IL_1-IL_8)
Ig	진입차선의 그리드 ($Ig_{j,k}$)
Cg	교차구역의 그리드 (Cg_1-Cg_{16})
CL	교차구역 내의 AV_i의 경로 (CL_1-CL_{12})
CSG	주 토큰 홀더 AV_i와 동시진행 그룹

자율주행 차량 AV_i가 섹션 3.1에서 서술한 시스템 모델 아래에서 교차로에서 움직이고 통제되는 것은 오토메타 이론에 포함된다. 따라서 교차로 내에서 AV_i의 움직임을 설명하기 위하여 FSA로 모델링 하는 것이 필요하다.

본 알고리즘에서 AV_i의 FSA는 다음의 4-tuple 모델로 정리할 수 있다.

$$VTokenIC = (\ \mathbf{E},\ \mathbf{S},\ \mathbf{N},\ \mathbf{F}\ )$$

• **E** : AV_i의 움직임에 영향을 미치는 물리적 환경 요인으로서 교차로와 차선이 이에 해당한다.

$E = \{QZ, (IL_1, IL_2,..., IL_8)\}, \{CZ, (Cg_1, Cg_2,..., Cg_{16})\}$

• **S** : AV_i의 움직임에 대한 상태로서, 교차로에 진입하여 통과하여 나가는 과정에서 3 단계의 상태로 나타난다.

$S = \{approaching, waiting(w\text{-}move, w\text{-}CZ), passing\}$

• **N** : input으로서 AV_i가 교차로에서 움직임 중 받는 정보와 메시지를 의미한다.

N-1 = {camera sensor, GPS, digital map}

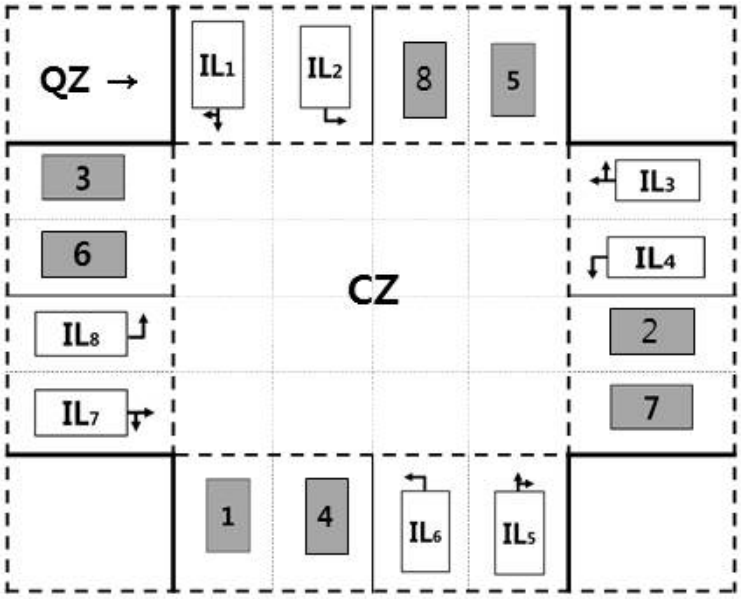
N-2 = {*request, ack, send-pt, ... , send-stch-tmp*}

• **F** : 교차로에서의 AV의 움직임 및 통제에 관한 함수

$F=\{f_1, f_2, f_3,.....f_n\}$이며, 이에 대한 세부내용은 섹션 3.4에서 기술한다.

3.2.1 (E) : 물리적 환경요인

AV_i의 오토메타 모델에서 물리적 환경은 AV_i가 운행되고 통제되는 전체 공간이며, AV_i가 위치하고 관리되는 시스템의 범위이기도 하다. 따라서 이에는 <그림 3.1>에 나타난 하나의 독립된 교차로 및 차선이 해당된다.



<그림 3.1> 교차로와 차선

(1) 교차로

본 알고리즘은 자율주행 차량의 교차로에서의 교통통제를 다루고 있으며, <그림 3.1>에서 나타난 것처럼 4지, 왕복 4차선의 교차로를 가정한다. 여기서 큰 점선 사각형은 교차로에 해당되며, 시스템 범위에 해당된다. 작은 점선의 사각형은 교차로의 교차구역(Cross Zone; CZ)이며, 자율주행 자동차들이 동시에 진입할 경우 충돌이 발생할 수 있는 구역이다. 즉, 교차구역은 일

반 상호배제 알고리즘의 임계구역에 해당한다. 또한, 큰 사각형(교차로)에서 작은 사각형(교차구역)을 제외한 구역은 대기구역(Queue Zone: QZ)과 진출구역(Exit Zone: EZ) 두 가지의 구역으로 세분된다. 여기서 대기구역은 AV_i가 교차로의 교차구역을 통과하기 위하여 대기하는 구역이며, 대기하는 열로서 나타난다. 또한, 그 AV_i는 교차구역 통과한 후 퇴장하면 한정된 시간 이내에 교차로 밖으로 나오게 된다. 여기서 AV_i가 교차구역 나와서 교차로 밖으로 나갈 때까지의 구역을 진출구역이라 한다. 따라서 교차로는 대기구역 QZ, 교차구역 CZ, 진출구역 EZ으로 구성된다.

(2) 대기구역과 진입차선

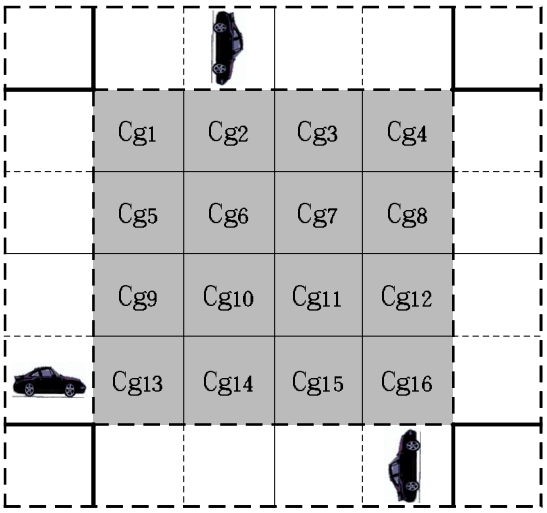
교차로의 교차구역을 통과하기 원하는 AV_i는 교차로 내의 대기구역으로 진입하여 교차구역 진입을 기다려야 한다. 즉, 대기구역은 AV_i가 교차구역 통과에 대한 권한을 얻기 위하여 기다리는 구역이다. 대기구역은 <그림 3.1>에서 나타난 바와 같이 진입차선(Input Lane; IL) 으로 구성되며, 총 8개의 진입차선이 방향별로 존재한다. 설명의 간편성을 위하여 진입차선은 1에서 8까지 번호를 붙이고, 표시는 각각 IL_1 ...IL_8으로 한다. 진입차선 중 홀수 차선(IL_1, IL_3, L_5, IL_7)에서는 직진 및 우회전이 가능하며, 짝수 차선(IL_2, IL_4, IL_6, IL_8)에서는 좌회전만이 가능하다.

모든 진입차선은 그리드로 세분이 되며, 이러한 진입차선 그리드는 Ig(Input grid)라 부르며, j번 진입차선의 k번째 그리드를 $Ig_{j,k}$로 표시한다 <그림 3.3>. 각 그리드에는 한 대의 차량이 들어 갈 수 있으며, 만약 진입차선의 길이가 70m이고, 하나의 그리드를 5m(차량의 길이 4m)로 가정하면, 하나의 진입차선은 14개의 그리드로 세분이 된다.

(3) 교차구역과 교차구역 그리드

교차구역은 자율주행 차량들이 이를 동시에 통과할 경우 차량의 경로가 교차하여 차량 간의 충돌이 발생할 수 있는 지역이다. 이러한 교차구역 내에서 충돌을 방지하고 안전한 운행관리를 위하여 교차구역을 그리드로 세분화한다(Penglin Dai, Qingfeng Zhuge, 2016).

<그림 3.2>에서처럼 교차구역을 직교하는 차선들을 결합하면, 교차구역은 여러 개의 작은 구역으로 나누어지며, 이를 교차 그리드(Cross Grid; CG)라 부른다. 본 시스템의 8차선 교차로는 16개의 그리드로 세분되는 것을 보여준다. 보다 분명한 표시를 위하여 i가 교차 그리드의 지수일 때($i \geq 1$) 교차 그리드를 $Cg_i \in CG$로 표시한다. 이 교차 그리드를 집합 개념으로 표시하면 $CG = \{Cg_1, Cg_2, \ldots, Cg_{16}\}$ 이다.



<그림 3.2> 교차 그리드

또한, 자율주행 차량이 교차구역에 진입하여 통과하는 경로상의 교차 그리드의 집합을 교차구역 논리적 차선(CZ Logical Lane)이라 하며, CL로 표시하기로 한다(Penglin Dai, Qingfeng Zhuge, 2016). 즉, 교차구역 논리적 차선은 대기구역 진입차선과 연결된 교차구역 내의 통과경로이다. 따라서 교차구역 논리적 차선은 교차구역 내에 진입차선 방향별(직진, 좌회전, 우회전)로 4개씩 총 12개가 존재한다. 이를 보다 분명하게 표시하기 위하여, i가 CL의 지수일 때(i≥1) 교차구역 논리적 차선을 $CL_i \in CL$로 표시한다. 이 교차구역 논리적 차선을 집합 개념으로 표시하면 $CL = \{CL_1, CL_2, \ldots, CL_{12}\}$이다. 예를 들어, 1번 진입차선 IL_1으로 진입하여 직진하는 AV_i는 교차구역 논리적 차선 $CL_1=\{Cg_1, Cg_5, Cg_9, Cg_{13}\}$을 통하여 교차구역을 이동한다. 교차구역 논리적 차선의 전체 내용을 표시하면 아래 <표 3.3>과 같다.

<표 3.3> CL 명세

구분	진입차선	CL	구분	진입차선	CL
CL_1	IL_1-직진	Cg_1,Cg_5,Cg_9, Cg_{13}	CL_7	IL_5-직진	Cg_{16},Cg_{12},Cg_8,Cg_4
CL_2	IL_1-우회전	Cg_1	CL_8	IL_5-우회전	Cg_{16}
CL_3	IL_2-좌회전	Cg_2,Cg_6,Cg_{11}, Cg_{12}	CL_9	IL_6-좌회전	Cg_{15},Cg_{11},Cg_6,Cg_5
CL_4	IL_3-직진	Cg_4,Cg_3,Cg_2, Cg_1	CL_{10}	IL_7-직진	Cg_{13},Cg_{14},Cg_{15},Cg_{16}
CL_5	IL_3-우회전	Cg_4	CL_{11}	IL_7-우회전	Cg_{13}
CL_6	IL_4-좌회전	Cg_8,Cg_7,Cg_{10}, Cg_{14}	CL_{12}	IL_8-좌회전	Cg_9,Cg_{10},Cg_7,Cg_3

① 충돌 관계와 동시진행 관계

교차구역은 차량들이 동시에 진입하여 통과할 경우 충돌이 발생할 수 있는 구역이다. 여기서 교차구역 내의 두 개의 논리적 차선 CL_i와 CL_j에서 AV_i와 AV_j가 동시에 통과할 경우 이들 간의 관계는 다음과 같이 기술될 수 있다.

교차구역 내에서의 논리적 차선간의 관계를 분석하기 위하여 진입차선 IL_2에서 교차구역에 진입하여 좌회전하는 AV_i와 IL_5에서 교차구역에 진입하여 직진하는 AV_j가 교차구역을 동시에 통과하는 경우를 가정해 본다. 이 경우 IL_2에서 교차구역에 진입하는 AV_i는 논리적 차선 CL_3={Cg_2, Cg_6, Cg_{11}, Cg_{12}}를 통과한다. 또한 IL_5에서 교차구역으로 진입해서 직진하는 AV_j는 논리적 차선 CL_7={Cg_{16}, Cg_{12}, Cg_8, Cg_4}를 통과한다. 이 경우, AV_i와 AV_j는 CL_3와 CL_7의 경로상 Cg_{12}에서 접점이 발생이 되어 충돌이 발생될 수 있다. 이러한 AV_i와 AV_j 또는 IL_2-좌회전과 IL_5-직진 사이의 관계를 “충돌관계”라 한다(Penglin Dai, Qingfeng Zhuge, 2016).

이와 다르게 진입차선 IL_1에서 교차구역으로 진입하여 직진하는 AV_i와 IL_5에서 진입하여 직진하는 AV_j가 교차구역에 동시에 통과하는 경우를 가정해 보자. 이 경우에는 AV_i의 논리적 차선 CL_1={Cg_1, Cg_5, Cg_9, Cg_{13}}과 AV_j의 논리적 차선 CL_7={Cg_{16}, Cg_{12}, Cg_8, Cg_4} 상에 접점이 없으므로 CZ를 서로 충돌 없이 동시에 통과할 수 있다. 이러한 AV_i와 AV_j 또는 IL_1-직진과 IL_5-직진 사이의 관계를 “동시진행 관계”라 한다.

② 동시진행 그룹

여기서 하나의 진입차선 IL_i을 기준으로 했을 때 IL_i과 동시진행 관계의 차선들의 조합(Compatible Stream Group; CSG)을 구성할 수 있다(Fei Yan, Mahjoub Dridi, 2013)해 본다. 예를 들어, IL_1-직진 차선과 동시진행 관계의 차선은 IL_1-우회전, IL_1-좌회전, IL_4-좌회전, IL_5-직진, IL_5-우회전 등 5개 차선이다. 이러한 동시진행 차선 조합들을 동시진행 그룹(CSG)이라 한다. 본 알고리즘에 적용하는 동시진행 그룹은 현재 신호등 교통체계에서 적

용하는 직진 후 좌회전 시스템을 적용한다. 따라서 IL_1-직진 차선과 동시진행이 되는 차선은 상기 5 개 차선 중 IL_1-우회전과 IL_5-직진의 두 차선이다. 이러한 본 알고리즘의 동시진행 그룹을 표시하면 <표 3.4>와 같다.

<표 3.4> 동시진행 그룹(CLG)

CSG	기준 진입차선	동시진행 차선		
CSG_1	IL_1-직진	IL_5-직진	IL_1-우회전	IL_5-우회전
CSG_2	IL_2-좌회전	IL_6-좌회전		
CSG_3	IL_3-직진	IL_7-직진	IL_3-우회전	IL_7-우회전
CSG_4	IL_4-좌회전	IL_8-좌회전		
CSG_5	IL_5-직진	IL_1-직진	IL_5-우회전	IL_1-우회전
CSG_6	IL_6-좌회전	IL_2-좌회전		
CSG_7	IL_7-직진	IL_3-직진	IL_7-우회전	IL_3-우회전
CSG_8	IL_8-좌회전	IL_4-좌회전		

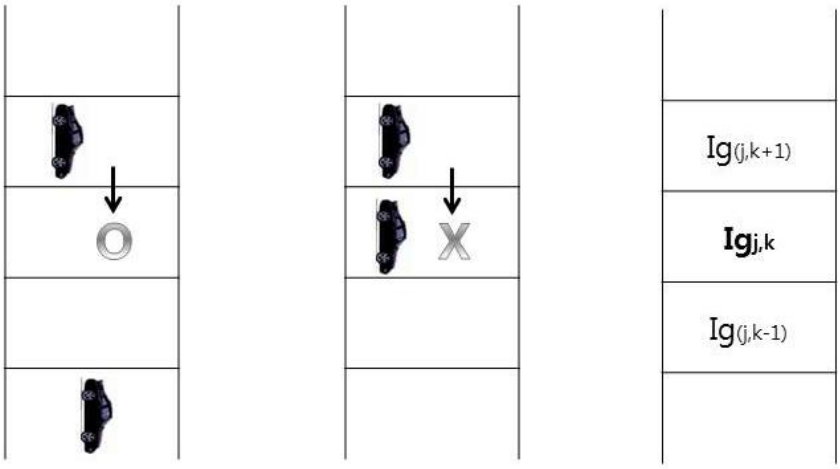
3.2.2 (S) : AV_i의 움직임과 상태

교차로에서 AV_i가 이동하는 것은 교차로의 그리드의 상태에 따라 결정되며, AV_i의 상태는 교차로 내의 위치에 따라 달라진다.

(1) 그리드 상태와 AV_i의 이동 원칙

교차로의 대기구역에 진입한 AV_i의 이동은 진입차선 그리드의 상태에 의하여 결정된다. AV_i가 위치한 그리드 $Ig_{j,k}$(j번 진입차선의 k번째 그리드)의 바로 앞의 그리드 $Ig_{j,(k-1)}$의 상태가 *off*(앞의 그리드가 *empty*) 이거나, *on*에서 *off*로 변경된(*occupied*에서 *empty*로 변경된) 경우에는, AV_i는 *off* 상태인 앞 그리드로 그리드를 하나 이동한다<그림 3.3>. 이 원칙은 교차로 전체 구역에서 모든 차량에게 병행적, 연속적으로 적용된다. 이 때 AV_i의 진로는

차선 및 그리드를 기준으로 결정되며, 차선 변경은 안 되는 것을 원칙으로 한다. 즉, 각 차선에 있는 차량들은 교차로를 선입선출(First-in First-out; FIFO)방식으로 통과하여야 하는 것을 의미한다.



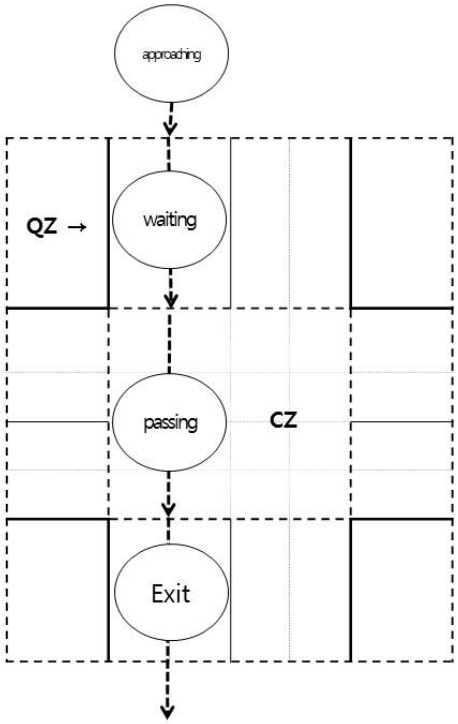
<그림 3.3> AV_i의 그리드 이동 원칙

(2) AV_i의 움직임과 상태

AV_i의 움직임은 자신에게 전달되는 정보 및 메시지에 의하여 결정되며, AV_i가 위치하는 구역에 따라 AV_i의 상태가 변경된다. AV_i가 교차로를 통과하는 절차에 따른 상태는 <그림 3.4>에 표시한 것처럼 세 가지로 기술될 수 있다.

• *approaching* 상태

이는 AV_i가 교차로의 시스템 밖에서 교차로에 진입하기를 원하여 교차로로 어프로칭 하고 있는 상태이다. 이때는 AV_i가 시스템 밖에 있으므로 교차로 통제 시스템의 제어를 받지 않는다.



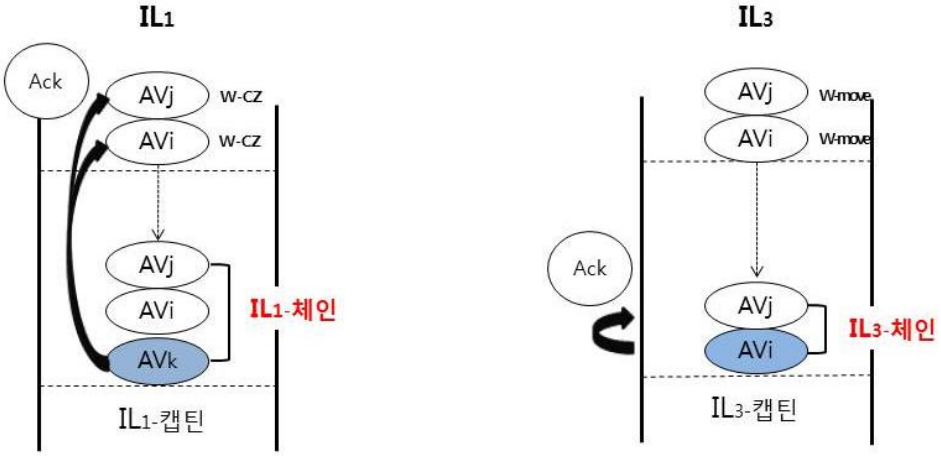
<그림 3.4> AV 상태 변화도

• *waiting* 상태

AV_i는 대기구역에 진입하기 위하여 어프로치하면 대기구역의 경계선에 접하여 센서로부터 경계선 정보를 받게 된다. 이 때 AV_i는 교차구역 진입에 대한 요청 메시지 전송 후에 대기구역 내의 진입차선(IL_i)을 이동하여 대기하여야 한다. 이처럼 AV_i가 대기구역 내의 진입차선(IL_i)에 진입한 때로부터 교차구역에 진입하기 전까지의 상태를 *waiting*이라 하며, 이는 다시 AV_i가 요청 메시지 전송 후 <*ack*>메시지를 받았는지 여부에 따라 *w-move*와 *w-CZ* 상태로 구분되어 진다.

AV_i는 진입차선(IL_i)에 진입하면 바로 *w-move* 상태가 되며, 그 이후 자

신의 진입차선(IL_i)의 첫 번째 그리드($Ig_{i,1}$)에 위치한 캡틴으로부터 <*ack*> 메시지를 받으면 AV_i의 상태는 교차구역 진입의 준비를 한 *w-CZ*로 변경된다. 이때 AV_i는 자신의 진입차선(IL_i) 체인의 하나의 멤버가 된다. 여기서 진입차선 체인은 진입차선의 캡틴이 팔로우어들에게 <*ack*> 메시지를 전송하여 생성하는 일련의 그룹이며, 향후 교차구역 진입 및 퇴장을 체인 단위로 연결하여 하게 된다. 따라서 진입차선(IL_i)의 체인은 IL_i 내에 *w-CZ* 상태로 존재하는 차량들의 그룹이다. 전술한 과정을 도해하면 <그림 3.5>로 나타난다.



<그림 3.5> 진입차선 체인의 형성 및 상태

이와 반면에, AV_i가 진입차선(IL_i)에 진입하여 <*ack*> 메시지를 받지 못하면, *w-move* 상태에서 진입차선 그리드를 이동하게 된다. 또한 이때에 IL_i 내에 AV_i 보다 이전에 진입한 차량이 없으면, AV_i는 *w-move* 상태에서 IL_i의 맨 앞 그리드($Ig_{k,1}$) 즉, 교차구역 진입 경계선에 접하게 된다. 이처럼 AV_i는 *w-move* 상태에서 교차구역 진입 경계선 정보를 받게 되면, *w-CZ*

로 상태가 변경되며, IL_i 체인의 새로운 캡틴이 된다. 이러한 AV_i의 움직임 및 상태의 변화를 도해하면 <그림 3.5>와 같다.

• *passing* 상태

AV_i가 토큰을 전달받아 통과권한을 얻은 후에 교차구역에 진입한 상태이다. 이는 AV_i가 교차로 진입 허락을 받아서 교차구역에 진입한 시점과 교차구역을 통과하여 나가는 시점 사이가 *passing* 상태이다.

3.2.3 (N) : AV_i에게 전달되는 정보 및 메시지(input)

AV_i의 움직임 및 통제는 철저하게 다른 차량들과 송수신하는 메시지와 자신에게 전달되는 센서 등의 정보에 의하여 이루어진다.

(1) 각종 정보 수신

AV_i는 차량 내에 장착된 센서, GPS와 디지털 맵으로부터 다음과 같은 정보를 전달 받는다.

- AV_i는 교차로의 경계선(대기구역의 진입 경계선, 교차구역 진입 경계선, 교차구역 퇴장 경계선)에 접하는 경우에 센서로부터 정보를 받아 자신이 그 경계선에 도착했음을 인지할 수 있다.
- AV_i는 센서 또는 디지털 맵으로부터 정보를 받아 자신이 위치한 차선 및 그리드 위치를 알 수 있다.
- AV_i는 자신과 직전, 직후의 그리드의 상태 및 그 그리드에 위치한 차량을 인식할 수 있으며, 이에 따라 교차로 내 그리드를 충돌 없이 규칙적으로 이동할 수 있다.

• AV_i는 대기구역 진입 경계선에 접하면, 시스템의 범위에 해당하는 교차로 전체 구역 내에 다른 차량의 존재여부에 대한 정보를 전달받는다. AV_i는 대기구역 진입 시에 교차로 내에 어떠한 아이디를 가진 차량이 있는지는 알 수 없으나, 자신 이외의 차량이 교차로 내에 존재하는지 여부에 대해서는 GPS로부터 정보를 받아 인지한다.

• AV_i는 교차구역 진입 경계선에 접하면, 교차구역 내에 차량의 존재여부에 대한 정보를 전달 받는다. AV_i가 주 토큰 홀더인 경우에는 교차구역 진입 전에 교차구역 진입에 대한 안전성 확인을 위하여 교차구역 내에 차량의 존재여부를 GPS로부터 정보를 받아 인지한다.

(2) 메시지의 송수신

본 알고리즘은 메시지 기반의 알고리즘으로서 AV_i는 다른 차량과의 의사소통을 철저히 메시지의 송수신에 의하여만 실시한다. 여기서 사용되는 메시지의 종류 및 내용을 요약하면 아래 <표 3.5>과 같다.

<표 3.5> 메시지 형태

<*request*>	AV_i가 교차구역에 진입하기 위하여 교차로 내의 모든 차량에게 전송하는 요청 메시지
<*ack*>	진입차선 캡틴 AV_i가 대기구역에 진입한 AV_j의 <*request*>에 대하여 보내는 응답 메시지
<*send-pt*>	현재 세션의 주 토큰 홀더 AV_i가 다음 세션의 넥스트 홀더에게 주 토큰을 전달할 때 사용하는 메시지
<*send-st*>	주 토큰 홀더 AV_i가 동시진행 진입차선의 캡틴에게 부 토큰을 생성하여 전달할 때 사용하는 메시지
<*chainmember*>	AV_i가 부 토큰 홀더가 되는 때에 주 토큰 홀더에게 자신의 체인 멤버들을 알리는 메시지
<*iNCZ*>	주 토큰 홀더인 AV_i가 교차구역에 진입할 때 부 토큰 홀더에게 알려주는 메시지
<*send-ptch-tmp*>	주 토큰 홀더가 자신의 체인 마지막 차량에게 주 토큰을 임시로 전달하는 메시지
<*send-stch-tmp*>	주 토큰 홀더 체인의 마지막 차량이 부 토큰 홀더 체인의 마지막 차량에게 주 토큰을 전달하는 메시지

3.3 *VTokenIC*의 특성

*VtokenIC*는 교차로에서 신호등 없이 자율주행 차량 사이의 통신만으로 교차로 내에서 차량의 이동 및 통과에 대한 제어를 하는 시스템을 설계하는 알고리즘이다. 여기서 핵심이 되어야 하는 내용은 교차로를 안전하면서도 효율적으로 통과하는 문제를 자율주행 차량 사이의 메시지 교환을 통한 협업으로 해결하는 방법을 제시하는 것이다. 이러한 면에서 *VtokenIC*는 다음의 두 가지 특성을 가진다.

(1) *VtokenIC*는 토큰 기반의 그룹 상호배제 알고리즘이다.

*VtokenIC*는 교차로에서의 자율주행 차량의 교통통제에 토큰기반의 그룹 상호배제 알고리즘을 적용한다. 또한 이는 교차로에서 중앙통제 장치인 신호등을 제거하고, 차량 간 통신을 기반으로 한 협업으로 원활한 교통흐름 조성과 사고 없는 안전한 운행을 목적으로 한다. 이를 위해서는 교차구역 진입에 대한 권한 관리가 필요하며, 이에 대한 문제를 의사소통 도구인 메시지 형태의 토큰을 이용하여 해결하는 것이다. 여기서 언급한 토큰이란 특별한 권한을 갖는 메시지를 의미하는 것으로서, 어떠한 하나의 토큰을 보유하면 교차구역을 통과할 수 있는 자격을 갖추는 것을 말한다.

*VtokenIC*는 섹션 3.2에서 기술한 그룹 상호배제 알고리즘이 가져야 할 속성인 안전성과 동시진행성 특성을 유지하도록 설계되어야 한다. 그러나 여기서 원활한 교통흐름과 안전한 소통은 서로 상충하는 개념으로서, 교통의 원활한 흐름을 향상하기 위해서는 안정성에 위험 요소가 증가할 수 있고, 안정성에 비중을 증가시키면 원활한 교통흐름에 방해 될 수 있다. 이러한 두 개의 상충된 속성을 동시에 만족시키는 프로토콜을 작성하는데 있어서는 고려해야 할 많은 요소들이 존재한다. 이를 해결하기 위하여 *VtokenIC*에서는 하나의 토큰을 운영하는 시스템을 확장하여, 주 토큰 이외에 부 토큰의 보조수단을 사용하여 동시진행성을 증가시킨다(Neeraj Mittal, Prajwal K. Mohan, 2007과 Hirotsugu Kakugawa, Sayaka Kamei, Toshimitsu Masuzawa, 2008).

이러한 두 개의 토큰은 각각 맡은 역할이 다른 바, 주 토큰은 교차로의 어떤 방향을 독점적으로 진행할지를 결정한다. 또한, 주 토큰은 교차구역의 하나의 세션을 관리하는 리더에게 주어지며, 주 토큰을 받은 주 토큰 홀더

는 교차구역에 진입할 수 있다. 주 토큰은 세션의 변화에 따라 지속적으로 순환하지만, 부 토큰은 하나의 세션에 사용되는 일시적 권한이다. 또한 이는 교차로 통과의 효율성 증가를 위하여 사용하는 보조 수단으로서, 주 토큰을 보유한 주 토큰 홀더에 의해 부여된다. 부 토큰 홀더도 세션의 리더인 주 토큰 홀더와 동시에 교차구역에 진입할 수 있으며, 이를 통하여 교차구역을 동시에 진입할 수 있는 차량의 수를 조절한다. 따라서 *VtokenIC*에서는 주 토큰을 효율적으로 순환하게 하는 것과 부 토큰을 체계적으로 생성하고 조정하는 것이 핵심 내용이 된다.

(2) *VtokenIC*는 세션의 흐름과 AV_i의 이동에 대한 알고리즘이다.

*VtokenIC*는 전체적인 차원에서 볼 때에 주 토큰의 순환으로 이루어지며, 주 토큰 홀더가 리더로서 참여하는 세션의 연속적인 흐름으로 볼 수 있다. 여기서 세션은 주 토큰 홀더가 하나의 세션에 진입함으로써 시작이 되며, 주 토큰 홀더와 동일한 그룹에 속하는 모든 차량들이 교차구역에 함께 진입한 후 그 모든 차량이 교차구역을 퇴장하는 시간단위를 말한다. 세션에 대한 개념을 집합으로 다음과 같이 표현할 수 있다(관련 표기법, <표 3.6>).

<표 3.6> 세션 표기법

표 기	내 용
$session_i$	AV_i가 참여하는 현재 시작되는 세션
pt_i-holder	$session_i$의 주 토큰 홀더
pt_i.holder-chain	주 토큰 홀더가 형성한 체인
pt_i.holder-ch.mem	주 토큰 홀더가 형성한 체인의 멤버
st_i-holder	$session_i$의 부토큰 홀더
st_i.holder-chain	부 토큰 홀더가 형성한 체인
st_i.holder-ch.mem	주 토큰 홀더가 형성한 체인의 멤버

• $session_i$={pt_i-holder, pt_i.holder-ch.mem , st_i-holder, st_i.holder-ch.mem}

• $session_i$의 시작 :

{(pt_i-1)-holder, (pt_i-1).holder-ch.mem} ∪

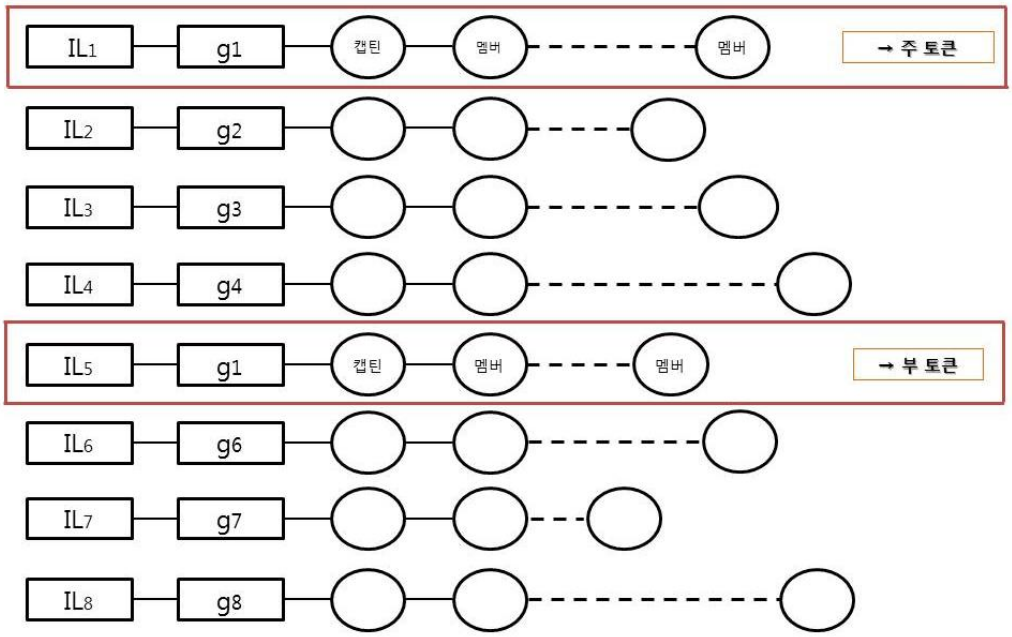
{(st_i-1)-holder, (st_i-1).holder-ch.mem} = {}

• $session_i$의 종료 → $session_i$+1의 시작

{pt_i-holder, pt_i.holder-ch.mem} ∪

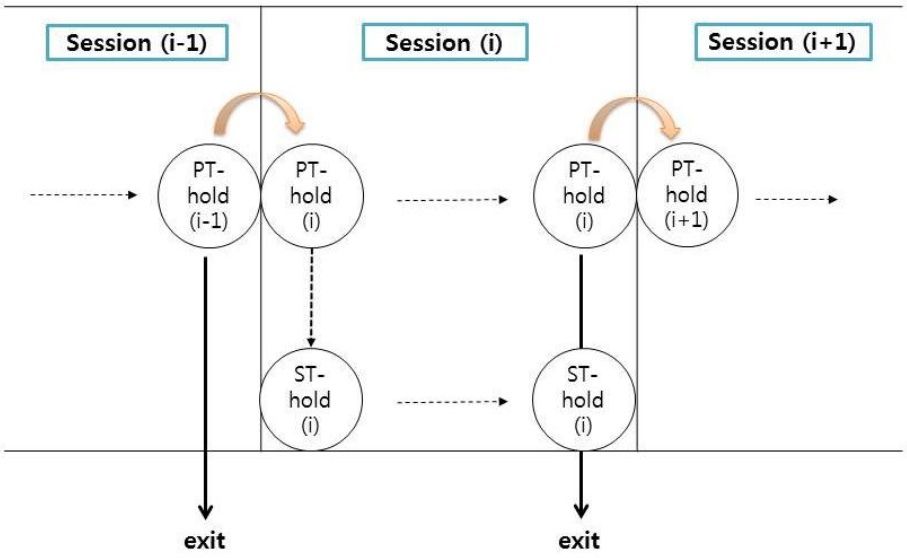
{st_i-holder, st_i.holder-ch.mem} = {}

즉, 새로운 주 토큰을 전달받은 주 토큰 홀더 AV_i는 자신이 위치한 차선(예를 들어 IL_1으로 가정)과 동일한 그룹에 해당하는 차선 IL_5의 캡틴 차량에게 부 토큰을 부여한다. 또한 주 토큰 홀더와 부 토큰 홀더는 교차구역 통과를 자신들의 체인멤버들과 함께 체인 단위로 하게 된다. 이처럼 하나의 세션($session_i$)은 <그림 3.6>에서 나타난 바와 같이 주 토큰 홀더 AV_i와 동일한 그룹에 속하는 모든 차량 즉, IL_1와 IL_5에 위치한 *w-CZ* 상태의 모든 차량들로 구성된다.



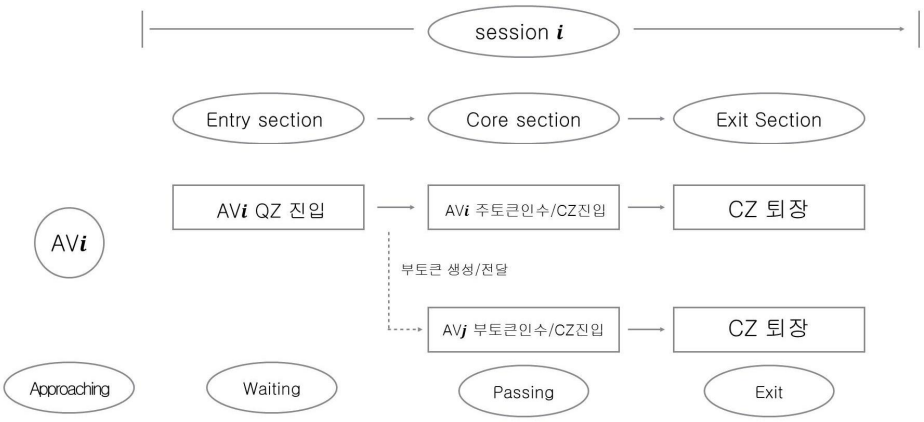
<그림 3.6> 세션의 구성

또한 AV_i는 교차구역을 통과하여 퇴장할 때 주 토큰을 넥스트 홀더에게 전달함으로써 새로운 다른 하나의 세션이 시작된다. 이처럼 *VTokenIC*는 교차로에서의 주 토큰이 순환되면서 주 토큰을 보유하는 주 토큰 홀더가 변경되고, 이에 따라 교차구역에 진입하는 차량 그룹이 변경되는 연속적인 세션의 흐름에 대한 알고리즘<그림 3.7>이라 할 수 있다.



<그림 3.7> 세션의 흐름

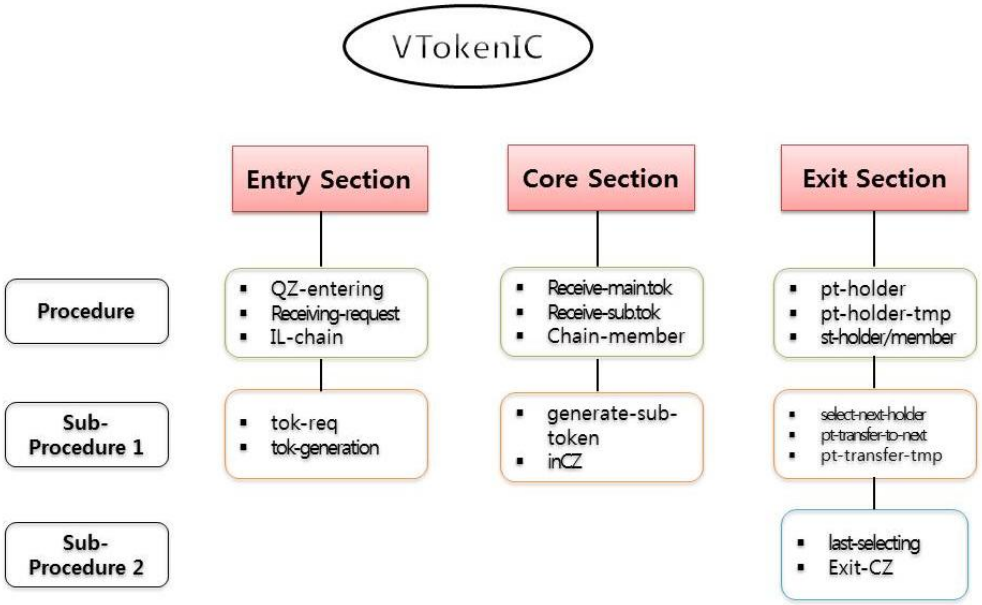
이와 더불어, 하나의 세션을 별도로 AV_i의 입장에서 보면 교차로에서의 이동이 다음 3 단계로 나누어져 진행이 된다. AV_i는 교차로에 진입하기를 원하면 대기구역에 진입하여야 하며(1단계: *Entry section*), 다음으로 대기구역 진입차선에서 대기 후 교차구역에 진입하여 통과하는 과정(2단계; *Core section*)을 거쳐, 마지막으로 교차구역을 퇴장하는 과정(3단계: *Exit section*)을 거치게 된다. 이러한 면에서 *VTokenIC*는 AV_i가 하나의 세션을 기준으로 해서 교차로를 진입하여 통과한 후 퇴장하여 나가는 이동과 통제 과정에 대한 알고리즘<그림 3.8>이라 할 수 있다.



<그림 3.8> AV의 흐름

3.4 *VTokenIC*의 프로토콜

*VTokenIC*의 정형적인 디스크립션(formal description)은 프로토 타입의 시제 논리 언어(temporal logical language)로 작성된다. 또한 각각의 프러시저와 서브 프러시저는 전송받은 메시지 형태와 외부 정보에 근거하여 연속적으로 작동된다. 또한, 이는 AV_i가 하나의 세션을 기준으로 해서 교차로를 진입하여 통과해 나가는 이동과 통제 과정에 대하여 <그림 3.9>와 같은 계층구조로 구성된다.



<그림 3.9> 알고리즘의 프레임워크

3.4.1 내부 변수

본 알고리즘 내의 각 AV_i는 다음의 변수를 사용하며, 아를 요약하면 <그림 3.10>과 같다.

- $session_i$: AV_i가 알고 있는 가장 최근의 세션
- *my-state* : AV_i의 현재 상태로서, 이의 값은 *approaching, w-move, w-CZ, passing*이다.
- *my-position* : AV_i가 속해있는 진입차선 체인의 구성을 나타내며, 이의 값은 $ILch_i$*-captain,* $ILch_i$*-member,* $ILch_i$*-tail* 이다.
- *my-token* : AV_i가 토큰을 보유하고 있는 형태를 나타내며, 이의 값은 pt_i*-hold,* st_i*-hold,* pt_i*-hold-tmp*이다.
- ts_i : 각 *request*의 타임스탬프 값으로, AV_i가 새로운 요청을 할 때마

다 하나씩 증가한다.

- *release-tail:* 진입차선 체인의 마지막 차량으로부터 받은 <*release*>의 수로서 <*release*>를 받을 때마다 하나씩 증가한다.
- $tok_i.type$: AV_i가 보유하고 있는 토큰의 종류이며, 이의 값은 *prime*과 *sub* 이다. 만약 토큰을 보유하지 않으면 그의 값은 ⊥이다.
- $tok_i.safe$: AV_i가 주 토큰을 전달 받은 때에 CZ의 진입에 대한 안정성이다. 이의 값은 T,F로 표시되며, *IC-AV*=⊥이면 $tok_i.safe$=T*이다.*
- $IL_i.req$: AV_i가 속해 있는 진입차선 체인의 미결요청 리스트 이며, 이는 $IL_1.req$부터 $IL_8.req$ 까지 8개가 존재한다.
- *IC-AV* : 교차로 전체 구역에 차량들이 존재하는지에 대하여 지피에스로부터 신호를 받는 변수이다. 값이⊥이면, 교차로 전체 구역 내에 차량이 없는 것을 나타낸다.
- *CZ-AV* : 교차구역 내부에 차량이 존재하는지에 대하여 지피에스로부터 신호를 받는 변수이다. 값이⊥이면, 교차구역 내에 차량이 없는 것을 나타낸다.
- *grid rule* : 교차로 내에서 AV의 그리드 이동원칙으로서, 직전 그리드가 비워져 있으면(*off* 상태), 그리드를 하나 이동한다.
- *token.sequence based on* IL *no*: 주 토큰이 순환되는 차선의 순서이다.

  $IL_1 \rightarrow IL_2 \rightarrow IL_3 \rightarrow IL_4 \rightarrow IL_5 \rightarrow IL_6 \rightarrow IL_7 \rightarrow IL_8 \rightarrow IL_1$
- *CRA*: 주 토큰 홀더와 동시에 CZ에 진입하는 차선의 캡틴이다.

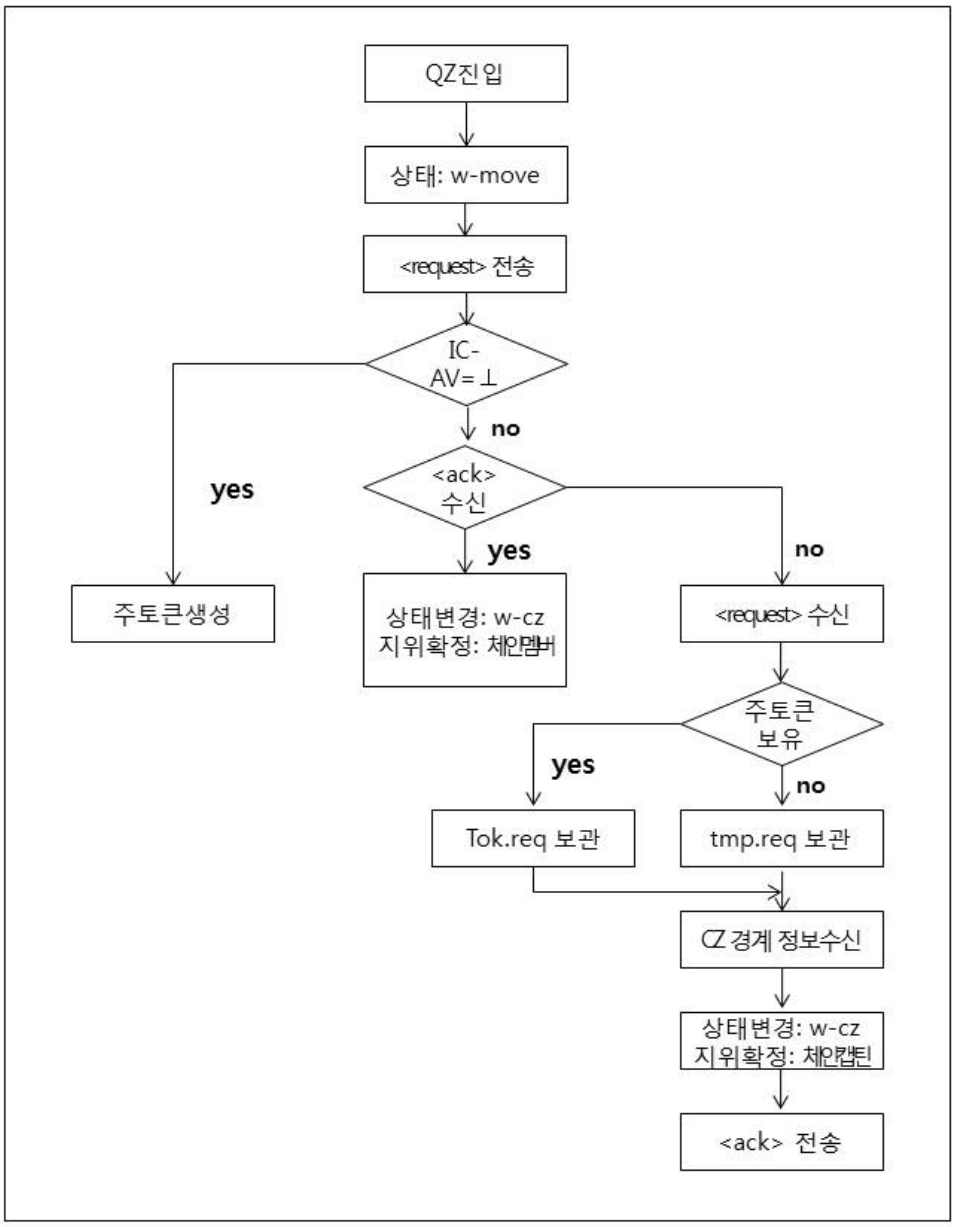
Variables

$session_i$: present session that AV_i leads
my-state={*approaching, w-move, w-CZ, passing*}
my-position={$ILch_i$-*captain*, $ILch_i$-*member*, $ILch_i$-*tail*}
my-token={pt_i-*hold*, st_i-*hold*, pt_i-*hold-tmp*}
IL={IL_1, IL_2, IL_3, IL_4, IL_5, IL_6, IL_7, IL_8}
IL_i: *IL* of AV_i ($IL_i \in IL$)
IL-tail
IL-captain
ts_i: timestamp value for each request
release-tail: no of <*release*> from *Ilch-tail*
tok_i.*safe*={T, F}
$token_i$.*type*={*prime, sub*};
IL.req={IL_1.*req*, IL_2.*req*, … , IL_8.*req*}
IC-AV : AV in IC
CZ-AV : AV in CZ, if *CZ-AV*=⊥ then tok_i.*safe*=T
grid rule : move one grid if front grid is empty
CRA: concurrent AV

<그림 3.10> 내부 변수

3.4.2 *Entry Section*

*Entry section*은 AV_i가 교차로를 통과하기를 원하여 교차로 대기구역에 진입하여 대기하는 단계이며, 이 *Entry section*에서의 AV_i의 진행 과정을 도해하면 <그림 3.11>과 같다.



<그림 3.11> *Entry section*의 플로우 차트

또한 *Entry section*은 <그림 3.12>의 3개의 프러시저(*QZ-entering, request, IL-chain)*와 <그림3.13>의 2개의 서브 프러시저 (*tok.req-enqueuing, tok-generation)*로 이루어진다.

Entry Section - procedure

```
procedure QZ-entering
 1.1   on receiving QZ-enter line signal from sensor;
 1.2   my-state=w-move;
 1.3   if IC-AV ≠ ⊥;
 1.4      then ts_i=ts_i+1;
 1.5           send <request, AV_i,ts_i,IL_i> to ∀AV;
 1.6      else call procedure tok-generation;
 1.7   end if;

procedure receiving-request
 2.1   on  receiving <request, AV_j,ts,IL>
 2.2   if (IL_j=IL_i) ∧ (my-position=ILch_i-captain);
 2.3      then send <ack> to AV_j;
 2.4           call procedure tok-req;
 2.5      else call procedure tok-req;
 2.6   end if;

procedure IL-chain
 3.1   on receiving <ack> from AV_j
 3.2   my-state=w-CZ;
 3.3   my-position=ILch_i.-member;
 3.4   move QZ by grid rule;
 3.5   wait for CZ entering;

 4.1   on receiving signal of CZ-enter-line from sensor:
 4.2   if my-state=w-move;
 4.3     then my-position=ILch_i.-captain;
 4.4          my-state=w-CZ;
 4.5          send <ack> ∀AV in IL_i;
 4.6   end if;
```

<그림 3.12> *Entry Section* - 프러시저

Entry Section - sub procedure

```
procedure tok.req-enqueuing
 5.1   if my-token=pt_i-hold;
 5.2     then enqueue (tok_i,reqQ,(AV_j,ts,IL));
 5.3     else ignore <request, AV_j,ts,IL>;
 5.4   end if;

procedure tok-generation
 6.1     then generate tok,type=prime;
 6.2          initialize variables;
 6.3     else wait for CZ;
 6.4   end if;
```

<그림 3.13> *Entry Section* - 서브 프러시저

(1) 대기구역 진입(1.1 - 1.6)

교차로 통과를 원하는 AV_i는 센서로부터 대기구역 진입 경계선 정보를 받으면 대기구역에 진입한다. 이때 AV_i는 *w-move* 상태로 변경되며, 타임스탬프는 하나 증가한다(1.1-1.2). 이 때 AV_i는 차량에 부착된 지피에스로 교차로 내에 차량이 존재하는지를 확인하여야 한다(1.3-1.4). 교차로 내에 차량이 존재하면, 대기구역에 진입한 AV_i는 교차구역의 통과를 위한 요청 메시지를 교차로 내의 다른 모든 AV들에게 브로드캐스트 한다(1.5). 그러나 이와는 달리 교차로 내에 차량이 없는 경우에는, AV_i는 주 토큰을 생성하여야 한다(1.6-1.7, 6.1-6.4).

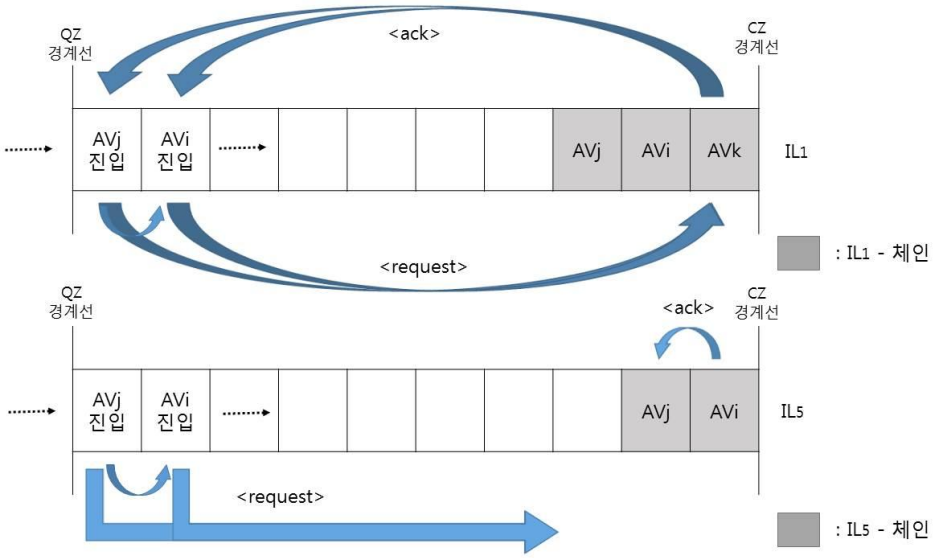
(2) 요청의 수신(2.1-2.4)

교차로 내에 있는 AV_i는 대기구역에 진입한 AV_j의 요청 메시지를 받은 때에는 자신이 AV_j가 진입한 차선의 캡틴이면, <ack> 메시지를 AV_j에게 보낸다(2.1-2.3). 또한, AV_i가 주 토큰 홀더인 경우에는 AV_j의 요청을 tok_i,*reqQ*에 등재한다. 이 이외의 모든 경우에 AV_i는 AV_j의 요청을 무시한다(5.1-5.4).

(3) 진입차선 체인의 구성(3.1-4.7)

대기구역에 진입한 AV_i는 자신의 진입차선 캡틴 AV_j로부터 <ack> 메시지를 받으면, *w-move*에서 *w-CZ*로 상태가 변하고(3.2), 자신이 위치한 진입차선(IL_i)의 체인 멤버가 된다(3.3). AV_i는 진입차선(IL_i)에 진입 후 그리드 이동원칙에 따라 자신의 그리드로 이동하여 대기한다(3.4-3.5).

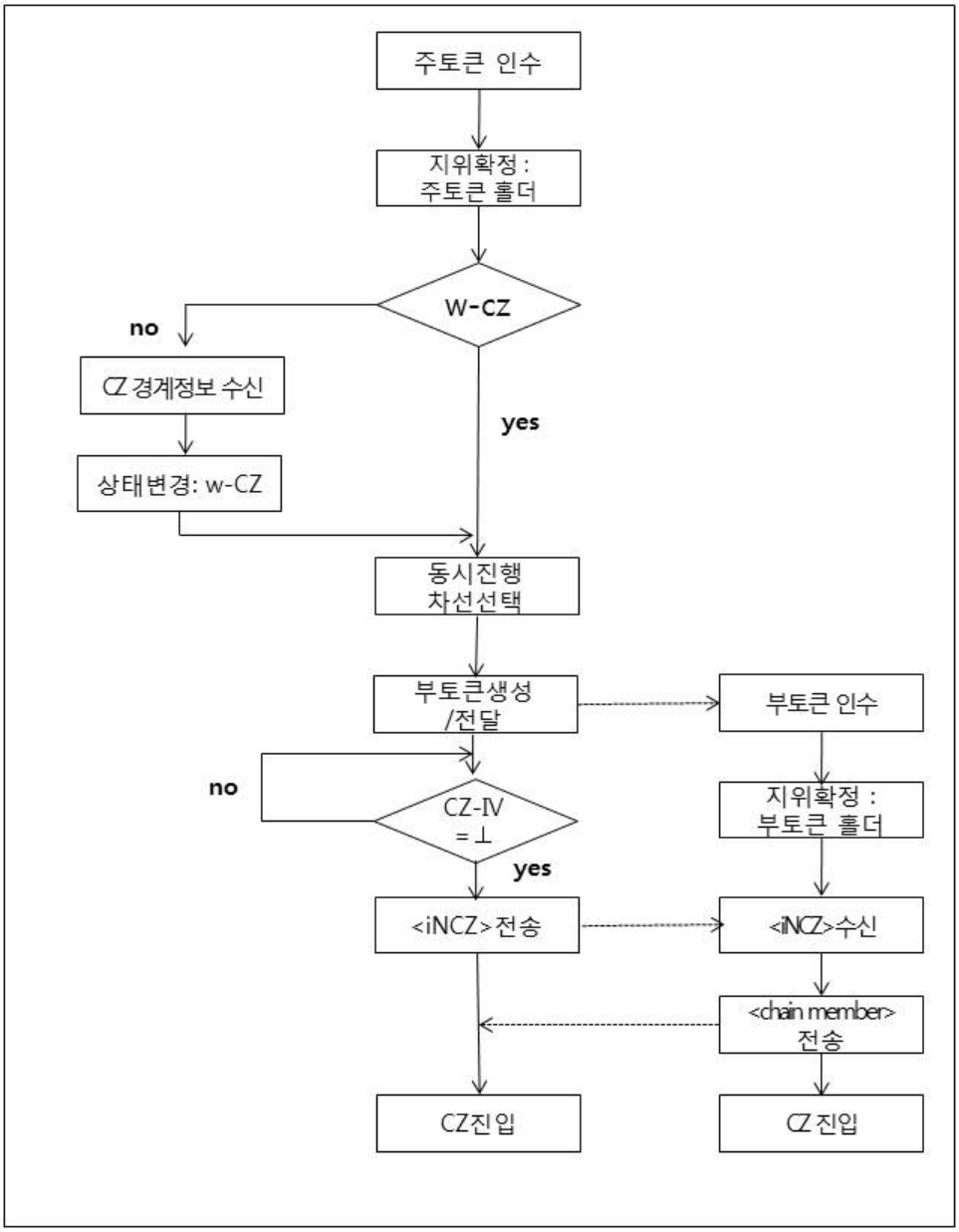
그러나 AV_i가 대기구역에 진입하여 <ack> 메시지를 받지 못한 때에는 첫째, 진입차선(IL_i) 내에 자신보다 먼저 온 AV_j가 있는 경우에는 그 직전 그리드까지 이동하여 <*ack*> 메시지를 기다린다. 두 번째, 자신의 앞에 다른 차량이 없는 경우에는, AV_i는 센서로부터 교차구역 진입 경계선 정보를 받은 후에 진입차선(IL_i) 체인의 캡틴이 된다. 이 때 AV_i의 상태는 *w-move*에서 *w-CZ*로 변경되며, 진입차선(IL_i)에 있는 모든 차량에게 <ack> 메시지를 보낸다(4.1-4.7). 이러한 진입차선 체인에 대한 예를 도해하면 <그림 3.14>와 같다.



<그림 3.14> IL 체인의 구성

3.5.3 *Core Section*

이 섹션은 본 알고리즘 중 대기구역의 진입차선에서 대기 중인 AV_i가 교차구역으로 진입하는 부분이다. 주 토큰은 교차로의 어떤 방향으로 독점적으로 진행할지를 결정하며, 주 토큰을 보유한 주 토큰 홀더는 교차구역에 진입할 수 있다. 또한 부 토큰은 교차로 통과의 효율성 증가를 위하여 사용하는 보조 수단으로서 주 토큰 홀더에 의하여 부여된다. 이처럼 부 토큰을 보유한 부 토큰 홀더도 주 토큰 홀더와 함께 교차구역에 진입할 수 있다. 주 토큰 홀더와 부 토큰 홀더는 교차구역에 진입할 때 자신들의 각 진입차선 체인의 멤버들을 모두 연결하여 같이 진입한다. 이러한 AV_i의 교차로 진입 및 통과 과정을 도해하면 <그림 3.15>와 같다.



<그림 3.15> *Core section*의 플로우 차트

또한, *Core section*는 <그림 3.16>의 3개 프러시저(*receiving-main-token, receiving-sub-token, member-inCZ)*와 <그림 3.17>의 2개 서브 프로시저 (*generating-sub.token, inCZ)*로 구성된다.

Core Section-procedure

procedure *receiving-main.token*
(PC) *my-position=ILch*$_i$*-captain*;
7.1 **on** receiving <*send-pt*> from AV$_j$;
7.2 *my-token=pt*$_i$*-hold*;
7.3 call **procedure** *generating-sub.token*
7.4 call **procedure** *inCZ*;
7.5 **end if;**

procedure *receiving-sub-token*
(PC) *my-position=ILch*$_i$*-captain*;
8.1 **on** receiving <*send-st*> from AV$_j$
8.2 *my-token=st*$_i$*-hold*;
8.3 send <*st-chainmember*> to AV$_j$;
8.4 *on receiving* <*iNCZ*> *from* AV$_j$;
8.5 call **procedure** *inCZ*;

procdure *member-inCZ*
(PC) *my-position=ILch*$_i$*-member*;
9.1 **on** receiving signal of CZ-enter line from sensor;
9.2 **if** *my-state=w-CZ;*
9.3 **then** call **procedure** *inCZ*;
9.4 **end if**;

<그림 3.16> *Core Section*-프러시저

Core Section-sub procedure

```
procedure generating-sub.token
 10.1   select ILch(i+4)-captain as CRA;
 10.2   if IL(i+4).req( ∈IL.req)≠⊥;
 10.3       then generate tok_i.type=sub;
 10.4            send <send-st> to ILch(i+4)-captain;
 10.5   end if;

procedure inCZ
 11.1   if my-token=pt_i-hold;
 11.2       then if tok_i.safe=T;
 11.3            then move in CZ by grid rule;
 11.4                 my-state=passing;
 11.5                 send <iNCZ> to ILch(i+4)-captain;
 11.6            else wait;
 11.7       else  move in CZ by grid rule;
 11.8            my-state=passing;
 11.9   end if;
```

<그림 3.17> *Core Section*-서브 프러시저

(1) 주 토큰의 인수(7.1-7.6)

AV_i는 주 토큰을 *w-CZ* 상태일 때 전달받으면 바로 주 토큰 홀더의 지위가 되어 교차구역 진입 절차를 시작한다. 그러나 AVi가 *w-move* 상태에서 주 토큰을 전달받는 경우에는, AV_i는 진입차선(IL_i)의 캡틴이 되어 *w-CZ* 상태일 때 새로운 세션의 주 토큰 홀더가 된다(7.1-7.2). 그 이후 AV_i는 동시진행 그룹을 선택하여 그들과 함께 교차구역에 진입한다(7.3-7.4).

(2) 부 토큰 생성 및 전달(10.1-10.5)

주 토큰 홀더인 AV_i는 표 3.4에 제시된 기준에 의하여 동시진행 차선을 선택하며, 그 선택된 차선에 차량이 있으면(10.1-10.2), 그 차선의 캡틴에게 부 토큰을 생성하여 전달한다(10.3-10.5). 만약, 선택된 차선에 차량이 없으면 부 토큰을 생성하지 않는다.

(3) 교차구역 진입(11.1-11.9)

AV_i는 주 토큰을 전달받은 때에 그 주 토큰을 사용하여 교차구역에 진입하는 것이 안전하지 않을 수도 있다. 이는 이전 세션에 해당되는 부 토큰 및 차량들이 교차구역을 통과하여 빠져 나가지 못하고 교차구역 내에서 이동 중일 수 있기 때문이다. 따라서 AV_i는 교차구역에 진입하기 전에 이전 세션의 차량들이 교차구역에서 모두 나갔는지를 확인하여야 한다. 본 알고리즘에서 이의 확인은 차량에 부착된 지피에스로 교차구역 내의 차량 존재 여부를 확인한다(11.1-11.2). AV_i는 교차구역 내에 차량이 없어서 진입하기에 안전하면 교차구역에 진입하며, 이때에 AV_i의 상태는 *passing*으로 변경된다(11.3-11.4). 이때에 AV_i는 자신의 교차구역 진입을 알리기 위하여 부 토큰 홀더 AV_j에게 <*iNCZ*>메시지를 보낸다(11.5).

(4) 부 토큰의 인수(8.1-8.5) 및 교차구역 진입(11.7-11.9)

AV_i는 대기구역에 진입하여 대기 중에 주 토큰 홀더 AV_j로부터 부 토큰을 전달 받는다(8.1). 이때 AV_i는 새로운 세션의 부 토큰 홀더가 된다(8.2). AV_i는 이처럼 부 토큰을 전달 받아 부 토큰 홀더가 된 때, 주 토큰 홀더

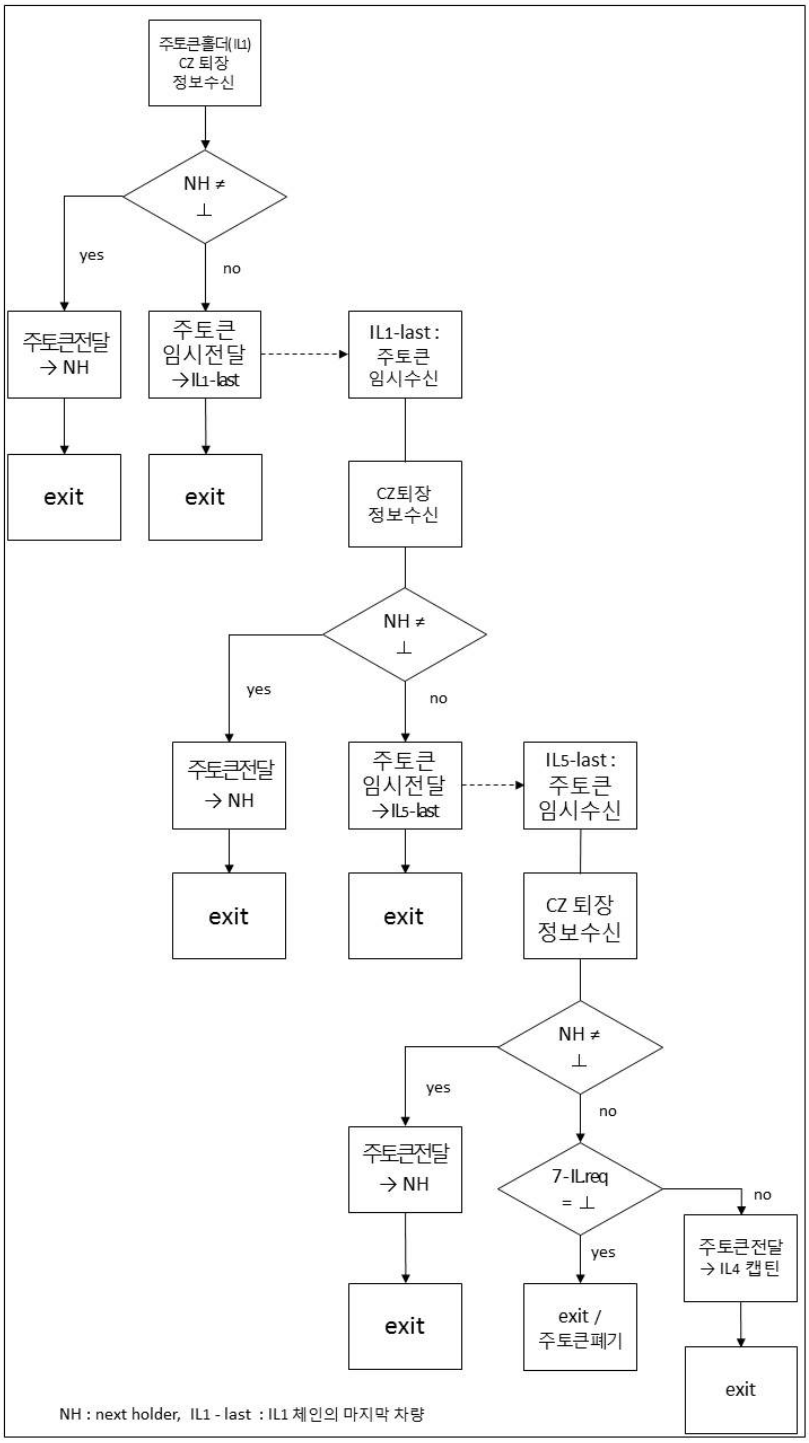
AV_j에게 <*chainmember*>메시지를 보내어 자신의 진입차선 체인의 멤버들을 알려준다(8.3). 부 토큰 홀더인 AV_i는 주 토큰 홀더로부터<*iNCZ*>를 받은 이후에 대기구역에 진입한다(8.4-8.5).

(5) 체인멤버의 교차구역 진입(11.1-11.4)

주 토큰 홀더나 부 토큰 홀더의 진입차선 체인 멤버인 AV_i는 자신의 캡틴인 주 토큰 홀더 또는 부 토큰 홀더가 교차구역에 진입하면, 그리드 이동원칙에 따라 이동하면 교차구역 진입 경계선에 접하게 된다(11.1). 이 때 AV_i의 상태가 *w-CZ*이면, 그리드 이동원칙에 의하여 교차구역에 진입하면 된다(11.2-11.4).

3.5.4 *Exit Section*

*Exit Section*은 본 알고리즘 중 AV_i가 교차구역 통과를 완료하여 퇴장하는 부분이다. 이는 주 토큰을 넥스트 홀더에게 전달하여야 하며, 필요한 경우 주 토큰을 임시로 전달하는 등 주 토큰의 순환을 효과적으로 하여 시스템의 성과를 증가시키는 중요한 과정이다. 이를 도해하면 <그림 3.18>과 같다.



<그림 3.18> *Exit section*의 플로우 차트

*Exit section*은 프러시저와 서브 프로시저로 구성된다. 프러시저는 <그림 3.19>에 나타난 바와 같이 다음 3 가지 프러시저로 구분된다. 이는 주 토큰 홀더인 경우 *pt-holder* , 임시 주 토큰 홀더인 경우 *pt-holder-tmp* , 기타 체인 멤버의 경우 *st-holder & member* 이다.

***Exit Section*-procedure**

procedure *pt-holder*
(PC) *my-token=pt_i-hold*;
12.1 **on** receiving signal of CZ-end line;
12.2 call **procedure** *selecting next-holder;*

procedure *pt-holder-tmp*
(PC) (my-position=$ILch_i$-tail ∨ $ILch_{(i+4)}$-tail)
13.1 ***on** receiving <send.ptch-tmp> from AV_j;*
13.2 *my-token=pt_i-hold-tmp*;
13.3 **on** receiving signal of CZ-end line;
13.4 call **procedure** *selecting-next-holder;*

14.1 **on** *receiving <send.stch-tmp> from AV;*
14.2 *my-token=pt_i-hold-tmp*;
14.3 **on** receiving signal of CZ-end line;
14.4 call **procedure** *selecting-next-holder;*

procedure *st-holder & member*
(PC) (my-token=st_i-hold ∨ ((*my-token*= ⊥)∧(my-state=passing))
15.1 **on** receiving signal of CZ-end line;
15.2 call **procedure** *exit-CZ;*

<그림 3.19> *Exit Section*-프러시저

또한, *Exit section*은 <그림 3.20>의 서브 프러시저 1과 <그림 3.21>의 서브 프러시저 2로 구분된다.

Exit Section-sub procedure 1

procedure *selecting-next-holder;*
16.1 select $ILch_{(i+1)}$-*captain as next-holder based on tok.sequence*;
16.2 grouping tok_i,*reqQ* based on no of *IL.req*;
16.3 **if** $IL_2.req(\in IL.req) \neq \perp$;
16.4 **then** call **procedure** *pt-transfer-to-next;*
16.5 **else** call **procedure** *pt-transfer-tmp;*
16.6 **end if;**

procedure *pt-transfer-to-next*
17.1 send <*send-pt*> to $ILch_{(i+1)}$-*captain*;
17.2 **if** *my-token*=pt_i-*hold*
17.3 **then** send <*next-inform*> to IL_i-*tail*, $IL_{(i+4)}$-*tail;*
17.4 call **procedure** *exit-CZ;*
17.5 **else** send <*next-inform*> to $IL_{(i+4)}$-*tail;*
17.6 call **procedure** *exit-CZ;*
17.7 **end if;**

procedure *pt-transfer-tmp*
18.1 **if** *my-token*=pt_i-*hold* ∧ *my-state=passing*;
18.2 **then** send <*send.ptch-tmp*> to IL_i-*tail*;
18.3 call **procedure** *exit-CZ;*
18.4 **else if** *my-position*=IL_i-*tail* ∧ *my-state=passing*;
18.5 **then** send <*send.stch-tmp*> to $IL_{(i+4)}$-*tail*;
18.6 call **procedure** *exit-CZ;*
18.7 **else** call **procedure** *last-selecting;*

<그림 3.20> *Exit Section*-서브 프러시저 1

Exit Section-sub procedure 2

```
procedure last-selecting;
 19.1   check 7-IL.req(IL(i+2), IL(i+3), IL(i+4), IL(i+5), IL(i+6), IL(i+7), IL_i)
        in turn;
 19.2   if 7-IL.req=⊥;
 19.3      then exit CZ by grid rule;
 19.4      else let IL(i+3).req=the first (IL_k.req ≠⊥) in 7-IL.req;
 19.5           then send <send-pt> to ILch(i+3)-captain;
 19.6                call procedure exit-CZ;
 19.7   end if;

prcedure exit-CZ
 20.1   if my-token=pt_i-hold;
 20.2     then dequeue ILch_i-member, ILch(i+4)-captain,
 20.3                  ILch(i+4)-member in tok_i,reqQ;
 20.4          exit CZ by grid rule;
 20.5     else if(my-token=pt_i-hold-tmp)∧(my-position=ILch_i-tai)l;
 20.6          then piggyback <release> on <send.stch-tmp>;
 20.7               exit CZ by grid rule;
 20.8          else if (my-position=ILch(i+4)-tail);
 20.9               then send <release> to AV_j;
 20.10                   exit CZ by grid rule;
 20.11              else exit CZ by grid rule;
 20.12  end if;
```

<그림 3.21> *Exit Section*-서브 프러시저 2

(1) 주 토큰 홀더의 교차구역 퇴장(12.1-12.2)

주 토큰 홀더인 AV_i는 교차구역 진입 후 통과를 완료하면, 교차구역 퇴장

라인의 경계선을 접하게 되어 센서로부터 경계선 정보를 받게 된다(12.1). 이때 AV_i는 교차구역을 퇴장하기 전에 다음 세션의 주 토큰 홀더에게 주 토큰을 전달하여야 한다(12.2).

주 토큰 홀더를 전달하는 절차는 다음과 같다(16.1-16.6). 첫째, AV_i는 <표 3.7>에 규정된 주 토큰 순환 기준에 의하여 다음 순서의 진입차선(만약 AV_i가 IL_1에 위치하면 다음 순서는 IL_2)의 캡틴을 넥스트 홀더로 선택한다(16.1). 여기서 주 토큰의 순환 기준은 현행 신호등의 직진 후 좌회전 체계를 적용하고, 8개 차선을 라운드어바웃(roundabout) 하는 방식을 적용한다.

<표 3.7> 주 토큰 순환

순서	주 토큰 홀더 차선	순서	주 토큰 홀더 차선
①	남→북 직진차선 (IL_1)	⑤	북→남 직진차선 (IL_5)
②	남→북 좌회전차선 (IL_2)	⑥	북→남 좌회전차선 (IL_6)
③	동→서 직진차선 (IL_3)	⑦	서→동 직진차선 (IL_7)
④	동→서 좌회전차선 (IL_4)	⑧	서→동 좌회전차선 (IL_8)

두 번째, AV_i는 자신이 보관하고 있는 미결요청들의 대기열($tok_i.reqQ$)을 진입차선별로 그룹핑 한다(16.2). 세 번째, AV_i는 다음 순서의 진입차선(IL_2) 내에 미결 요청이 있는 경우에는, 주 토큰을 다음 순서의 진입차선(IL_2) 캡틴에게 <*send-pt*> 메시지를 보내어 전달한다(16.3-16.4). 이 때 AV_i는 자신의 진입차선 체인의 마지막 차량과 부 토큰홀더 체인의 마지막 차량에게 넥스트 홀더를 알려준다(17.1-17.3). 또한 AV_i는 미결요청 대기열($tok_i.reqQ$)에서 자신의 진입차선 체인의 멤버, 부 토큰 홀더, 부 토큰 홀더 진입차선 체인의 멤버들을 삭제한 후 교차구역을 퇴장한다(20.1-20.4). 이와 다르게 만

약 다음 순서의 진입차선(IL_2) 내에 미결 요청이 없는 경우에는 AV_i는 자신의 진입차선(IL_1)의 마지막 차량에게 주 토큰을 임시로 전달한다(16.5, 18.1-18.3). 또한 AV_i는 미결요청 대기열($tok_i.reqQ$)에서 자신의 진입차선 체인의 멤버, 부 토큰 홀더, 부 토큰 홀더 진입차선 체인의 멤버들을 삭제한 후 교차구역을 퇴장한다(20.1-20.4).

(2) 임시 주 토큰 홀더의 교차구역 퇴장(13.1-14.4)

AV_i는 주 토큰 홀더 AV_j로부터 교차구역 내에서 주 토큰을 전달 받은 때에는 임시 주 토큰 홀더가 된다(13.1-13.2). 임시 주 토큰 홀더인 AV_i가 교차구역 통과를 완료하고 센서로부터 교차구역 퇴장라인 정보를 받는 때에는, AV_i는 넥스트 홀더의 진입차선(IL_2)의 미결요청을 확인하여야 한다(13.3-13.4). IL_2 내에 미결 요청이 있는 경우에는, 주 토큰을 진입차선 IL_2의 캡틴에게 <*send-pt*> 메시지를 보내어 전달한다. 이 때 AV_i는 부 토큰홀더 체인의 마지막 차량에게 결정된 넥스트 홀더를 알려준다(17.5-17.7). 이후 AV_i는 교차구역을 퇴장하며, <*release*> 메시지를 <*send-pt*> 메시지에 업어서 보낸다(20.5-20.7). 이와는 달리 만약 넥스트 홀더의 진입차선(IL_2) 내에 미결 요청이 없는 경우에는, 부 토큰 홀더 진입차선의 마지막 차량에게 주 토큰을 임시로 전달한다(16.5, 18.4-18.6). 이 이후 AV_i는 퇴장하며, <release> 메시지를 넥스트 홀더에게 보낸다.(20.5-20.7).

AV_i는 주 토큰 홀더 체인의 마지막 차량으로부터 주 토큰을 전달 받은 때에도 임시 주 토큰 홀더가 된다(14.1-14.2). 이처럼 임시 주 토큰 홀더인 AV_i가 교차구역 통과를 완료하여 센서로부터 교차구역 퇴장라인 정보를 받는 때에는, AV_i는 넥스트 홀더의 진입차선(IL_2) 내에 미결 요청이 있는지를

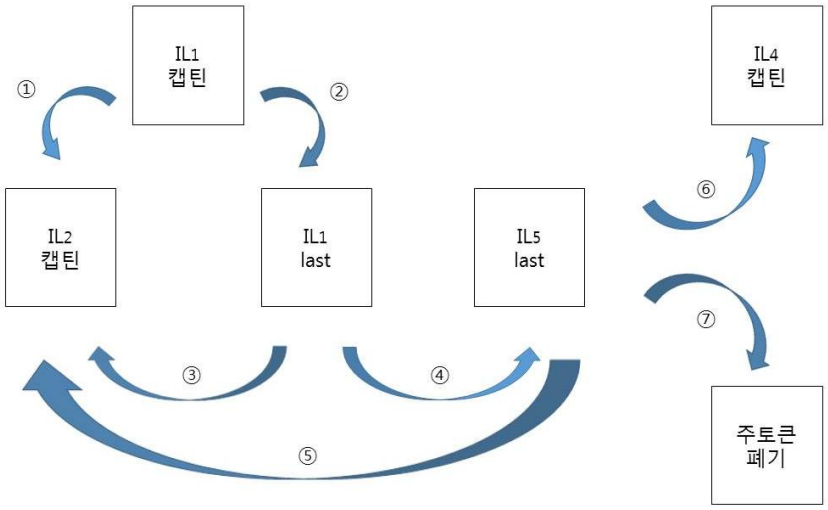
확인하여야 한다(14.3-14.4). 만약 IL_2에 미결요청이 있는 경우에는, AV_i는 주 토큰을 <*send-pt*> 메시지를 보내어 넥스트 홀더에게 전달한다(17.1). 이 이후 AV_i는 교차구역을 퇴장하며, <*release*> 메시지를 넥스트 홀더에게 보낸다(20.9). 만약 넥스트 홀더의 진입차선(IL_2) 내에 미결 요청이 없는 경우에는, 나머지 전체 7개 차선의 차선별 미결 요청의 상황(7-*IL.req*)을 다음과 같이 확인하여야 한다(19.1-19.7).

• 7-*IL.req* 확인 방법

이 경우는 임시 주 토큰 홀더인 AV_i는 부 토큰 홀더 체인의 마지막 차량 즉, 교차구역 내의 마지막 차량이므로 넥스트 홀더 차선인 IL_2에 차량이 없는 경우에는 주 토큰을 임시로 전달할 수 없게 된다. 따라서 이때 AV_i는 나머지 7개 차선의 미결계정을 확인하여, 미결계정이 없는 경우 즉 교차로에 차량이 없는 경우에는 주 토큰을 폐기하고 교차구역을 나가야 한다.

그러나 만약 나머지 7개 차선에 차량이 있는 경우에는 AV_i는 보관중인 주 토큰을 전달할 차선을 결정하여야 한다. 이는 상기한 예로서 먼저 표 3.5의 주 토큰 순환 순서를 적용하면, 나머지 7개 차선의 순서는 IL_3, IL_4, IL_5, IL_6, IL_7, IL_8, IL_1 이 된다. 그리고 이 순서를 적용하되 해당 차선에 미결요청이 없으면 다음 차선으로 순서가 넘어 간다.

예를 들어 IL_3에 미결요청이 없고, IL_4가 상기 7개 차선 중 첫 번째로 미결 요청이 있는 차선이라 가정하면, AV_i는 IL_4의 캡틴을 넥스트 홀더로 선택하고 주 토큰을 전달한다. 이때 AV_i는 <release> 메시지를 업어서 보내고, 교차구역을 퇴장한다(20.2-20.7). 이러한 주 토큰의 순환 과정을 그림으로 요약하여 도해하면 <그림 3.22>와 같다.



<그림 3.22> 주 토큰의 순환

(3) 진입차선의 체인 멤버의 교차구역 퇴장

교차구역 내에 있는 부 토큰 홀더 또는 토큰 홀더들의 진입차선 체인멤버인 AV_i는 교차구역 퇴장라인에 접했을 때에는 바로 그리드 원칙에 따라 교차구역을 퇴장한다(15.1-15.2, 20.11).

IV. 실험 및 결과분석

*VTokenIC*는 자율주행 자동차의 교차로 교통통제 시스템에 상호배제 이론을 적용한 알고리즘이다. 따라서 본 알고리즘의 분석은 컴퓨터 공학의 그룹 상호배제 문제에 대한 효율성 분석과 교통의 효율성 측면에서의 결과 분석 두 가지로 실시한다. 본 섹션에서는 먼저 그룹 상호배제 알고리즘들의 일반적인 평가항목인 메시지 복잡도, 동기화 지연, 동시진행도를 분석한다. 다음으로 교차로에서의 교통의 효율성 측면에서의 평가는 주어진 시간 내의 교통 처리량을 기초로 프로그램 작성을 통한 시뮬레이션 결과를 분석한다. 본 알고리즘에서는 상기한 분석을 위하여 다음의 평가항목이 사용된다.

- 메시지 복잡도(message complexity) : AV_i가 교차로에 진입하여 교차구역을 통과하여 나갈 때 까지 소요되는 메시지의 수
- 동기화 지연(synchronization delay) : 현재 주 토큰 홀더인 AV_i가 교차구역을 나온 이후와 다음 주 토큰 홀더가 교차구역에 진입하는 사이에 요구되는 시간이다.
- 동시진행도(concurrency) : 하나의 세션에 최대한 동시에 참여할 수 있는 차량들의 수
- 시스템 처리량 (system throughput) : 매분 교차로를 통과할 수 있는 차량들의 수
- 대기 시간(waiting time) : AV_i가 대기구역에 진입하여 교차구역에 진입까지의 대기 시간
- 대기길이(queue length) : 진입차선에서의 평균 차량의 대기 길이

4.1 *VTokenIC*의 상호배제 알고리즘 평가항목

*VTokenIC*의 그룹 상호배제 알고리즘의 평가항목은 메시지 복잡도, 동기화 지연, 동시진행도 세 가지 이며, 레퍼런스 알고리즘들과의 비교는 <표 4.1> 과 같으며, 이의 항목별 상세한 내용은 다음과 같다.

4.1.1 메시지 복잡도

*VTokenIC*에서는 8 종류의 메시지 <*request*>, <ack>, <send-pt>, <*iNCZ*>, <send-st>, <*chainmember*>, <*send-ptch-tmp*>, <*send-stch-tmp*>를 사용한다. AV_i가 교차로에 진입하여 교차구역을 통과하여 나갈 때까지 메시지가 가장 많이 교환되는 최악의 경우에 메시지 복잡도는 *1.125n+5*로서 내역은 다음과 같다.

(1) AV_i가 진입차선(IL_1으로 가정) 진입 시 <*request*>메시지 브로드캐스트: *(n-1)*

여기서 *n*은 교차로 내에 있는 차량의 수이다. 교차로 내의 차량의 수는 교통 흐름의 특성상 동적 집합이지만 이를 주 토큰 홀더가 주 토큰을 전달받은 시점에 컷오프 하여 교차로 내에 있는 차량의 수를 *n*으로 정한다. AV_i는 IL_1에 진입 시 교차로 내에 있는 자신을 제외한 모든 *(n-1)*의 차량에게 메시지를 브로드캐스트 해야 한다. 이처럼 메시지를 브로드캐스트 할 때에 가장 많은 메시지 트래픽이 발생하게 된다.

(2) AV_i가 IL_i-체인 캡틴이 되어 팔로우어들에게 <*ack*> 송신: *0.125n(=n/8)*

AV_i는 진입차선(IL_1)에 진입 할 때 IL_1에 차량이 없는 경우에는 IL_1-체인의 캡틴이 되며, AV_i는 이처럼 IL_1-체인의 캡틴이 된 후에는 자신의 차선

IL_1로 진입한 모든 차량에게 <*ack*>메시지를 보내야 한다. 이 때 만약 8개의 각 진입차선에 동일한 수의 차량이 진입하는 것으로 가정하면, AV_i는 *n/8(=0.125n)*의 <*ack*>메시지를 보내게 된다.

(3) AV_i가 주토큰 홀더가 되어 교차구역 진입 시 : *3*

AV_i는 주 토큰 홀더가 되면 자신과 동시에 교차구역에 진입할 차선 IL_5의 캡틴에게 <*send-st*> 메시지를 보내어 부 토큰을 전달한다. 이 때 부 토큰을 전달받은 AV_j는 IL_5-체인의 멤버 리스트를 AV_i에게 알리기 위하여 <*chainmember*> 메시지로 전송한다. 또한 주 토큰 홀더 AV_i는 교차구역에 진입 시 이를 부 토큰 홀더에게 알리기 위하여 <*iNCZ*> 메시지를 전송한다. 이처럼 AV_i가 교차구역에 진입할 때 3개의 메시지 교환이 필요하다.

(4) 주 토큰 홀더 AV_i가 교차구역을 퇴장할 때 토큰 전달 : *3*

AV_i가 교차구역 퇴장할 때에 넥스트 홀더가 없는 경우에는 자신의 차선 IL_1-체인의 마지막 차량에게 <*send-ptch-tmp*>를 보내어 주 토큰을 임시 전달한다. 또한 임시 주 토큰 홀더가 교차로를 퇴장할 때에도 넥스트 홀더가 없으면 부 토큰 홀더의 IL_5-체인의 마지막 차량에게 <*send-stch-tmp*>를 보내어 주 토큰을 임시 전달한다. 임시 주 토큰 홀더가 교차로를 퇴장할 때 넥스트 홀더에게 <*send-pt*>메시지를 보내어 주 토큰을 전달한다. 이처럼 AV_i가 교차구역을 퇴장할 때에는 최대한 3개의 메시지 교환이 필요하다.

따라서 AV_i가 교차구역에 진입하고 통과를 완료하여 퇴장하기까지의 최대한 메시지 복잡도는 상기한 (1)부터 (4)까지의 메시지의 수를 더한 *1.125n+5*가 된다.

그러나 이와 반면에 AV_i가 진입차선(IL_1)에 진입하여 IL_1-체인의 멤버가 되는 경우에는 *n*개의 메시지 교환만 필요하다. 이때에는 AV_i가 진입차선 IL_1

에 진입하면서 *n-1*개의 <*request*> 메시지 브로드캐스트 하고, AV_i가 IL_1의 캡틴 AV_j로부터 한 개의 <*ack*> 메시지를 받아 IL_1-체인의 멤버가 된 이후에는 체인 캡틴을 따라서 연결되어 교차구역에 진입하고 퇴장하게 되므로 메시지 교환이 필요 없게 된다. 따라서 AV_i가 진입차선 체인 멤버로서 교차구역에 진입하고 퇴장하는 최소한의 메시지 복잡도는 *n*이다.

섹션 2에서의 Mamun-Nagazato의 알고리즘의 메시지 복잡도는 *n+2*이며, Mittal-Mohan의 알고리즘의 메시지 복잡도는 *2n-1* 이다. 또한 자율주행 자동차의 교차로 교통통제에 상호배제 이론을 적용한 Weigang의 알고리즘은 AV_i가 모든 차량에게 메시지 브로드 캐스트를 최소한 3번 하여야 하므로 메시지 트래픽은 3n 이상 수준이다. 본 알고리즘의 메시지 복잡도는 교차로에서의 동적인 교통처리를 위한 특수성을 감안할 때 작은 수준이다.

4.1.2 동기화 지연

그룹 상호배제 알고리즘에서 동기화 지연은 두 개의 연속된 세션 사이의 시간으로 정의된다. 본 알고리즘의 경우 동기화 지연은 이전 세션의 마지막 주 토큰 홀더가 교차구역을 퇴장하면서 주 토큰을 넥스트 홀더에게 토큰 전송함에 따른 시간의 지연으로서 두 가지 요소로 구분된다. 첫 째는 네트워크 전송 지연(Pt)이며, 두 번째는 컴퓨터 처리시간 지연(Ps)이다. 전체 지연시간을 P라 하면 P=Pt+Ps이다. 본 알고리즘의 동기화 지연은 Ps는 무시할 정도의 시간이므로 네트워크 전송지연 Pt의 최대치 *t*로 평가된다.

4.1.3 동시진행도

동시진행도는 하나의 세션은 운영 중이며, 어떤 세션은 다음 순서를 대기하는 상황에서 현재 세션에 참여할 수 있는 최대 개수에 의하여 정의된다. 따라서 높은 정도의 동시진행도는 보다 효율적인 자원의 활용을 뜻하며, 그룹 상호배제 알고리즘의 최대 동시진행도는 n이 시스템 내의 프로세스의 전체 수일 때 n이 된다.

본 알고리즘은 시스템 범위는 4지, 왕복 4차선의 교차로를 가정하고 있으며, 교차로 내에는 8개의 진입차선이 방향별로 존재한다. 교차로 내에 있는 차량의 수는 전술한 바와 같이 주 토큰 홀더가 주 토큰을 받는 시점에 컷오프 한 차량의 수 n으로 계산한다. 따라서 시스템 내의 차량들은 각 차선으로 나누어져 존재하게 되며, 각 차선에 동일한 량의 차량이 존재한다고 가정하면 차선별로 $n/8$의 차량이 있게 된다. 이때 본 알고리즘의 최대 동시진행도는 $n/8$이 되지만, 교차구역 진입은 주 토큰 홀더 체인과 부 토큰 홀더 체인의 두 개의 차선이 동시에 하게 되므로 본 알고리즘의 동시진행도는 $n/4(=2n/8)$가 된다. 따라서 교차로 교통 통제에 적용된 본 알고리즘의 동시진행도는 일반 그룹 상호배제 알고리즘에 비해 낮은 수준이다.

<표 4.1> 그룹 상호배제 알고리즘 성과 비교

구 분	메시지 복잡도	동기화 지연	동시진행도
VTokenIC	$1.125n+5$	t	$n/8$
Mamun	$n+2$	$2t$	n
Mittal	$2n-1$	t	n
Weigang	$3n$		$n/8$
Kakugawa	$5Q+1$	$3-4t$	n

* n : 프로세스의 수, Q : quorum, t : 최대 메시지 지연
(Hirotsugu Kakugawa, Sayaka Kamei, Toshimitsu Masuzawa, 2008)

4.2 시뮬레이션 결과

이 섹션에서는 본 알고리즘의 교통적인 측면에서 시스템 처리량인 교통처리량과 차량들의 차선에서의 대기시간 및 대기길이에 대한 시뮬레이션 결과를 제시한다. *VTokenIC*의 차량의 흐름을 시험하기 위하여 MOVE, SUMO, GLOMOSIM으로 구성된 VANET 시뮬레이터와 ns-3을 적용한 기존 관련 알고리즘들의 결과 값들을 참고로 해서 컴퓨터 프로그램을 작성하여 실험한다. 교차로는 100m*100m 이며, 통신기기의 전송범위는 200m이고, 통신 프로토콜은 애드호크 모드를 적용한다. 네트워크는 원 홉이므로 라우팅 규약은 필요하지 않다.

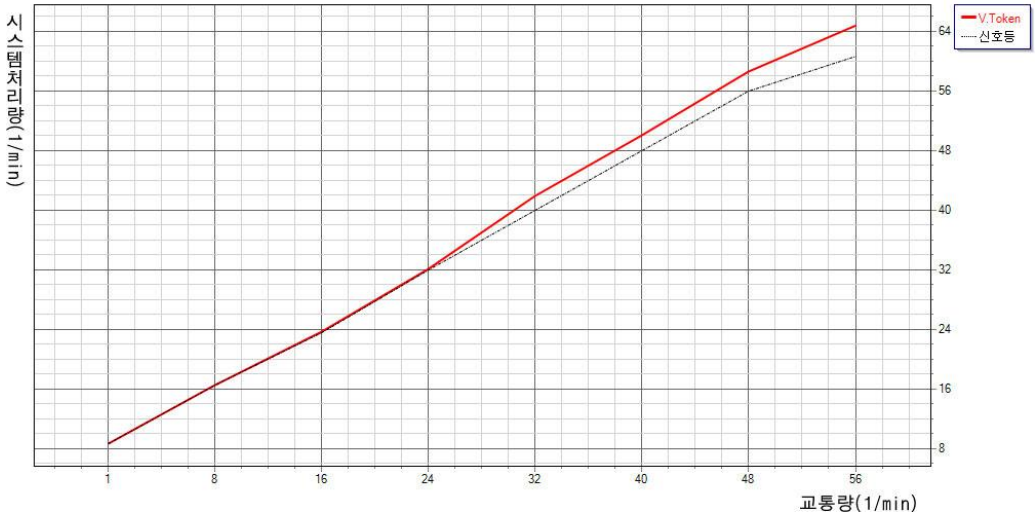
본 시뮬레이션은 대부분의 교차로 교통통제 연구에서와 마찬가지로 상이한 교통량 수준에서 성과를 산출하기 위하여 교차로의 volume to capacity를 변화시킨다. 또한 교차로의 용량은 64대/분으로 정하고, 교차로에 진입하는 전체 차량의 수는 8대에서 64대 까지 변화시킨다((Weigang Wu, Jiebin Zhang, Aoxue Luo, 2015). 또한 교차로 내의 차선별 교통량은 동일한 도착율과 무작위 도착율 두 가지를 구분해서 실험한다. 차선별 차량의 수는 전체 차량의 수를 무작위로 20가지 경우의 수를 작성하여 실험한다.

4.2.1 시스템 처리량

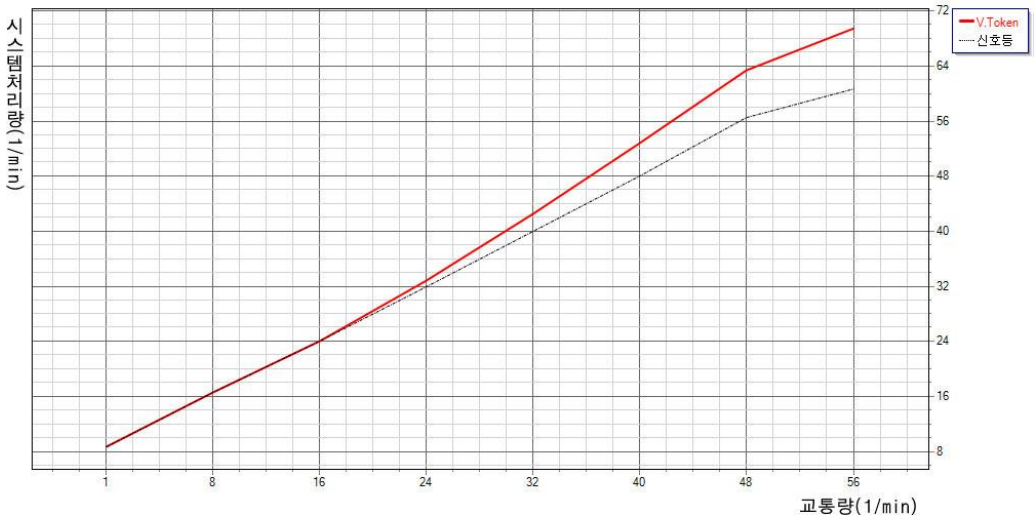
시스템 처리량의 결과는 <그림 4.1>과 <그림 4.2>에 나타난다. 교통량의 증가함에 따라 교통처리량도 증가한다. 본 알고리즘은 동일한 도착율의 경우 1분당 66대, 무작위 도착율의 경우 1분당 70대까지 포화상태 없이 처리한다. 이와 같이 교통 처리량의 경우 무작위 도착율의 경우가 동일 도착율

의 경우보다 도 많은 교통 처리량을 나타낸다. 이는 번잡한 차선의 차량들은 동시진행의 기회가 더 크기 때문이다.

또한, 본 알고리즘은 동일 도착율의 경우에 신호등 보다 교통 처리량이 많다. 이는 본 알고리즘은 신호등의 경우와 달리 일정한 시간을 기다리지 않고 해당 차선 체인의 차량이 다 통과하면 다음 순서의 차선으로 진행이 변경되므로 회전이 빠르기 때문이다.



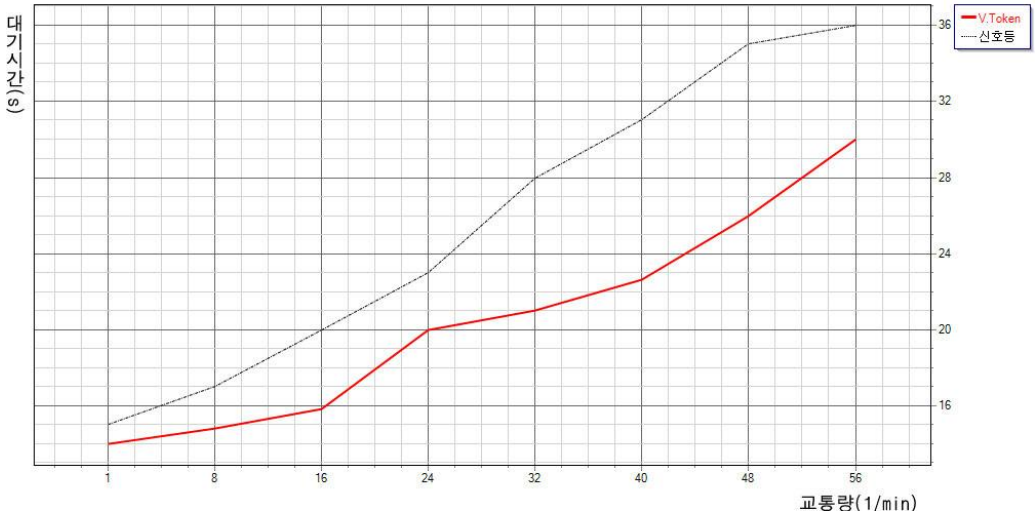
<그림 4.1> 시스템 처리량(동일 도착율)



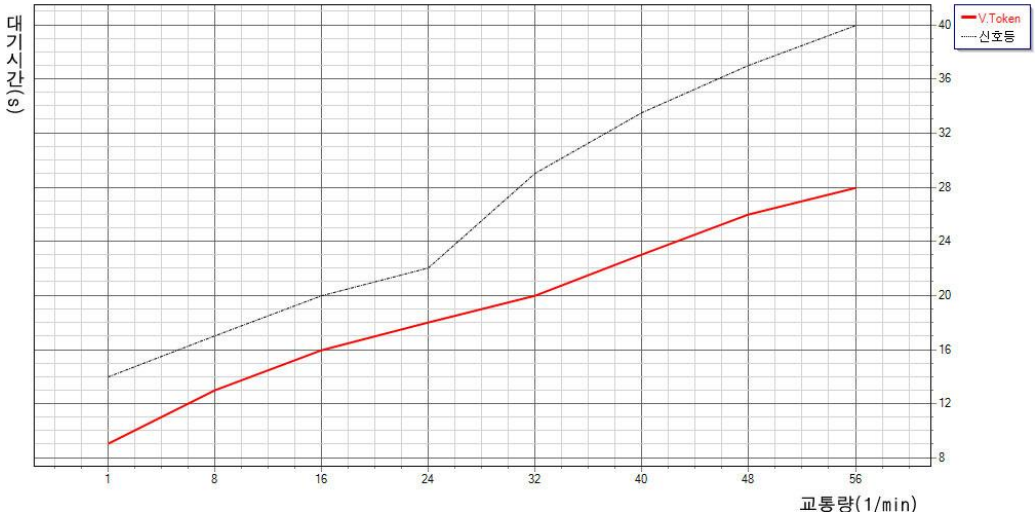
<그림 4.2> 시스템 처리량(무작위 도착율)

4.2.2 평균 대기시간

<그림 4.3>과 <그림 4.4>는 평균 대기시간의 결과를 나타낸다. 대기시간은 교통량이 증가함에 따라 증가한다. 본 알고리즘의 대기시간은 5-30초 사이이다. 이를 교통 신호 알고리즘과 비교했을 때 본 논문의 분산 알고리즘이 보다 짧은 대기시간을 유지한다. 또한 무작위 도착율의 경우의 대기시간이 교통 처리량의 경우와 동일한 이유로 동일 도착율의 경우 보다 짧다.



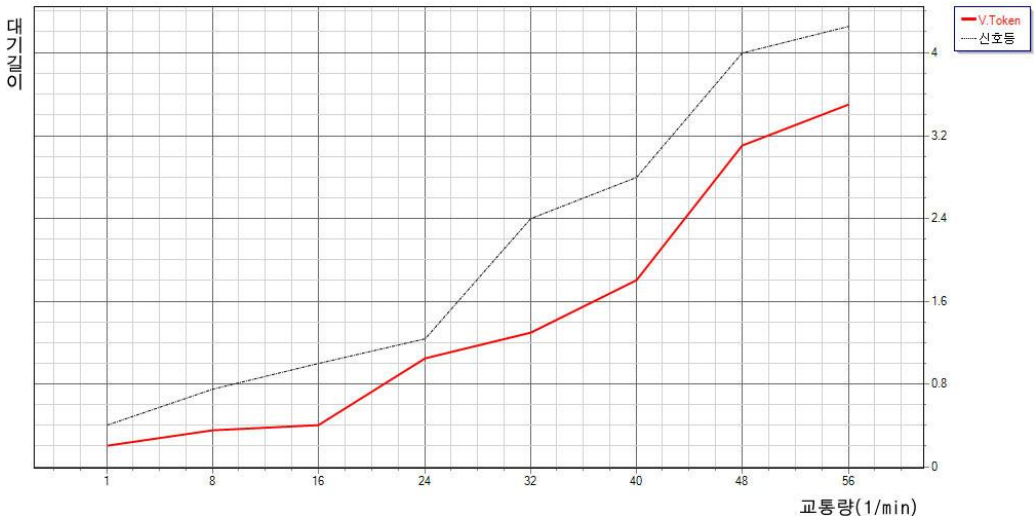
<그림 4.3> 평균 대기시간(동일 도착율)



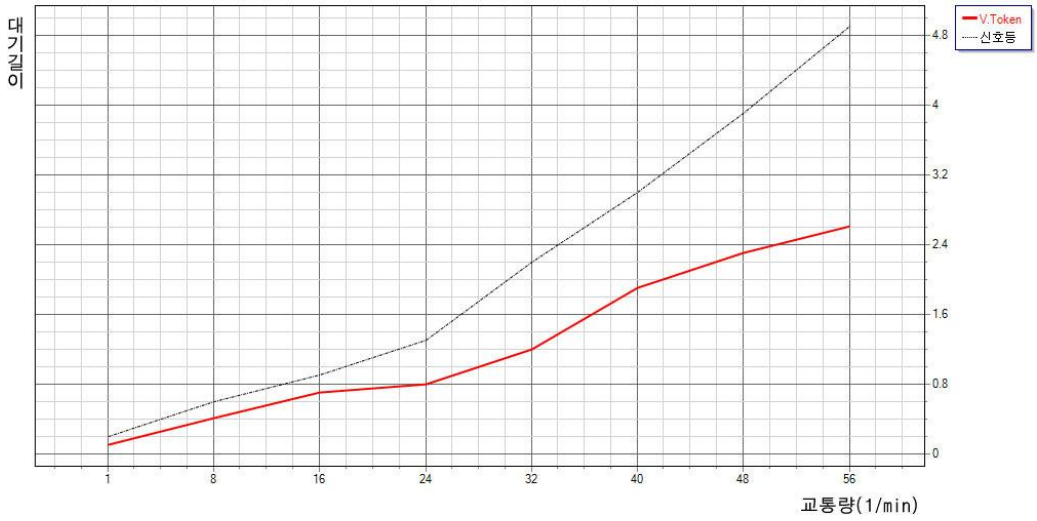
<그림 4.4> 평균 대기시간(무작위 도착율)

4.2.3 평균 대기길이

평균 대기길이의 실험 결과를 보면 교통량의 증가에 따라 대기길이도 증가한다. 이는 많은 차량이 교차구역을 통과하기를 원하면 진입차선에서 대기하는 차량이 많아 대기길이가 증가한다. 평균 대기길이의 결과는 <그림 4.5>와 <그림 4.6>에 나타나며, 무작위 도착율의 경우 평균 대기시간에서와 마찬가지로 동일한 도착율의 경우 보다 짧다. 또한, 무작위 도착율의 경우 대기길이는 교통량이 늘어날수록 교통 신호등과의 차이가 커진다.



<그림 4.5> 평균 대기길이(동일 도착율)



<그림 4.6> 평균 대기길이(무작위 도착율)

V. 결론

현재 진행되고 있는 교차로 교통 통제에 대한 연구인 교통신호 스케쥴링 최적화 및 자율주행 차량의 궤적관리 연구는 컴퓨터 공학의 클라이언트 서버와 흡사한 중앙 통제장치 기반이다. 이는 하나의 장치에 의하여 통제가 되므로 이러한 장치의 결함에 영향이 크고, 장치의 변경에 대한 유동성이 적은 문제점을 지니고 있다. 이를 해결하기 위하여 본 논문은 자율주행 차량들의 통신만을 통하여 교차로를 안전하게 통과할 수 있는 자율주행 차량의 교차로 트래픽 제어 시스템을 설계하였다.

중앙통제 장치가 없는 교차로 교통 통제 시스템에서는 차량들은 서로간의 메시지 교환을 통하여 교차로 통과 권한에 대한 경쟁을 하며, 교차로에서의 안전성 확보를 위해서는 차량 간의 정보교환으로 완벽한 협업이 이루어져야 한다. 여기서 본 논문은 효율적인 교통 흐름의 통제 및 메시지 트래픽을 감소시키기 위하여 진입차선 체인을 활용한다. 진입차선 체인은 진입차선의 캡틴이 팔로우어들에게 메시지를 전송하여 형성되는 일련의 차량의 클러스터(집단)이며, 향후 교차구역 진입 및 퇴장을 차량이 연결된 체인 단위로 하게 된다. 즉, 토큰 하나로 교차구역에 동시에 진입할 차량의 집단의 체인을 관리함으로써 메시지 트래픽을 감소시켰으며 또한 교통 흐름의 효율성을 증진시켰다.

또한, 본 논문은 교차로를 통과한 차량은 교차로 내에 체류하지 못하고 나가야 하는 동적 상황에서 교통 흐름을 증가시킨다. 이를 위하여 클러스터

형태로 진입한 차량들에 대하여 토큰의 유기적인 순환을 통하여 넥스트 홀더에게 토큰이 전달될 수 있도록 알고리즘을 설계하였다. 이와 더불어 본 논문은 주 토큰과 부 토큰 두 종류의 토큰을 사용함으로써 시스템의 효율을 극대화 하였다. 주 토큰은 하나의 차선의 캡틴에게 주어지며, 주 토큰은 교차로 통과의 효율성 증가를 위하여 보조 수단인 부 토큰을 생성하여 교차구역에 동시에 진입할 수 있는 차선 체인의 캡틴에게 전달한다. 이러한 형태의 자율주행 차량의 교차로 통제 시스템 설계를 위하여 핵심이 되는 것은 교차로에서 안전성을 어기지 않으면서 교통 흐름을 원활하게 할 수 있도록 주 토큰을 순환하게 하는 것과 부 토큰을 생성하고 조정하는 것이다. 본 논문은 주 토큰 홀더가 교차구역을 나갈 때에 넥스트 홀더가 없을 때에는 주 토큰을 자신의 체인의 마지막 차량과 부 토큰 홀더 체인의 마지막 차량에게 임시로 전달하여 기다림으로써 교통 처리량을 증가시켰다.

본 알고리즘의 시뮬레이션을 통한 성과 평가는 적은 메시지 비용으로 교통량의 변화에 따른 교통 상황에 맞추어 교통량을 잘 처리할 수 있음을 보여준다. 또한 우리의 연구는 기존 신호등을 통한 시스템과 비교할 때 교통 처리량 및 대기시간 등에 있어서 효율성이 높음을 알 수 있다. 더욱이 본 연구는 신호등과 ICU 같은 중앙통제 장치가 없는 것을 전제로 주 토큰의 순환에 집중하여 설계하였기 때문에 보다 많은 차선의 교차로뿐만 아니라 신호등이 없는 작은 교차로에서 언제든지 적용이 가능하다.

본 연구는 자율주행 차량의 교차로 교통통제 연구의 시작 단계이며, 향후의 무인 자율주행 자동차의 시대에 필요한 기반 기술로서 의미가 있다. 본 연구에서는 교차로의 임계구역인 교차구역을 하나의 유닛으로 구성하여 상호배제를 적용하였다. 그러나 교차로에서의 교통 통제의 시스템 개선을 위

해서는 교차구역을 여러 개의 유닛으로 세분화하고, 차량의 속도의 개념을 추가 적용하여 연구할 필요가 있다. 또한, 이 이외에 추가로 연구해야 할 부분으로 메시지 비용을 추가로 감소할 수 있는 방법과 교차로 차선에 센서 설치를 통한 정보 활용방안이 있을 수 있다. 또한 교차로 이외의 다른 영역에 적용 가능한 개념의 확장(비행기 활주로 등) 등 이슈가 있다.

참고 문헌

Weigang Wu, Jiebin Zhang, Aoxue Luo, and Jiannong Cao (2015). "A Distributed mutual exclusion algorithm for intersection traffic control" *IEEE Transactions on parallel and distributed systems*, vol. 26. No1, January 2015

Fei Yan, Mahjoub DRIDI, Abdellah el Moudni (2013). "An autonomous vehicle sequencing problem at intersections: A genetic algorithm approach" *Int.J.Appl. Math. Comput. Sci*,. 2013, No1, 183-200.

Md. Abdus Samad Kamal, Jun-ichi Imura, Tomohisa Hayakawa, Akira Ohata (2015) "A vehicle-Intersection Coordination Scheme for Smooth flows of traffic without using traffic lights". *IEEE Transactions on parallel and distributed systems*, vol. 16. No3, June 2015

Li Li, Fei-Yue Wang (2006) "Cooperative driving blind crossing using intervehicle communication". *IEEE Transactions on parallel and distributed systems*, vol. 55. No6, November 2006

Xiangjun Qian, Jean Gregoire, Fabien Moutarde, Arnaud De La FortelleCañete, (2014). coordination of autonomous and legacy vehicles at intersection". *17th International IEEE conference on*

Intelligent Transportation Systems, Oct 2014

Joyoung Lee, Byungkyu Park (2012). "Development and Evaluation of a Cooperative vehicle Intersection control algorithm under the connected vehicles environment", *IEEE Transactions on parallel and distributed systems*, vol. 13. No1, March 2012

Feng Zhu, Satish V. Ukkusuri (2015). "A linear programming fomulation for autunomous intersection control within a dynamic traffic assignment and connected vehicle environment", *Transportation Research part C* 55 (2015) 363-378

Ismail H. Zohdy, Hesham A. Rakha (2016). "Intersection management via vehicle connectivity: The intersection cooperative adative cruise control system concept", *Journal of Intelligent transportation system* 20(1):17-32, 2016

Karima Benhamza, Hamid Seridi (2015). "Adaptive traffic signal control in multiple intersections network". *Journal for Intelligent & Fuzzy systems* 28 (2015) 2557-2567

L.A Prasanth, S. Bhatragar (2011). "Reinforcement learning with function approximation for traffic signal control", *IEEE Transactions on parallel and distributed systems*, vol. 12. No2, June 2011

Quazi Ehsanul Kabir Mamun, Hidenori Nakazato (2006). “A new token based Protocol for Group mutual exclusion in distributed systems”, *Proceedings of the fifth international symposium ton parallel and distributed computing*(ICPDC 06)

Neeraj Mittal, Prajwal K. Mohan (2007). "A priority-based distributed group mutual exclusion algorithm when group access is non-uniform“. *J. Parallel Distrib. comput.* 67 (2007) 797-815

Hirotsugu Kakugama, Sayaka Kemei, Toshimitsu Masuzawa (2008). "A token based distributed group mutual exclusion with quorum“, *IEEE Transactions on parallel and distributed systems*, vol. 19. No9, Sep 2008

Priyamvadha Thambu, Jonny Wong (1995). “An efficient token-based mutual exclusion algorithm in a distributd system” *J. systems software* 1995; 28:267-276

Glenn Ricart, Ashok K. Agrawala (1981). “An optimal algorithm for mutual exclusion in computer networks”, *Communications of the ICM,* January 1981: volume 24, number 1

Yuh-Jzer Joung (2003). “Quorum-based algorithms for group mutual algorithm”, *IEEE Transactions on parallel and distributed systems*, vol. 14. No5, May 2003Journal

Ranganath Atreya, Neeraj Mittal, Sathya Peri (2007). “A quorum-based group mutual exclusion algorithm for a distributed system with dynamic group set”, *IEEE Transactions on parallel and distributed systems*, vol. 18. No10, Oct 2007

Ranganath Atreya, Neeraj MittalM (2005). “ A dynamic group mutual exclusion algorithm using surrogate-quorums”, 25th *IEEE International conference on distribured computing systems*(ICSCS ‘05)

K. M. Chandy, J. Miara (1984). “The drinking philosophers problem", *ACM Transaction on programming languages and systems*, vol 6, No 4, October 1984, page 632-646

Abhishek Swaroop, Awadhesh Kumar Singh (2013). "A hybrid algorithm to solve group mutual exclusion problem in message passing distributed systems" *International Journal of computer applications*(0975-8887) volume 67-No9 April 2013

Mamotu Maekawa (1985), "A root N algorithm for mutual exclusion in decentralized systems" *ACM Transaction on programming languages and systems*, vol 3, No 2, May 1985, page 145-159

Kerry Raymond (1989), “A tree-based algorithm for distributed mutual exclusion” *ACM Transaction on programming languages and systems*, vol 7, No 1, Feb 1989, page 61-77

Leila Hernane, Jens Gustedt, Mohamed Benyettou (2012), "A dynamic distributed algorithm for read and write locks", 2012 20th *Euromicro International conference on parallel, Distributed and network-based processing*

Claus Wagner, Frank Mueller , "Token-based read/write locks for distributed mutual exclusion", (+49) (30) 2093-301

Abhishek Swaroop, Awadhesh Kumar Singh (2009), "Efficient group mutual exclusion protocols for message passing distributed computing systems", *Department of computer engineering national institute of technology*. Kurukshetra, Haryana, India, Pin-136119, Oct, 2009

Lamport, L. (1978), "Time, clocks, and the ordering of events in a distributed system", *Communications of the ACM*, vol 21, pp.558-565

Wu, M.Y., Shu, W., (2002), " An efficient distributed token-based mutual exclusion algorithm with central coordinator". *Journal of parallel and distributed computing* vol 62, no 10, pp 1602-1613, 2002

Chang Y.I., Singha, M., Liu, M.T. (1991), "A dynamic token-based distributed mutual exclusion algorithm" *In the proceedings of the 10th annual international phoenix conference on computers*

and the communications, pp 240-246 1991

Bertier, M., Arantes,L., Sens, P., (2006), "Distributed mutual exclusion algorithms for grid applications: a hierarchical approach", *Journal of parallel an distributed computing*, vol 66, no 1, pp 128=144

Swaroop, A., Singh, A,K., (2007), "A token-based for fair algorithms for group mutual exclusion in distributed system" *Journal of computer science*, vol 3, no 10, pp 829-835

Joung, Y,J., (2002). "The congenial talking philosophers problem in computer networks". *Distributed computing*, vol 15, no 3, pp 155-175, 2002i

Park, J.H., Gang S., Kim, K., (2003), "Group mutual exclusion based secure distributed protocols" *Proceedings of the Hawaii International Conference on System Science*, pp.1-10.

Cantarell, S., Dutta, A,K., Pilit, F., Villain, V., (2001). "Token-based group mutual exclusion for asynchronous rings", *In the proceedings of IEEE international conference on distributed computing systems*, pp 691-694, 2001I

Manivannan,D., Singhal, M.A. (1996), "A decentralized token generation scheme for token based mutual exclusion algorithms", *International journal of computer systems science*

and engineering, vol11, no 1, pp45-54